City Indicator Consistency Analysis: An Approach to Inconsistency Detection

by

Yetian Wang

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering
University of Toronto

City Indicator Consistency Analysis: An Approach to Inconsistency Detection

Yetian Wang

Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering
University of Toronto

2017

Abstract

Cities use a variety of metrics to evaluate and compare their performance. The ISO 37120 standard provides a definition for city indicators that measure a city's quality of life and sustainability. A problem that arises in indicator-based comparisons, is whether the comparison is invalid due to inconsistencies in the data used to derive them? In this thesis we present three types of inconsistency analysis for automating the detection of inconsistencies in open city data. Namely, definitional consistency analysis that evaluates if data used to derive a city indicator is consistent with the indicator's definition (e.g., ISO 37120); transversal consistency analysis that evaluates if city indicators published by two different cities are consistent with each other; and longitudinal consistency analysis that evaluates if an indicator published by a city is consistent over different time intervals. City indicator consistency analysis enable the possibility of consistently measuring and comparing performances of cities.

Acknowledgments

I would like to thank my supervisor, Professor Mark S. Fox for the constant support and guidance throughout this research and during the preparation of this thesis. His knowledge and patience steered me in the right direction whenever I needed guidance.

I would also like to thank Dr. Daniela Rosu for her valuable comments.

Finally, I would like to thank my parents, Zhifeng Wang and Hong Zheng, and my wife, Huangyuan Wu for their support and encouragement.

Table of Contents

List of Tables

List of Definitions

List of Figures

List of Appendices

Chapter 1
Introduction

1 Introduction

Cities use a variety of metrics to evaluate and compare their performance. With the introduction of ISO37120, which contains 100 indicators for measuring a city's quality of life and sustainability, it is now possible to consistently measure and compare cities. A problem that arises in indicator-based comparisons, is whether the comparison is invalid due to inconsistencies in the data used to derive them?

The goal of PolisGnosis project (Fox, 2015) is to construct an intelligent agent that can diagnose a city's performance. It will automate the longitudinal analysis, i.e., how and why a city's indicators change over time, and transversal analysis, i.e., how and why cities differ from each other, in order to discover the root causes of differences. The agent must satisfy the following requirements:

1. Indicator Independence. Since there are a vast number of indicators used by cities, beyond those defined in the ISO 37120 standard, and ISO standards evolve over time, we do not want our agent to have any knowledge of indicator definitions "hardwired" into its code. An indicator's definition must be an input to the agent.

2. City Independence. Cities openly publish vast amounts of data that that our agent would like to use. But the data lacks any standard models or vocabularies - every dataset differs in structure, attributes and content. It would be practically impossible to construct an agent that can understand these datasets. Hence the agent will assume that cities will adopt or translate into a standard for representing the information used to derive its indicators.

3. Analysis Independence. Given the variety of indicators and the ensuing variety of data used to derive them, a variety of methods of analysis may also be required. Rather than hardwire these methods into PolisGnosis, it would be better if they too were inputs to the agent.

To achieve indicator and city independence, indicator definitions, the indicator values and the data used to derive them must be represented using a standard representation, and used as input to the agent. The Global City Indicator Ontologies (GCIO) (Fox, 2015) provide standard representations in the form of ontologies for many of the indicator themes in the ISO 37120 standard [1].

Before diagnosis can be performed, the city indicators must be evaluated to be consistent in order to attain meaningful comparison. City indicators with inconsistent definitions are incomparable.

City indicators represented using ontologies can be evaluated using existing ontology consistency checkers and reasoners to verify its logical consistency. That is, there exists at least one model that satisfies all axioms of the ontology. In addition to logical consistency, we also need to evaluate different types of consistency for city indicators in the context of knowledge about indicator values, definitions, theme and city specific indicator knowledge. We are interested in verifying if the published indicators are indeed following the indicators' definitions. For example, ISO 37120 stated that absolute homelessness is defined to be those who lives outside or in a temporary emergency shelter. Thus if a city publishes indicator data that involved homeless population (e.g. homeless population size per 100 000 population) must refer to the types of homeless person within the scope of the definition of homelessness in order to be consistent with the definition. The indicators also need to be evaluated between cities (e.g. definition of homelessness are the same between cities) and compared over time within a city (e.g. definition of homelessness remain unchanged) when the results of city indicators are compared transversally and longitudinally.

City indicator consistency is a problem of evaluating both consistency of the representation of indicator (its definition and theme specific knowledge) and the information it carries (measure and meta-information). We define three types of consistencies of city indicator definitions depending on the definitions it is evaluated with respect to:

[1] See http://ontoogy.eil.utoronto.ca for the complete list of ontologies.

- Definitional consistency evaluates if data used to derive a city indicator is consistent with the indicator's definition (e.g. ISO 37120). For example, if the indicator is a student/teacher ratio, then a city's reported indicator is inconsistent if it includes teachers that do not satisfy the ISO 37120 definition, e.g., administrative staff.

- Transversal consistency evaluates if city indicators published by two different cities are consistent with each other. For example, if the indicator measures homeless population size per 100 000 population, then indicators are transversally inconsistent if the homeless definition used by each differs. Note that each city's indicator can be definitional consistent but not transversal consistent.

- Longitudinal consistency evaluates if an indicator published by a city is consistent over different time intervals. For example, if the indicator measures a city's PM10 air pollution, then the indicator is longitudinally inconsistent if the geospatial dimensions of the city have changed, which may arise through amalgamation.

When a city such as Toronto publishes its indicator data, it publishes a set of indicator instances as well as city specific knowledge such as the definition of homeless person and homeless shelter represented using separate ontology. City indicator consistency requires the evaluation of the consistency of Toronto's city specific knowledge with respect to the standard (e.g. GCI ontologies), city specific knowledge of another city (e.g. New York's definition of homeless shelter), and different versions of the same indicator (e.g. Toronto's definition of homeless shelter in 1998) when it comes to definitional, transversal and longitudinal consistency evaluation respectively.

This thesis defines a set of definitional, transversal and longitudinal inconsistencies and a process for detecting them in city indicator values and the supporting data used to derive them. Given an indicator definition, an indicator's value and supporting data, both represented using the GCIO, the equivalent graph representation is analyzed to detect inconsistencies. The algorithm performs sub-graph matching to detect mismatches such as temporal, geospatial, measurement, and population definition differences.

1.1 Summary of Contribution

The main contributions of this thesis are as follows:

1. Formally defining definitional, transversal, and longitudinal inconsistencies for city indicators
2. Providing an implementation of inconsistency detection in Prolog.

1.2 Overview of Dissertation

The chapters of this thesis are outlined as follows:

Chapter 2 provides background information on existing city indicator standards, ontologies within the domain of city indicator representation, types of consistency and consistency evaluation tools such as the Protégé reasoners.

Chapter 3 introduces definitional consistency analysis and defines a set of definitional inconsistencies in terms of type, temporal, geographical, and measurement inconsistency. Correspondence between indicator values and supporting data and indicator's definition is also defined. First Order Logic (FOL) axioms are provided for each type of inconsistency as well as a brief example.

Chapter 4 introduces transversal and longitudinal consistency analysis and defines a set of transversal and longitudinal inconsistencies in terms of type, temporal, geographical, and measurement inconsistency. Correspondence between two sets of indicator values and the supporting data and the notion of prime and secondary classes is also defined. Similar to Chapter 3, First Order Logic (FOL) axioms and examples are provided for each type of inconsistency.

Chapter 5 introduces the implementation City Indicator Consistency Checker (CICC) which was implemented using SWI-Prolog. Examples of definitional, transversal and longitudinal consistency analysis are provided in this chapter using indicator values and the supporting data published by Toronto and New York City which were represented as instances of ISO 37120 15.2 indicator's definition which uses the GCIO and city specific ontologies created for demonstration purpose.

Chapter 6 provides conclusion for this dissertation and discusses possible future research work.

Chapter 2
Background

2 Background

This chapter is divided into three parts. The first part reviews city indicators that have been developed by various organizations. The second part reviews ontologies that have been developed for representing city indicator definitions and the data used to derive them. The third part reviews ontology-based methods for determining the consistency of information.

2.1 City Indicators

In 2007, it was recognized that "there are thousands of different sets of city (or urban) indicators and hundreds of agencies compiling and reviewing them. Most cities already have some degree of performance measurement in place. However, these indicators are usually not standardized, consistent or comparable (over time or across cities), nor do they have sufficient endorsement to be used as ongoing benchmarks." (Hoornweg et al., 2007). For example, in (Hoornweg et al., 2007, p 32), the WHO list of city indicators related to homeless population size was defined to be "estimated number of homeless people" without an exact definition of 'homeless people' nor what method should be used for estimation. Cities who publish indicator data with ambiguous definition provide little value in comparing and evaluating city performance. One city might include only homeless people who live on the street in their data while another city includes people that live in shelters as well. A number of approaches have been made to standardize the definitions of city indicators across cities around the world.

2.1.1 ISO 37120 Standards

The Global City Indicators Facility (GCIF) at University of Toronto developed a set of city indicators with clear and precise definitions (Global City Indicators Facility, 2010a; McCarney, 2011). These indicators formed the basis of the ISO 37120 "Sustainable development of communities – Indicators for city services and quality of life" standard. The standard contains 100 indicators covering themes such as Education, Energy, Health, Safety, Finance and Shelter (ISO37120, 2014). For example, the ISO 37120 standard defines the indicator "The number of homeless per 100 000 population" as follows:

> "The number of homeless per 100 000 population shall be calculated as the total number of homeless people (numerator) divided by one 100 000th of the city's total population (denominator). The result shall be expressed as the number of homeless per 100 000 population.
>
> The following definition is used by the United Nations to define homelessness: Absolute homelessness refers to those without any physical shelter, for example, those living outside, in parks, in doorways, in parked vehicles, or parking garages, as well as those in emergency shelters or in transition houses for women fleeing abuse." (ISO37120, 2014).

ISO 37120 indicator definitions reduce possible ambiguities of interpretation by cities, leading to greater consistency in measurement and comparability across cities.

2.1.2 City Anatomy

City Protocol is a framework for cities to improve performance that benefit citizens and their quality of life (CPA, 2015a). It seeks to define a common systems view for cities around the globe, and then embraces or develops protocols that will help to break city silos. City Protocol aims at working across diverse cities by interconnecting them and finding common solutions.

Data Interoperability and City Indicators (DICI) is a project from City Protocol that aims at creating data interoperability protocols to enable transparent and smart data sharing among cities and citizens (CPA, 2015a). It extends ISO 37120 indicators with additional indicators that conform the schema defined by the City Anatomy (CPA, 2015b). City anatomy is an organizing framework that builds a platform and tools to support the evaluation of city performances with the following elements:

- Structure - a set of physical structures
- Society - the living entities that make up a city's society, and
- Interactions - and the flow of interactions between them

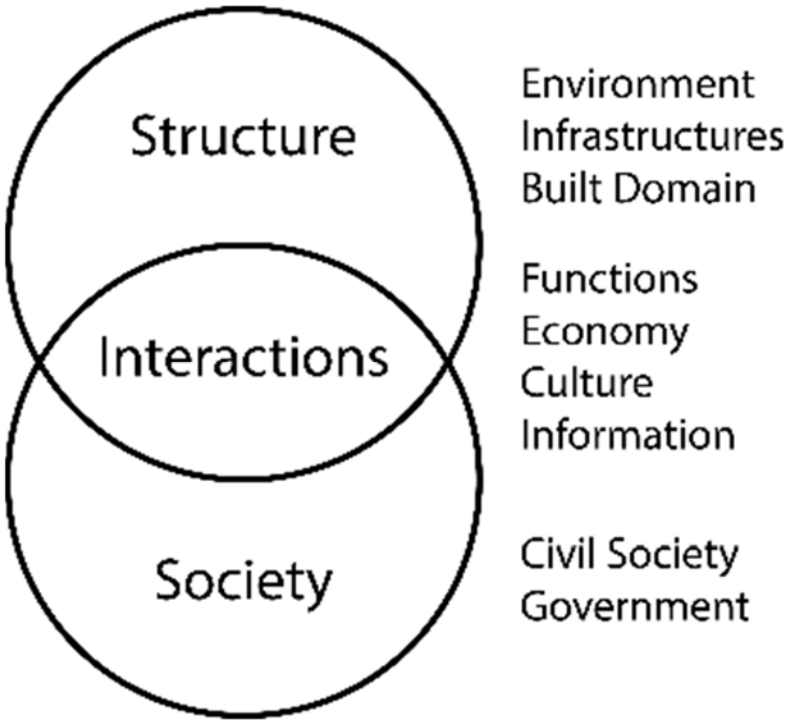


Figure 1 Elements of City Anatomy, adapted from CPA (2015a)

For example, City Protocol extends ISO 37120 indicators in categories such as education, shelter, and transportation with new indicators such as the following: Percentage of social housing, Percentage of empty housing, Percentage of housing ownership, Percentage of housing for rent, Proximity to convenience shopping, and Percentage of subscribers to city sports facilities. These indicators provided by City Protocol extends the indicators "to function like living, working or shopping" (CPA, 2015b).

2.1.3 The Organization for Economic Co-operation and Development (OECD)

The Organization for Economic Co-operation and Development (OECD)[2] is a forum that provides a platform to help governments to improve and coordinate policies in terms of economic growth, financial stability, and social development and develops solutions to common problems of member countries. OECD provides a number of indicators for various areas such as

[2] www.oecd.org

health, education, environment and trade. Indicators in each category were further classified into specific subjects. For example, the OECD Main Economic Indicators was classified into subjects such as production, sales, international trades, Composite Leading Indicators, etc. Each indicator is described in English along with methods used to collect and compile the results of the indicators from individual countries (OECD, 2013).

The indicators' definitions were not published in Sematic Web compatible formats, but their statistical data were published following the Statistical Data and Metadata eXchange (SDMX)[3] which is a standard used to publish raw statistic data and metadata. Some of OECD's data was used in the research of Capadisli et al., (2013) where an effort was made to transform raw statistic data and metadata into RDF representation. RDF data Cube vocabulary, which builds upon SKOS[4], SCOVO[5], etc. (Cyganiak & Reynolds, 2014), was implemented to described multi-dimensional statistic data and PROV-O[6] ontology was used for provenance information (Capadisli et al., 2013).

2.1.4 IBM Smart Cities Initiative

IBM Institute for Business Value as part of the IBM Smart Cities initiative proposed a solution for measuring performances of cities and compare the result across different cities from the globe (Dirks, Keeling, & Dencik, 2009). IBM identified a number of core systems that operate as base systems of a city, e.g., city services, citizens, business, transport, communication, water and energy (Dirks & Keeling, 2009). Indicators such as cost of healthcare, number of days to start a business, congestion cost, car ownership rate, percentage of online population, etc., were introduced for the cities' core systems.

The IBM Intelligent Operations Center that supports the IBM Smart Cities initiative provides a key performance indicator (KPI) subsystem that identifies a set of KPIs, metrics, events, and

---

[3] https://sdmx.org/

[4] https://www.w3.org/2004/02/skos/

[5] http://sw.joanneum.at/scovo/schema.html

[6] https://www.w3.org/TR/prov-o/

conditions (Smith, 2011). The KPI subsystem includes indicators spanning categories such as public safety, transportation and water (Smith, 2013). As defined by Smith (2011), KPIs are "quantifiable measurements employed by organizations to monitor and assess performance. They help reduce complex organizational performance information into a consumable format that allows organizations to more easily assess performance by comparing KPIs to the organization's stated objectives". One shortcoming is that organizations have different indicators thus it is difficult to compare the indicators cross organizations. Examples for IBM KPIs are crime response time, firefighter injuries, and public safety budget (Smith, 2011).

2.2 City Indicator Ontologies

With standards of city indicators established we now need to translate these standards into machine-readable format that can be published on Semantic Web. In this section we review a number of ontologies created to represent information about city indicators. With these ontologies, cities are able to publish their indicator data in Semantic Web compatible format such as RDF and OWL. Cities publish indicator value and supporting data by creating instances of indicators, their metadata and supporting data[7].

2.2.1 PolisGnosis Project

The PolisGnosis project aims to "develop theories, embodied in software, to perform longitudinal analysis (i.e., how and why a city's indicators change over time) and transversal analysis (i.e., how and why cities differ from each other at the same time), in order to discover the root causes of differences". Information about the indicator represented by ontologies (indicated by Global City Indicator Ontologies in Figure 2) are taken as input to the PolisGnosis Analysis Engine along with a set of consistency and diagnosis axioms used to determine the root cause of change in indicators (Fox, 2015a).

[7] See http://ontoogy.eil.utoronto.ca for example of indicators published using ontologies that represent ISO 37120 standards.

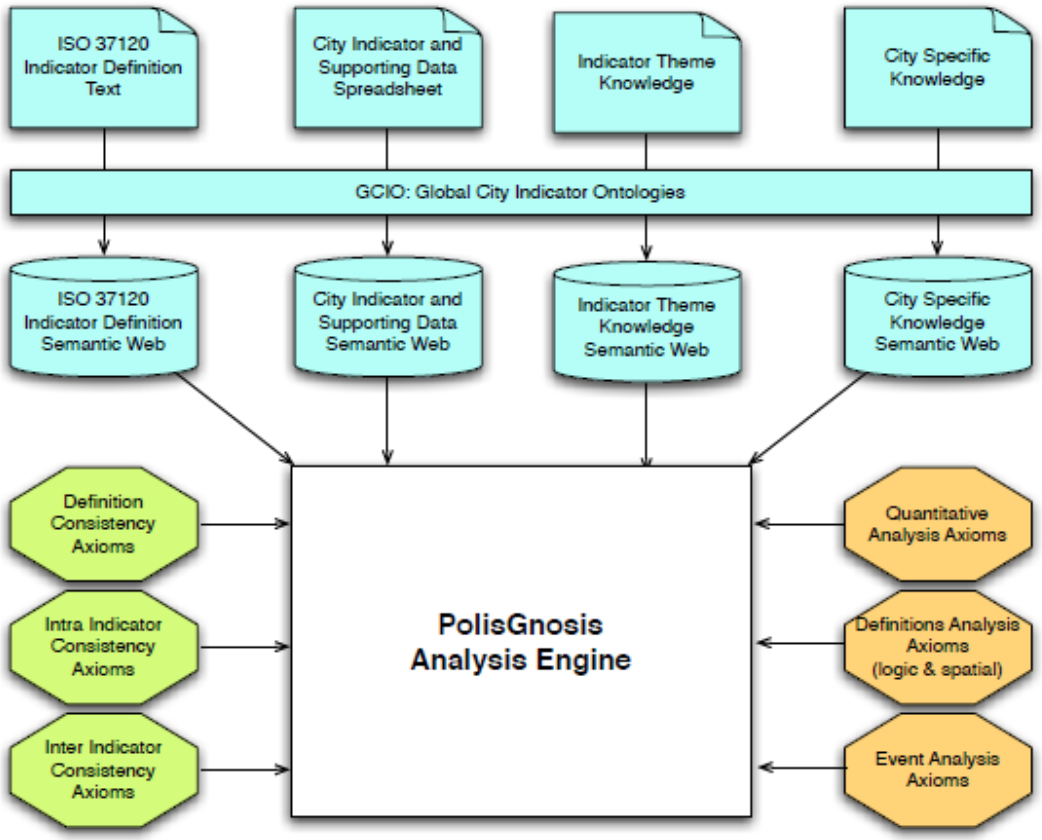


Figure 2 PolisGnosis Architecture, adapted from Fox (2015a)

The very first task is to develop "a set of ontologies that can be used to represent ISO 37120 indicators, their supporting data and theme specific knowledge". The objectives of these ontologies are summarized below as stated by Fox (2015a):

1. Represents the meta information of a "published indicator value such as its units, scale, year published and who published it".
2. Represents the "complete definition of each indicator in the ISO 37120 standard".
3. Represents "each city's indicator value (for a particular year), plus the supporting data used to derive it, using the definitions".
4. Represents "theme specific indicator knowledge", such as basic knowledge about shelter and housing, such as households, slums, homeless person, homeless shelters, etc.
5. The representation of a "city's theme specific indicator knowledge". For example, what is the definition of homeless person in the city? What type of shelters are considered as homeless shelters?

The ontologies created within the PolisGnosis project will be discussed in the following section and an overall structure of these ontologies in section 2.2.2.

2.2.2 Global City Indicator Ontologies

A number of Global City Indicators (GCI) ontologies have been developed for city indicator themes such as education (Fox 2015b), innovation (Forde & Fox, 2015), shelter (Wang & Fox, 2015), health (Falodi & Fox, 2015), finance (Wang Z. & Fox, 2016), and environment (Dahleh & Fox, 2016). Other themes such as transportation, safety, and energy are undergoing development process. Each themed ontology provides a set of class and properties that represent the definition of an indicator, its value and supporting data, as well as the theme specific knowledge required to analyse this indicator. For the purpose of our example that will be presented in later sections, we also briefly describe how ISO 37120 shelter indicators are defined.

GCI Foundation Ontology

GCI Foundation ontology was created by (Fox, 2013) to represent general fundamental concepts such as time, geographic location, statistics, and provenance. The concepts represented by GCI Foundation ontology provide a foundation for each theme specific GCI ontology such as GCI-Education (Fox, 2015b), GCI-Innovation (Forde & Fox, 2015), GCI-Shelter (Wang & Fox, 2015), and GCI-Health (Falodi & Fox, 2015), etc. GCI Foundation ontology provides the ability to represent meta information of a city's indicator value spanning the following concepts (Fox, 2015c):

- Placenames: unique identifiers for cities that the city or area that is measure by the indicator,
- Time: when an indicator is valid, or when it was produced,
- Measurement: quantities and units of indicators,
- Provenance: how an indicator was derived and by whom,
- Validity: the degree to which an indicator is believed to be correct, and
- Trust: the degree to which the individual or organization is trusted to produce an indicator correctly.

Placename

An area within a city can be represented with a number of ontologies. For the reasons stated in Fox (2015c), the GCIO selected Schema.org[8] and GeoNames[9] ontology to represent required geospatial information. Schema.org provides classes of placenames such as sc:City, sc:Country, and sc:State. It also provides classes for sc:GeoCoordinates (i.e., elevation, latitude, and longitude) and sc:GeoShape that represents the shape of an area as a polygon or circle. GeoNames ontology has the class geo:Feature which contains properties such as name, featureClass, featureCode, population, postalCode, nearbyFeatures, and wikipediaArticle, etc. Properties such as 'population' and 'postalCode' are data properties thus the information we can retrieve from geo:Feature is limited. Wang & Fox (2015) implemented Schema.org and GeoNames ontology in GCI Shelter Ontology by extending geo:'Feature' class with properties that relate geo:'Feature' to sc:'GeoCoordinates' and sc:'GeoShape'.

Time

In Fox (2015c), it was discussed that the temporal relationship between the indicator and its supporting data need to be represented with a time ontology that supports reasoning about time points, time intervals and the relationships among them. OWL-Time (Pan & Hobbs, 2004) was chosen to be embedded in the GCI Foundation ontology. More information about OWL-Time can be found in Pan & Hobbs (2004) and was summarized in Fox (2013). In general, OWL-Time represent time as a point (ot:Instant)[10] or an interval (ot:Interval). A ot:'DateTimeDescription' class was used to represent a date plus time using properties such as year, month, day, hour, etc.

Cities publish their data based on a specific time span (i.e., yearly, monthly, etc.) thus city indicators must be able to indicate the temporal period that the indicator was published for. This

---

[8] the Schema.org ontology is available at: http://schema.org/. We will use the prefix "sc:" to identify classes and properties from the ontology.

[9] www.geonames.org

[10] We will use the prefix "ot:" to identify classes and properties from the OWL-Time ontology

can be achieved via the property gci:'for_time_interval'[11] or appropriate subproperties that relates the ISO 37120 indicator to a valid 'DateTime' (e.g. ot:DateTimeInterval class from OWL-Time).

In OWL-Time (Pan & Hobbs, 2004), the class ot:DateTimeInterval is linked to the class ot:'DateTimeDescription' has the following properties: unitType, year, month, week, day, dayOfWeek, dayOfYear, hour, minute, second, and timeZone. Where unitType has one of (owl:oneOf) the following values: :unitSecond :unitMinute :unitHour :unitDay :unitWeek :unitMonth :unitYear which specifies the temporal unit type of an individual. Pan & Hobbs (2004) provides examples for temporal unit type: the temporal unit type of 10:30 is minute (unitMinute), for March 20, 2006 it is day (unitDay), also 2013 has the temporal unit type of year (unitYear). Therefore an interval must have the unitType property with a value unitYear and 'year' property with a value '2013' in order to represent the year of 2013.

Relation	Symbol	Symbol for Inverse	Pictoral Example
X *before* Y	<	>	XXX YYY
X *equal* Y	=	=	XXX YYY
X *meets* Y	m	mi	XXXYYY
X *overlaps* Y	o	oi	XXX YYY
X *during* Y	d	di	XXX YYYYYY
X starts Y	s	si	XXX YYYYY
X finishes Y	f	fi	XXX YYYYY

Figure 3 Temporal Intervals, adapated from Allen (1983)

[11] We use the prefix "gci:" to identify classes and properties from the GCI Foundation ontology

OWL-Time, based on the work of (Allen, 1983) has defined intervals with properties such as hasBeginning and hasEnd with range restricted to ot:Instant. It also provided interval relation properties such as intervalEquals, intervalBefore, intervalMeets, intervalOverlaps, intervalStarts, intervalDuring, intervalFinishes, and their inverse properties: intervalAfter, intervalMetBy, intervalOverlappedBy, intervalStartedBy, intervalContains, intervalFinishedBy, as shown in Figure 3. This provides us the ability to evaluate if all temporal components of the supporting data of the indicator are consistent with the indicator value. For example, a 15.2 indicator published in 2013 intended to measure the homeless population size ratio for the interval January 1, 2013 to December 31st, 2013. The interval is related to the indicator via the property gci:'for_time_interval'. We need to consider the following questions regarding the indicator's supporting data: when was the homeless population size collected? Was it collected before or during the interval? It is obvious that the supporting data would be inconsistent with the definition if it was collected before or after the interval specified by the indicator.

In addition to represent the meta information about an indicator, the GCI Foundation ontology is also capable of representing an indicator's definition and supporting data.

Measurement

As stated by Fox (2015c), "a city indicator is a measure of some property of a city. At the core of an indicator lies a number". GCI Foundation ontology represents an indicator as a 'Quantity' which represents "what is being measured for the city" (Fox, 2015c). A 'Quantity', e.g. 'length of ladder', is linked to a 'Measure'. A 'Measure' is measured with a number and a 'unit of measure', e.g. '3 metres'. The scale of 'unit of measure' can be a nominal, ordinal, interval or ratio scale. In order to represent these concepts, GCI Foundation ontology implements the Ontology of units of Measure and related concepts (OM) (Rijgersberg et al., 2013). The three core classes of OM include om:'Quantity'[12], which is a parent class of all city indicators. The om:'Quantity' class is linked to a om:'Unit_of_measure' and a om:Measure. A om:'Unit_of_measure' can be a om:'Singular_unit' (e.g. metre, kg) or a om:'Compound_unit' which has subclasses such as om:'unit_multiplication' and om:'unit_division' (e.g. m/s). Units of

[12] We use the prefix "om:" to identify classes and properties from the OM ontology

measure can also be multiple or submultiples of units. Multiple and submultiple units can be expressed with the use of prefixes such as ‘kilo’ or ‘milli’. E.g. kilometer (multiple), millimeter (submultiple). A set of base units are defined for each systems of units. For example, meter and seconds are base units for SI. GCI Foundation ontology defined a gci:’population ratio unit’ as an instance of om:’Unit_of_measure’ to represent the unit for a population ratio indicator. A gci:’population_ratio_unit’ has a om:numerator and om:denominator both linked to gci:’population_cardinality_unit’ (Figure 4). A om:’Measure’ has a om:’Unit_of_measure’ and is linked to a datatype via the property om:’numerical_value’ (Fox, 2015c). Figure 5 depicts the class structure combined from GCI Foundation and OM ontologies.

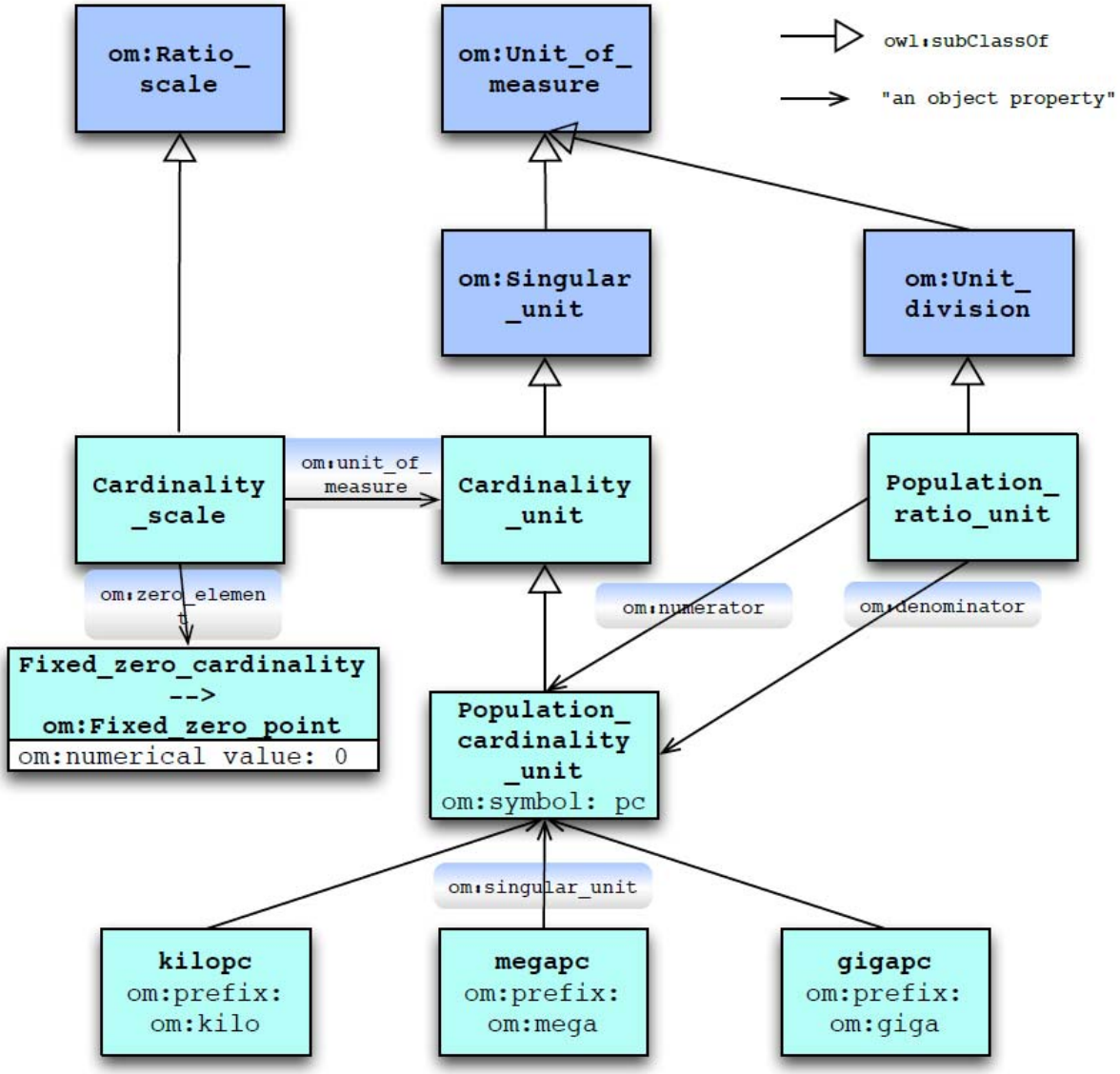


Figure 4 GCIO Unit of Measure, adapted from Fox (2015c)

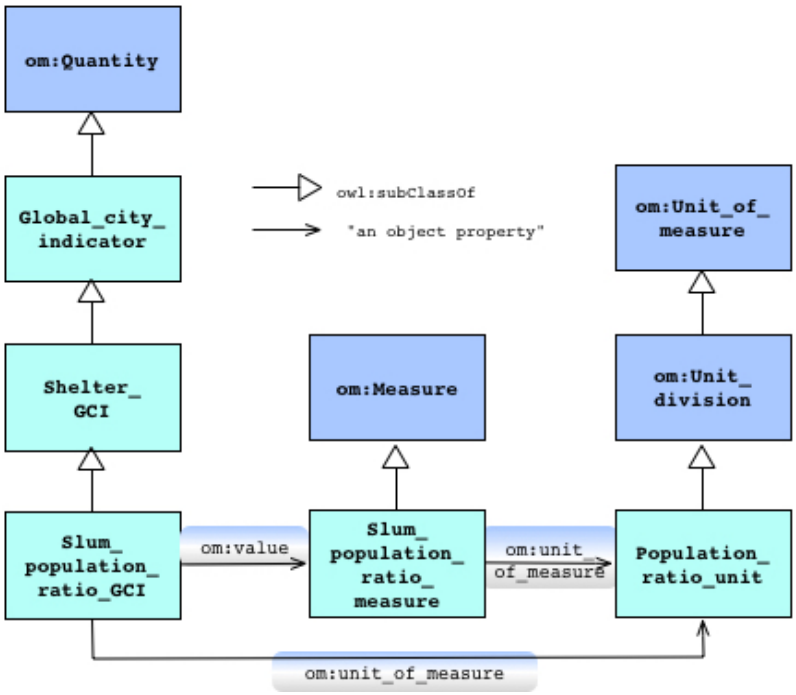


Figure 5 OM implemented by GCIO. Adapted from Wang & Fox (2015)

Statistics

A ratio indicator has om:'numerator' and om:'denominator' which are properties that link to a specific kind of om:'Quantity'. GCI Foundation ontology has defined a subclass of om:'Quantity' called gci:'Population_size' with its unit linked to gci:'population_cardinality_unit'. The gci:'Population_size' class has a property gci:'cardinality_of' that is linked to gci:'Population' class which was then linked a 'Person' class or its subclass depending on the theme specific knowledge of the indicator.

GCI Foundation ontology implemented the GovStat[13] general statistic ontology which provided the class 'gs:Population'. A 'gs:Population' is linked to a 'Parameter' (e.g., count, mean, standard deviation) by the gs:'is_described_by' property. In the case of 15.2 indicator it requires a count of the homeless and city population. In statistics it is almost always the case that only a

[13] The GovStat Ontology is not available online, but a version with the GCI extensions can be found at: http://ontology.eil.utoronto.ca/govstat.owl. We will use the prefix "gs:" to identify classes and properties from the ontology.

portion of the population is measured. This portion is represented by the class gs:'Sample', and the parameter being measured is represented as a subclass of gs:'Statistic' (Fox, 2015c). Figure 6 depicts the main classes and properties of the GovStat ontology.

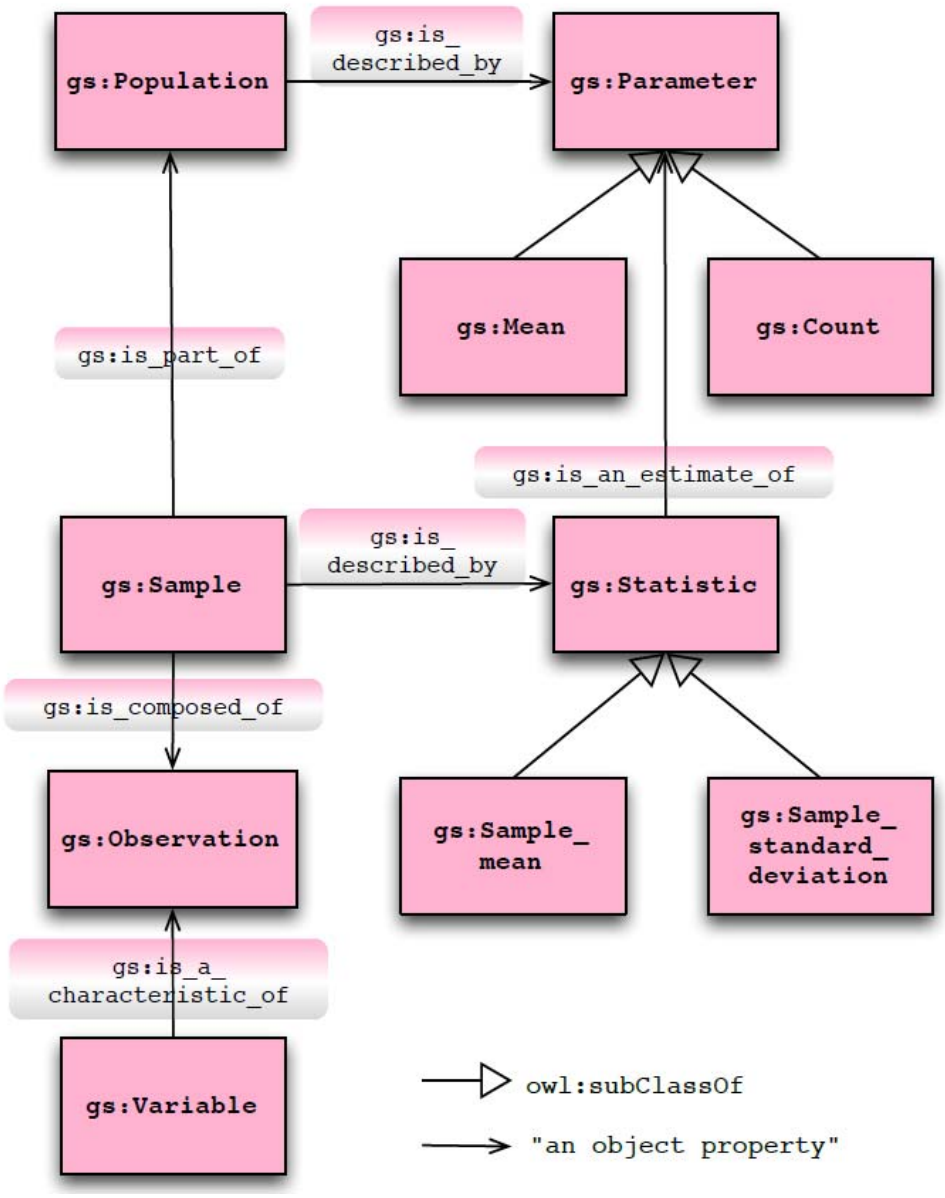


Figure 6 GovStat Ontology, adapted from Fox (2013)

In order to represent its collection, the 'Population' class must identify the area in which the population resides, i.e., the city, and what characterizes a member of the population (Fox, 2013). Therefore the GCIO extended gs:'Population' class with the following two properties:

- ‘located in’ whose range is geo-spatial feature (e.g. City) that indicates the location of where the Population was drawn from.
- ‘defined by’ whose range is a class that all members of the Population are subsumed by (e.g. Homeless population is defined by a homeless person) as discussed in the following section.

Figure 7 illustrates the class structure of a ratio indicator defined by GCI Foundation ontology.

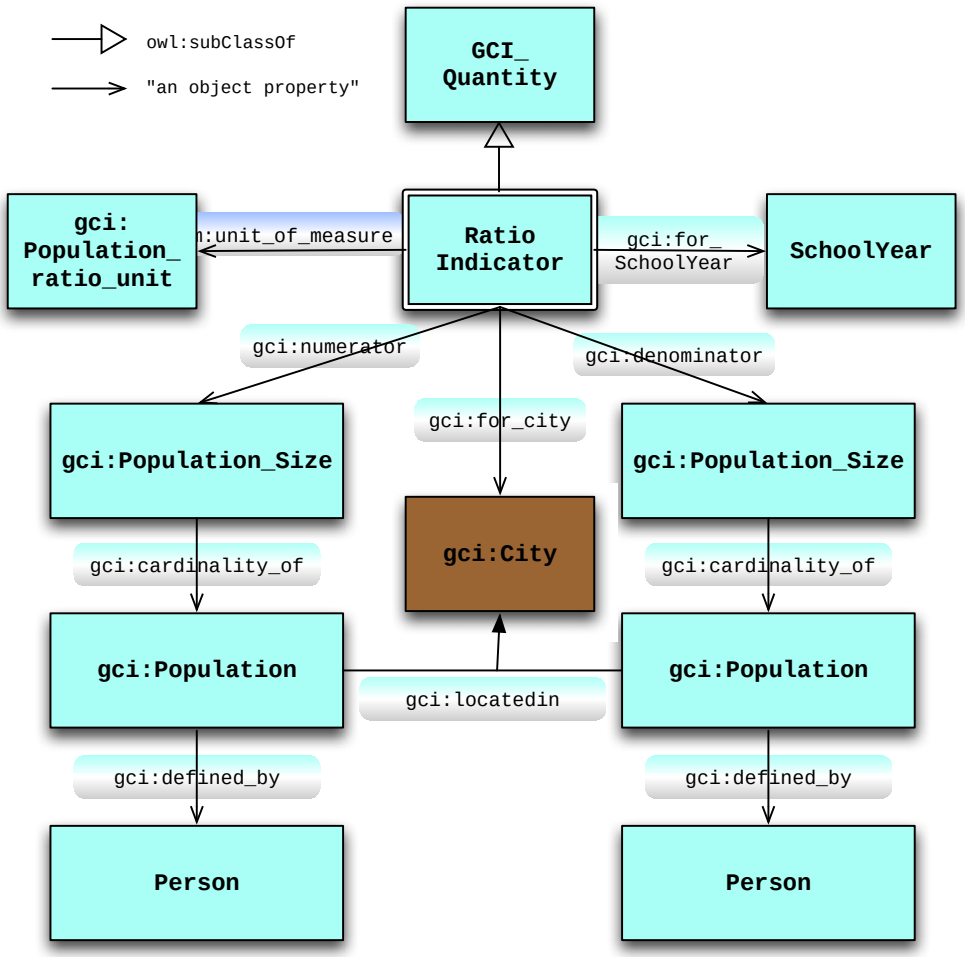


Figure 7 GCI Ratio Indicator, adapted from Fox (2015b)

GCI Shelters Ontology

GCI Shelter ontology (Wang and Fox, 2015) was created to represent shelter theme specific indicator knowledge. The GCI Shelter ontology includes concepts such as households, slum

household, homeless person, and household with unregistered titles, slum area, shelters, and living conditions. These concepts are necessary when representing indicator definition outlined by the ISO 37120. For example, based on the definition of ISO 37120 for homeless person stated previously, we can exapand the definition with more detail: “Cooper (1995) discusses the ideas of relative and absolute homelessness. Absolute homelessness occurs when there is neither access to shelter nor the elements of home. A person may be in relative homelessness; that is, they may have a shelter but not a home” (Habitat, 2000). In Wang & Fox (2015), absolute and relative homeless person was defined as follow: “both subclasses of Homeless_person possess an object property gcis:livesIn[14] but with different constraint values. Relative_homeless_person lives in homeless shelters (gcis:Homeless_shelter) while Absolute_homeless_person can live in any place throughout the city but not a homeless shelter or a home. We use the class ‘sumo:Place’ to represent places that an absolute homeless person can live in.“ A homeless shelter can be further classified into emergency shelter, single adult shelter, and family shelter, etc. (Wang & Fox, 2015). Figure 8 provides an overview of the class and properties from GCI Shelter ontology.

[14] We use the prefix “gcis:” to identify classes and properties from the GCI Shelter ontology

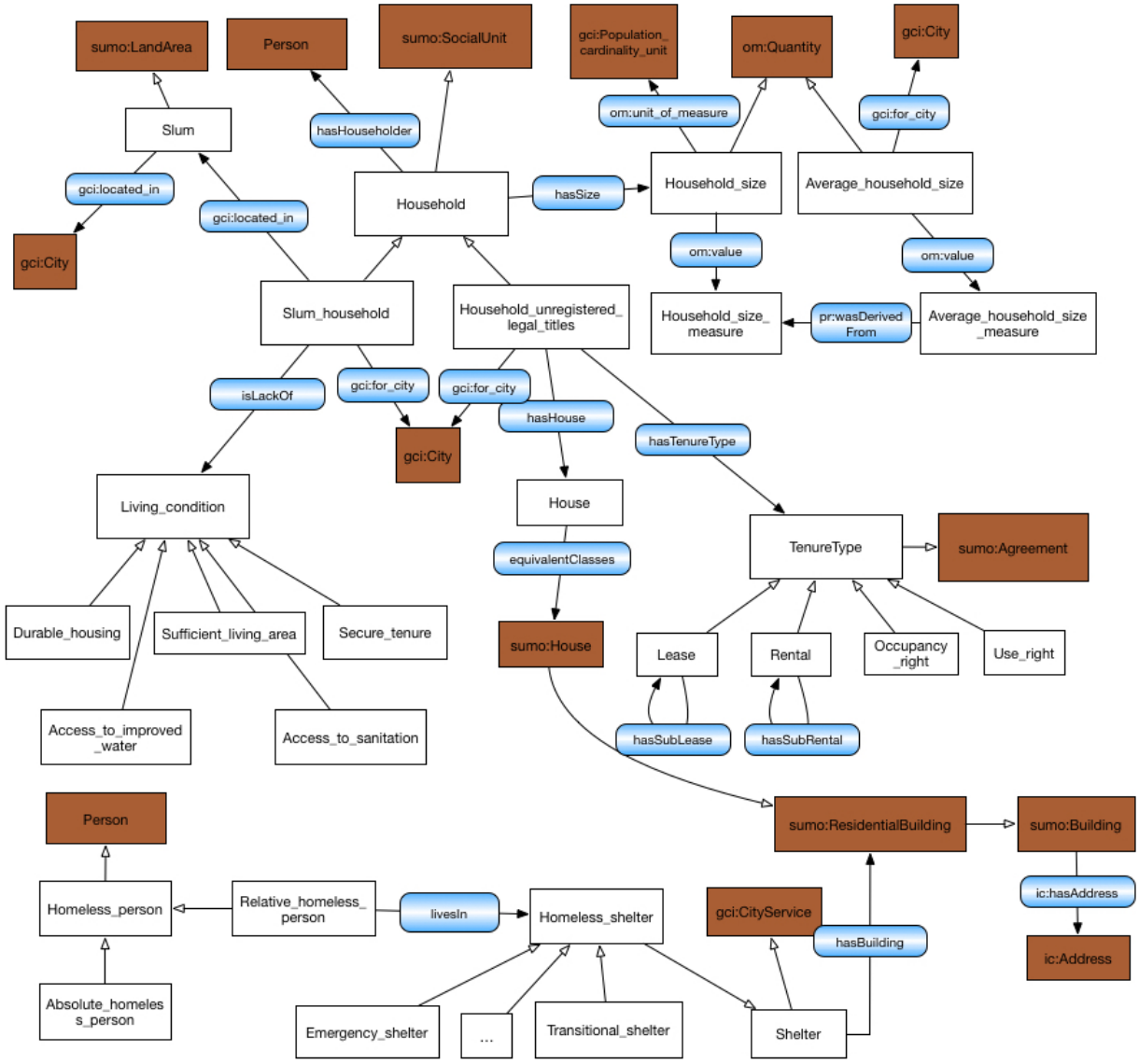


Figure 8 GCI Shelter Ontology, adapated from Wang & Fox (2015)

ISO37120 Shelter Indicator Definition

Wang & Fox (2015) define a Shelter ontology for representing the definitions and supporting data for the shelter theme indicators defined in ISO 37120. In the following we provide a detailed example of 15.2 shelter indicator definition.

Each indicator defined in the shelter theme of ISO 37120 standards was represented by class, e.g. iso37120:'15.2'[15] which is the subclass of om:'Quantity' class with a unit of measure that is a subclass of 'om:Unit_of_measure' and a value that is a subclass of 'om:Measure' where the actual numerical value of the indicator is linked to via a data property om:'numerical_value'. Meta information about this indicator is linked via object properties such as gci:'for_city', gci:'for_time_interval' which link the 15.2 indicator a 'City' and a 'Interval' that represent a year.

The supporting data of the ISO 37120 15.2 indicator, namely, homeless population size (isos:'15.2_Homeless_population_size')[16] and population size of the city (isos:'City_population_size'), can also be represented. Both are represented as subclasses of om:'Quantity' with a value and unit of measure. Each population size is a 'cardinality of' a population (gs:'Population') with a property gci:'located_in', that identifies the area that the Population is drawn from. This links a 'Population' to a 'City', and gci:'defined_by' that identifies the members of the Population that are to be counted. In our case, a 'homeless population' is 'defined by' a 'homeless person'. A set of consistency axioms are implemented in Prolog to ensure that indicator and supporting data are linked to the same unit of measure, placename, and temporal entities. Intra-indicator consistency axioms ensure all individuals within an indicator is consistent. E.g. populations of 15.2 indicator are referring to the same instance of city. Inter-indicator consistency axioms verify if individuals from two indicators are the same instance. E.g. Both 15.2 indicators published by Toronto and New York City are referring to the same year. Consistency axioms also describes arithmetical constraints that involves numerical comparison and arithmetical operations. For example, the value of a ratio indicator should be the quotient of its numerator divided by its denominator.

Figure 9 depicts the structure of 15.2 indicator. Other concepts related to shelter indicators defined by ISO 37120 such as slum household population size, household size, and population size of households with unregistered titles, are also represented by class and properties. Detailed

---

[15] 'iso37120' is the prefix used for 'http://ontology.eil.utoronto.ca/ISO37120.owl#'.

[16] 'isos' is the prefix used for 'http://ontology.eil.utoronto.ca/GCI/ISO37120/Shelters.owl#'.

information about shelter indicator definition represented in OWL can be found in (Wang & Fox, 2015).

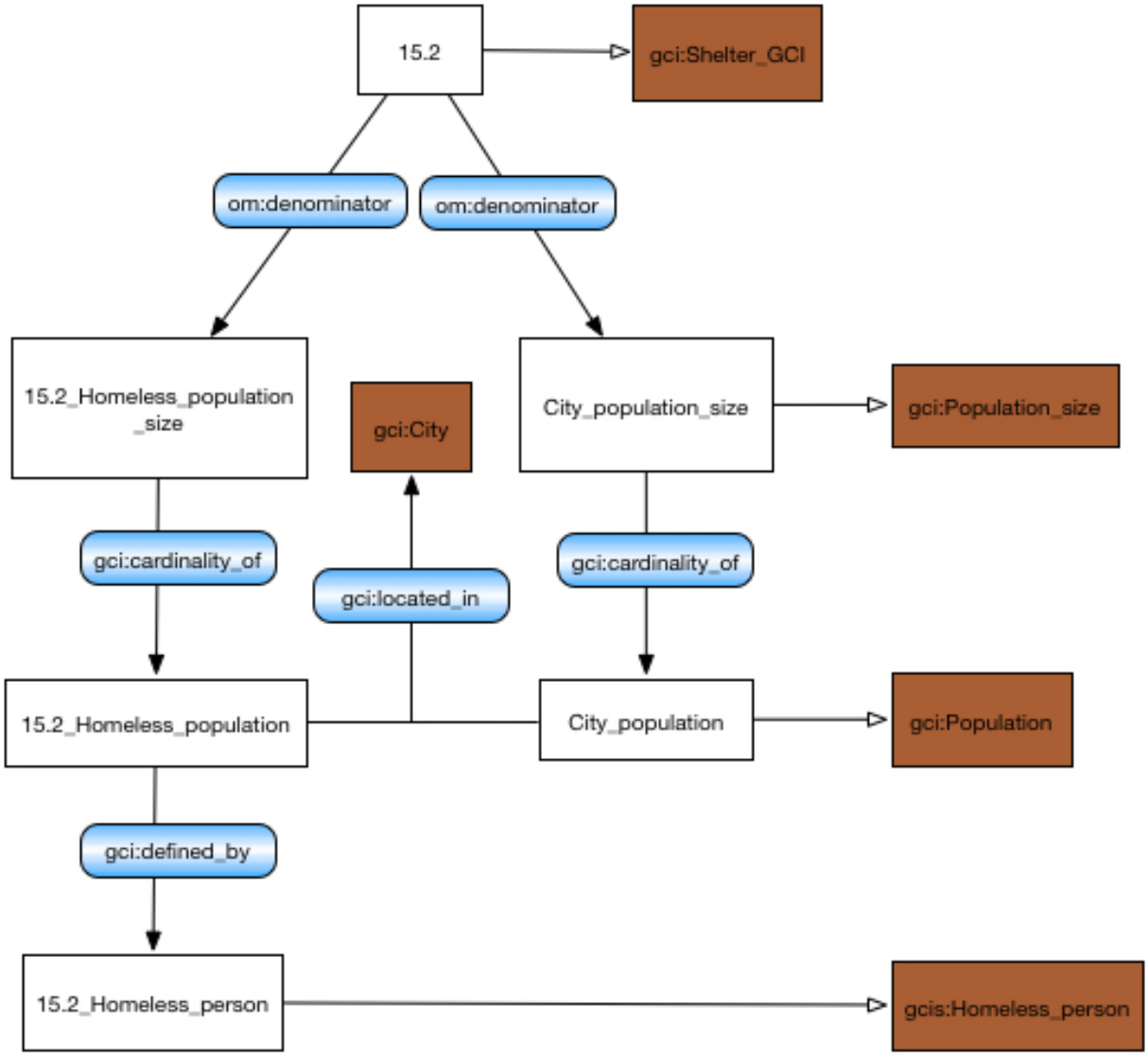


Figure 9 15.2 shelter indicator definition, adapted from Wang & Fox (2015)

The GCI ontologies discussed have represented the definition of ISO 37120 shelter indicators. Cities are able to publish a city indicator's value and its supporting data by creating instances of classes from the definition. The shelter theme specific knowledge can be represented with the GCI Shelter ontology. Cities can extend the GCI Shelter ontology to represent city-specific knowledge such as the types of homeless shelters used when defining a 'homeless person'. The GCI Foundation ontology was used as a basis, since it provides the ability to represent the meta

information of a published indicator. In next section we provide an overall view of the architecture of the ISO 37120 ontologies we described.

2.2.3 Architecture of the PolisGnosis GCI Ontologies

Figure 10 depicts "the organization of files used to define the ISO 37120 ontology the PolisGnosis project is developing. At the highest level, i.e., ISO 37120 Ontology level, the ISO 37120 module contains the globally unique identifier (IRI) for each ISO 37120 indicator" (Fox 2015a). For example, for "Number of homeless per 100 000 population" indicator, the IRI is: http://ontology.eil.utoronto.ca/ISO37120.owl#15.2. This is also written as 'iso37120:15.2' by using the prefix 'iso37120:' to represent the IRI for ISO 37120 Ontology.

The indicator definitions for each theme are represented by a different set of '.owl' files. For example, the ISO37120/shelter definition discussed above, is used to represent definitions of ISO 37120 shelter themed indicators. These indicator definitions will then implement corresponding GCI ontologies to represent theme specific knowledge. For example, the GCI Shelter ontology discussed previously was used to represent concepts related to shelter and housing social services. As discussed previously, all GCI theme specific ontologies use GCI Foundation to represent generic concepts such as population, quantity, measure and meta information (Fox, 2015c).

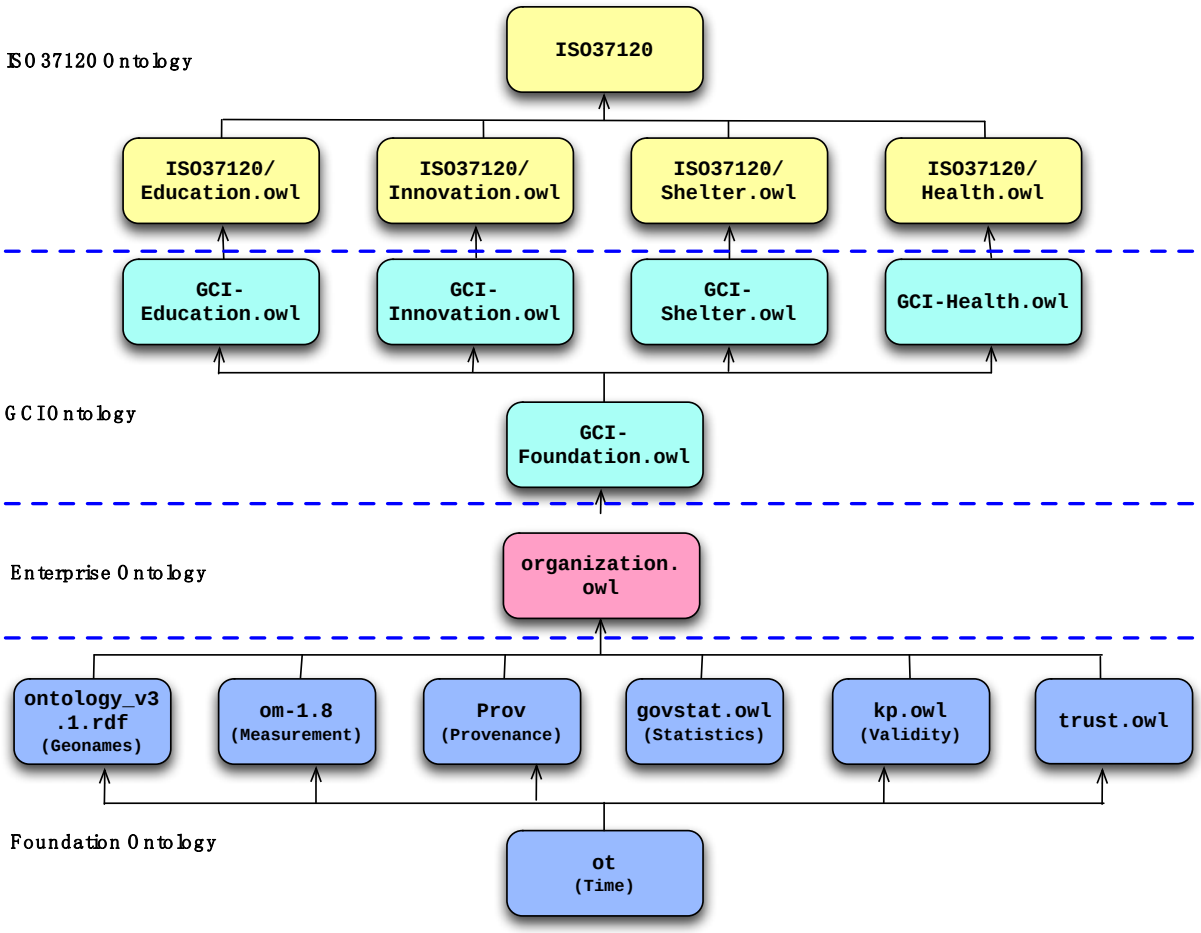


Figure 10 Structure of ISO 37120 Ontologies, adapted from Fox (2015b)

A complete list of theme specific indicator definitions and their corresponding GCI ontology with theme specific knowledge is shown in Table 1 below with their development status as of June 2016:

Indicator Definition	**Theme Specific Ontology**	**Development Status**
ISO37120/Education.owl (Fox, 2015b)	GCI Education (Fox, 2015b)	Complete
ISO37120/Shelter.owl (Wang & Fox, 2015)	GCI Shelter (Wang & Fox, 2015)	Complete

ISO37120/Innovation (Forde & Fox, 2015)	GCI Innovation (Forde & Fox, 2015)	Complete
ISO37120/Health (Falodi & Fox, 2015)	GCI Health (Falodi & Fox, 2015)	Complete
ISO37120/Finance (Wang Z. & Fox, 2016)	GCI Finance (Wang Z. & Fox, 2016)	Complete
ISO37120/Environment (Dahleh & Fox, 2016)	GCI Environment (Dahleh & Fox, 2016)	Complete
ISO37120/Transportation	GCI Transportation	In progress
ISO37120/Energy	GCI Energy	In progress
ISO37120/Safety	GCI Safety	In progress

Table 1 List of ISO 37120 indicator definitions and GCI theme ontologies

Prefixes used in the GCI ontologies are listed in Table 2 below:

Prefix	**Full Namespace**	**Ontology**
iso:	http://ontology.eil.utoronto.ca/ISO37120.owl#	IRIs for each ISO37120 indicator
gci:	http://ontology.eil.utoronto.ca/GCI/Foundation/GCI-Foundation.owl#	GCI Foundation ontology
isos:	http://ontology.eil.utoronto.ca/GCI/ISO37120/Shelters.owl#	The ISO37120 shelter indicators definitions
gcis:	http://ontology.eil.utoronto.ca/GCI/ Shelters/GCI-Shelters.owl#	GCI Shelter ontology

sumo:	http://www.ontologyportal.org/SUMO.owl#	Suggested Upper Merged Ontology (SUMO)
geo:	http://sws.geonames.org/	GeoNames
sc:	http://schema.org/	Schema.org
ic:	http://ontology.eil.utoronto.ca/icontact.owl#	International Contacts Ontology[17]
om:	http://www.wurvoc.org/vocabularies/om-1.8/	Ontology of units of Measure and related concepts (OM)

Table 2 Prefixes used in the GCI ontologies

2.2.4 IBM's Scribe Ontology

IBM's Smart Cities project has developed the Scribe ontology to represent city knowledge that includes classes such as city organization and services, flow of events and messages, key performance indicators, stakeholders, departments, and resources, etc. (Uceda-Sosa et al., 2011). Figure 11 illustrates Scribe's structure of representing message and event where a message is a subclass of event and also its subject. Scribe is also able to represent how city services area are linked to a city organization (e.g. Agency owns CityServiceArea). Cities can customize Scribe to represent different organizations, services in a city. As stated by Fox (2013), Scribe's "OWL definitions of the classes and properties are provided but axiomatization of the definition is limited and so its use of foundational ontologies" (Fox, 2013).

[17] 'ic:' is the prefix used for http://ontology.eil.utoronto.ca/icontact.owl

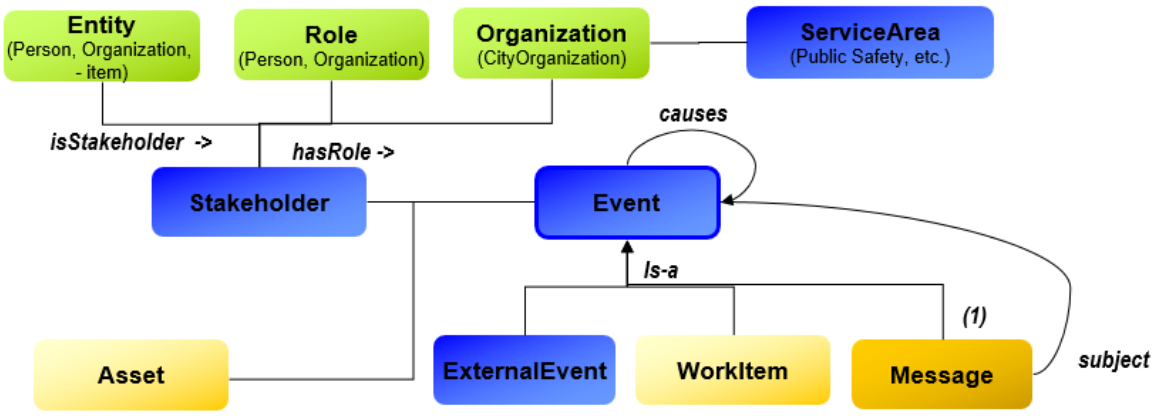


Figure 11 Scribe's model of a message, adapted from Uceda-Sosa et al. (2012)

Scribe ontology was intended to capture "dynamic aspects of city services" (Uceda-Sosa et al., 2011) and inter-departmental communication within a city. But limited information is provided for measuring performances of cities at city level. Indicator consistency analysis based on Scribe ontology becomes difficult since no indicator definitions nor relations between Scribe ontology and indicator definitions have been provided.

2.2.5 PAS 182 Smart City Concept Model

Publicly Available Standards (PAS) 182 is a document developed by BSI Group that defines a smart city concept model (SCCM) to increase interoperability of data across all sectors for a city. Sectors propose different terminology and model that creates difficulties when sharing data and knowledge across sectors. The SCCM provides a framework that creates a structure and classifies information and link datasets across sectors within a city (BSI, 2014). The SCCM includes Concepts such as Event, Object, Person, Organization, and Place.

SCCM contains a set of views which illustrates a scenario where data is shared in a city with a portion of classes and properties. For example, Figure 12 below shows a view of Item. As stated in BSI (2014), "an Item might be an Object such as a lamp post, a building, or a road, but an Item might also be an Organization, … a Person, … or a Community, …" (BSI, 2014). The class Item is also associated with a Place and a State such as temperature in a room.

Some metadata concepts are not covered by the SCCM. For example, Time, validity, and trust are not concepts included in the SCCM. Therefore it does not have the ability to describe the metadata and provenance information of datasets.

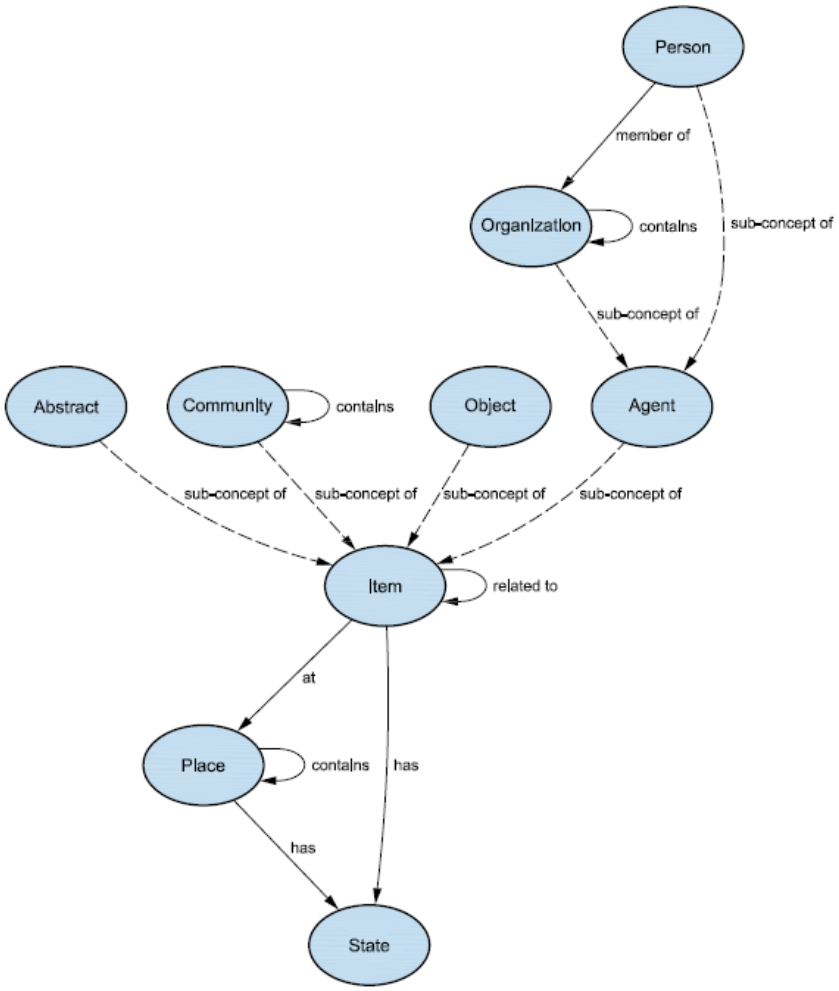


Figure 12 PAS 182 SCCM Item View, adapted from BSI (2014)

SCCM is ideal for integration of information across sectors. But concepts provided in SCCM are generally high level concepts such as Item and Objects. More specific concepts are needed when representing indicator definitions in order to perform indicator consistency analysis.

2.2.6 City Anatomy Ontology

As part of the City Anatomy model introduced in section 2.1.2, City Protocol developed a foundation ontology in OWL, City Anatomy Ontology (CAO), that represents “common

vocabulary and formal knowledge model linked to the City Anatomy model" (CPA, 2016). This foundation ontology will be extended for each domain according to domain specific knowledge. The ontology has been designed to represent the city from both a systems science perspective and perspective of dynamic processes of a city. A set of competency questions such as: which are the systems in a city, what is the structure of each system, and how does each system relate/interact with other systems, are proposed for the CAO to answer.

As described in section 2.1.2, City Protocol defined three systems of evaluating a city's performance as: structure, society, and interactions between society and structure. The city is a 'system of systems', it is "a set of relationships between multiple layers of a relatively large and permanent human settlement, with an administrative and legal status supported by local laws" (CPA 2016). Figure 13 below depicts core entities and their relationships of CAO.

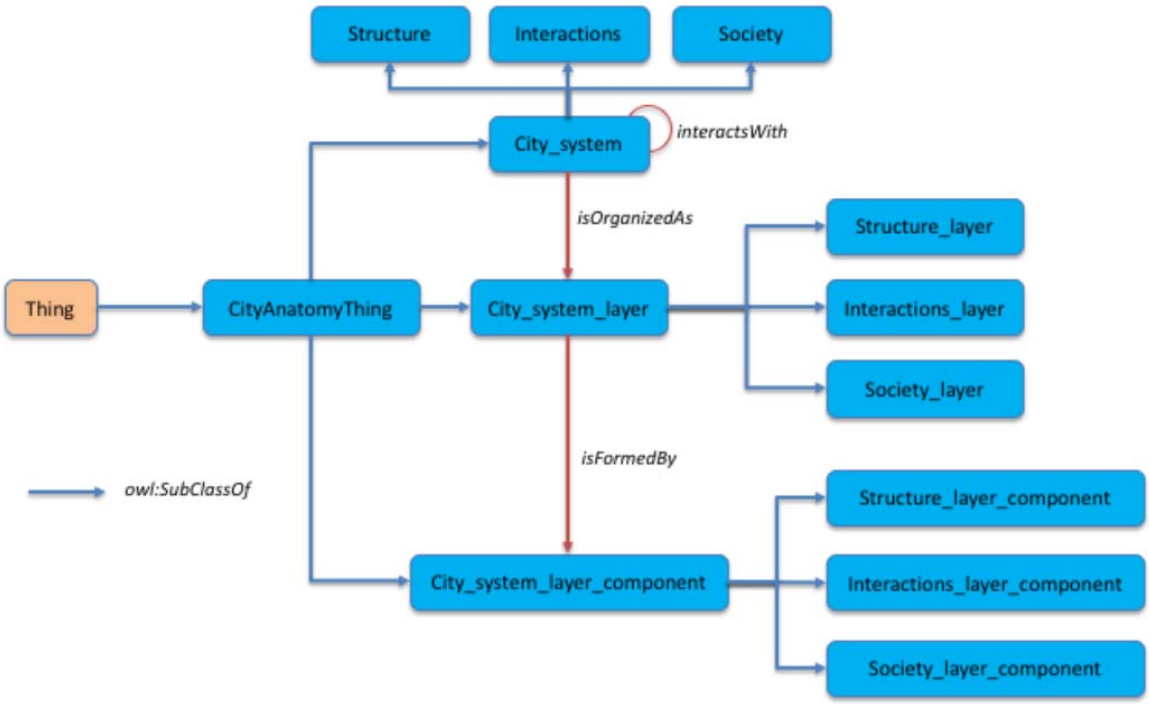


Figure 13 Core Concepts of CAO, adapted from CPA (2016)

Indicators are represented in CAO as a general concept with classes such as City_indicator, Structure_indicator, etc, and are linked to OM concepts such as Quantity, Measure, and Unit of

measure. There are currently no specific definitions defined in CAO for each indicator described in section 2.1.2. Figure 14 depicts indicator concepts from CAO.

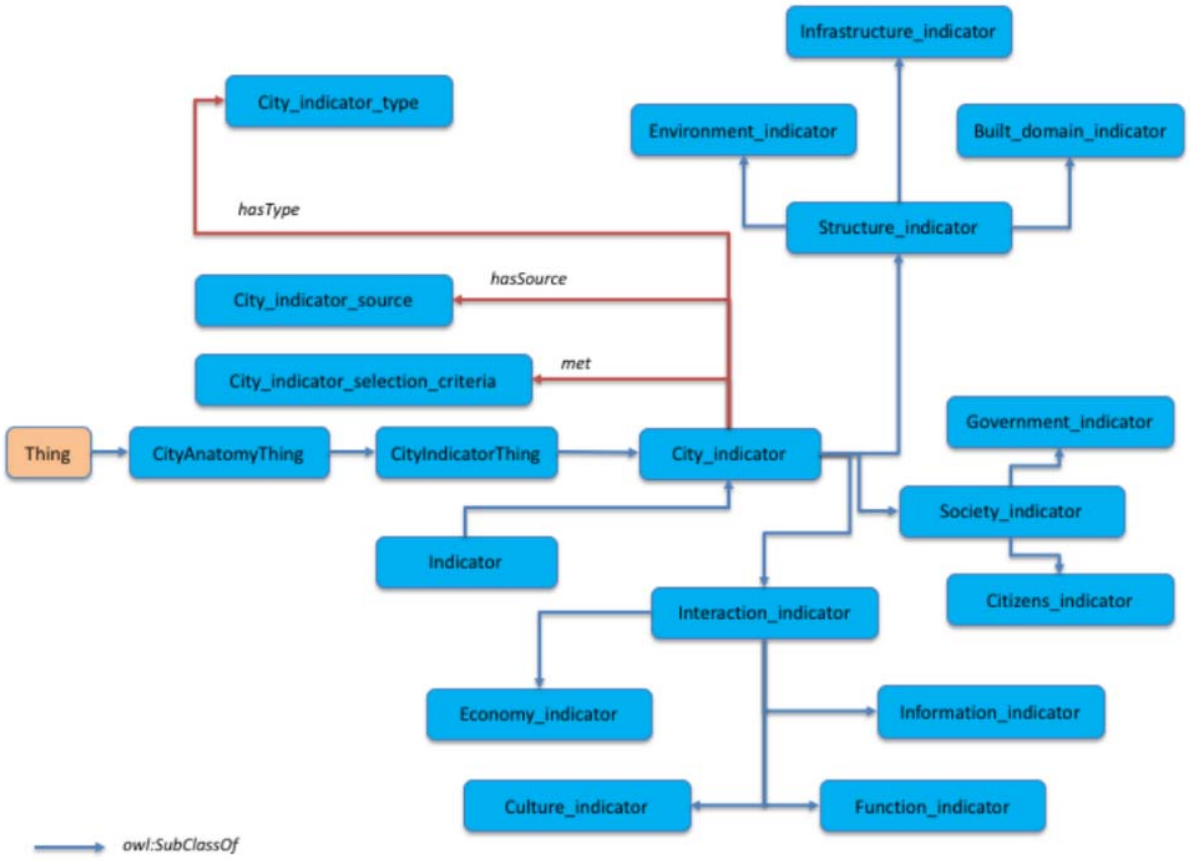


Figure 14 CAO Indicator Concepts, adapted from CPA (2016)

2.3 Consistency

It is important to evaluate the consistency of published indicator data to ensure it follows the definition of the indicator. Otherwise the indicator data has no value when comparing performance between cities (i.e., transversal) and to data published for the same city during another period of time (i.e., longitudinal). When comparing indicators their value and supporting data must be consistency with each other. For example, the indicators are measuring the same set of population and are using the same units. In this section we review a number of consistency evaluation tools.

2.3.1 Database Consistency

Currently, the majority of city indicator data is published in the form of spreadsheets and stored in statistical database management system (Fox, 2015a). In the context of database, a consistency check is usually done by 'DBCC CHECKDB' command in SQL Server which verifies the logical and physical integrity of the objects in a database (Mariuta, 2014). Etzion & Dahav (1998) described a model that uses high-level abstractions called stabilizer types denoting behavior patterns occurred during database update exceptions for consistency restorations. This approach provides high-level language to the exception handling portion of the application and substantially reduces the required programming (Etzion & Dahav, 1998). Cong, et al. (2007) employed a class of conditional functional dependencies (CFDs) proposed in Bohannon, et al. (2007) to capture inconsistencies and errors of the data. Algorithms were also provided to improve consistency of the data. One algorithm introduced was able to automatically compute a repair of the inconsistent database that satisfies the given set of CFDs. Another algorithm incrementally finds a repair in response to updates to a clean database.

A good practice for a database administrator is to run a consistency check on a regular basis. This is to ensure that both data and structure in the database is free of contradiction. Most cities publish their indicator data as spreadsheet on the web (Fox, 2015c). As stated previously, the goal is to represent indicator data in a Semantic Web compatible format eventually by using ontologies. Cities create instances when an indicator and its supporting data is published (Fox, 2015c) and these instances play role of data as in the relational database world (Martinez-Cruz, Blanco, & Vila, 2011). As with data in a relational database, consistency checking process is also an important aspect for any information represented with ontologies. Next we review a number of ontology consistency evaluation systems.

2.3.2 Ontology Consistency

Ontology consistency checkers verify if an ontology is logically consistent by verifying if there is at least one model that satisfies all axioms in the ontology. Within an ontology, its class definitions (TBox) and individual assertions (ABox) must be free of contradiction in order to be consistent. A class must have at least one instance in order to be consistent (concept satisfiability) (Grau, 2006). As stated by Horridge et.al (2009), "An inconsistent ontology, is an ontology that, by virtue of what has been stated in the ontology, cannot have any models, and

entails everything." (Horridge, Parsia, & Sattler, 2009). For example, if class 'All_male_shelter' and 'All_female_shelter' are disjoint classes, then a class 'All_male_female_shelter' that is subclass of both 'All_male_shelter' and 'All_female_shelter' is unsatisfiable since there can be no instance of it that maintains the consistency of the ontology. An individual is consistent with respect to a class if it is an instance of that class in every model of the ontology. For example, the class 'Homeless_person' has a property 'livesIn' with all values restricted to instances of 'Homeless_shelter'.

$$\text{'Homeless\_person'} \sqsubseteq \forall \text{livesIn.'Homeless\_shelter'}$$

$$\text{livesIn(john,houseX)}$$

$$\text{House(houseX)}$$

$$\text{'Homeless\_person'(john)}$$

If we assert that 'john' is a homeless person and lives in 'houseX' which is an instance of the class House. This is a contradiction since 'john' cannot be an instance of the class 'Homeless_person'.

In the following sections we review some of the existing ontology consistency evaluation system and discuss whether these systems are suitable for consistency evaluation for Global City Indicators.

2.3.3 ConsVISor

ConsVISor is an web-based consistency checker that evaluates the general consistency of an RDF or DAML+OIL ontology with Prolog axioms. ConsVIsor is able to detect contradictions between classes where no instances can be created for that class. E.g. Baclawski, et al. (2001) presented an example where a class 'Amphibian' was unable to be instantiated due to the fact that it is a subclass of two disjoint classes 'LandVehicle' and 'WaterCraft'. Figure 15 shows the architecture for ConsVISor (Baclawski, et al., 2002).

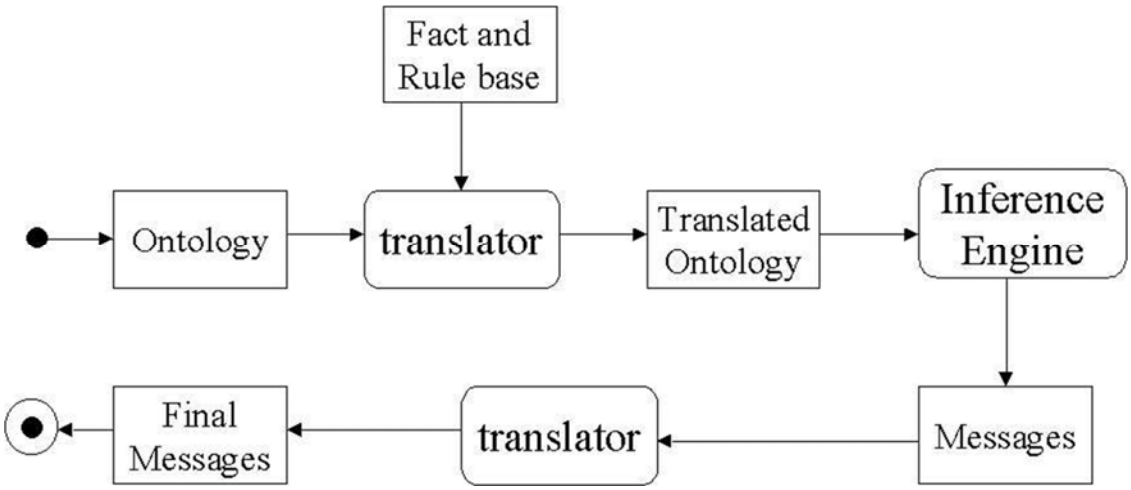


Figure 15 ConsVISor Architecture, adapted from Baclawski, et al. (2002)

ConsVISor can also detect inconsistency caused by cardinality restriction. Baclawski, et al. (2002) had demonstrated an example where it was entailed that the number of instances of a class is smaller than that of another class, but simultaneously greater than twice as much. Baclawski, et al. (2002) has shown that $\#Y \geq \#X \geq 2\#Y$ where "#X and #Y represents the number of instances of class X and class Y respectively. This implies that class X is either empty or has an infinite number of instances" (Baclawski, et al., 2002).

ConsVISor also checks potential errors such as spelling mistakes in class names. Since classes with spelling mistakes in class names can still be inferred as a class thus it is both syntactically correct and semantically consistent. ConsVISor will print warning messages in this case. ConsVISor "assumes that if two resources have different names and are not explicitly specified to be the same, then they are distinct resources." (Baclawski, et al., 2002)

ConsVISor is a consistency checker that aimed to ensure consistency within a single ontology. It provides error and warning messages when an ontology is self-contradicting and inconsistent with itself. But it does not have the ability to check consistencies cross two or more ontologies. This was discussed in section 'Conclusions and Future Work' of Baclawski, et al. (2002). The aim was to ensure consistency between smaller ontologies when they are used to merge into a

larger and complex ontology. ConsVISor[18] can be accessed through a demonstration version but as of November 2015 the webpage remains inaccessible after an ontology has been submitted.

2.3.4 SimpleConsist

SimpleConsist checks the consistency of instance data with respect to an ontology described in OWL. It is a plugin consistency checker within the Dacura data curation system (Mendel-Gleason, et al., 2015). It focuses on checking the consistency based on the structure of the ontology such as class hierarchy, properties associated to each class, domain and range restriction.

SimpleConsist checks for inconsistencies within an ontology by checking the set of constraints are satisfied at a triple-store state S. The state alters to S' when triples are updated or inserted into the triple-store. If constraints do not hold for S' then SimpleConsist can roll-back to the previous state S. Counter-example witnesses L are provided to the failure of the constraints. These witnesses are realised as resources not conforming to the constraints. Failure to provide a witness of the negation of the constraint is viewed as success. A portion of the failure witnessing predicates are listed in the Table 3 below.

¬ duplicateClasses(L)	No two classes may have the same name.
¬ orphanSubClasses(L)	No subclass can be a child of an unspecified class.
¬ invalidRange(L)	Ranges must refer to classes or types, and must be unique.
¬ duplicateProperties(L)	No two properties have the same name.
¬ orphanSubProperties(L)	No Subproperty is the child of an unspecified property.
¬ orphanProperties(L)	Instances must not use properties which are not defined.

Table 3 List of constraints, adapted from Mendel-Gleason, et al. (2015)

[18] ConsVISor: http://vistology.com/OLD/www/consvisor.shtml

SimpleConsist provides constraints to ensure that large-scale data are consistent in each state and ensure that the data remain consistent with every update.

2.3.5 Semantic Law Checker with Vehicle Ontology

Defined a framework that connects linguistic and conceptual problem in law-text to semantic deficiencies such as polysemy, endophora, exophora, under specification, inconsistency, and false agreement, etc. The semantic deficiencies including inconsistencies were identified through a Vehicle ontology written in OWL that complies with official documents such as the Brazilian Traffic (law) Code (CTB), National Traffic Code (Brazil) when formalizing definitions for concepts such as vehicles, automobile, electrical vehicles, propulsion type, etc. Logical inconsistencies for vehicle concepts were identified in law-text, for example, in CTB, an "Electrical vehicle" was defined both disjoint and subsumed by an "Automotor vehicle" (Freitas, Candeias Jr, & Stuckenschmidt, 2011).

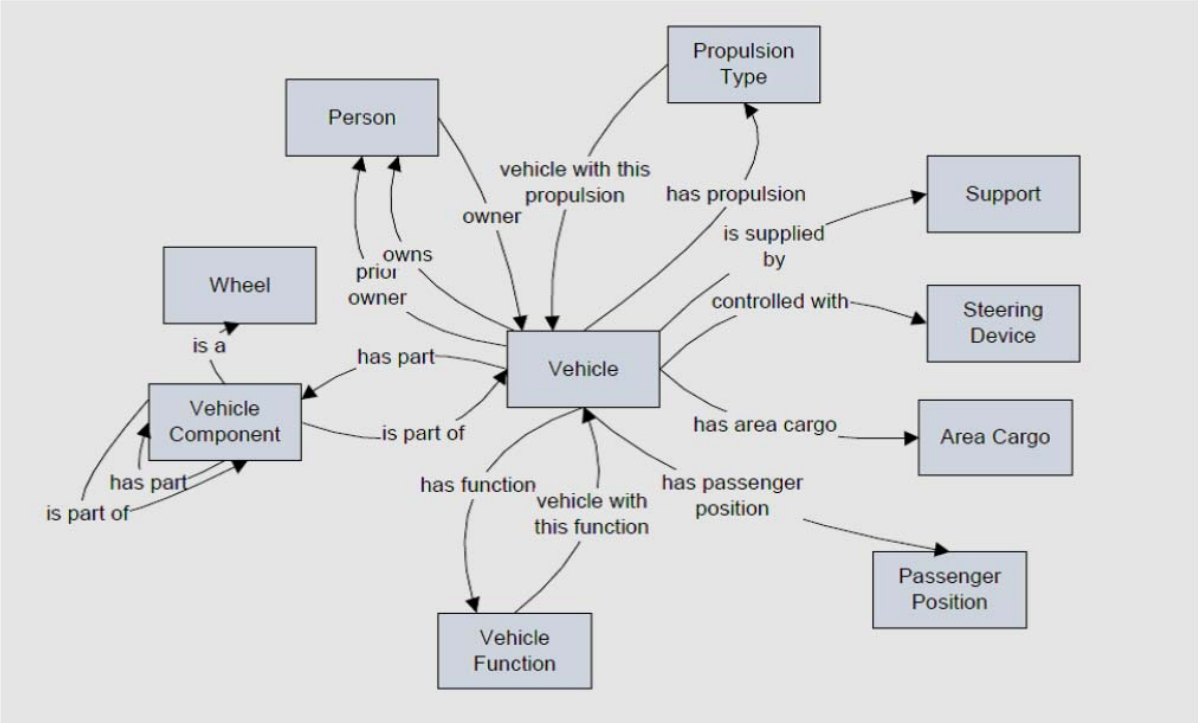


Figure 16 Classes and Properties of Vehicle Ontology, adapated from Freitas et al. (2011)

A semantic law checker shown below was envisaged with its goal set to providing services such as checking contradictions between new laws and old ones (Freitas, Candeias Jr, &

Stuckenschmidt, 2011). As the goal of this paper was to outline a framework of semantic deficiencies in law codes which from an ontological aspect, and many of the semantic deficiencies defined were modelling problems which can be captured with a well modeled ontology. Inconsistencies buried in the concepts of the ontology, for example, is the definition of an electrical vehicle in 2010 consistent with itself in 2030, cannot be identified.

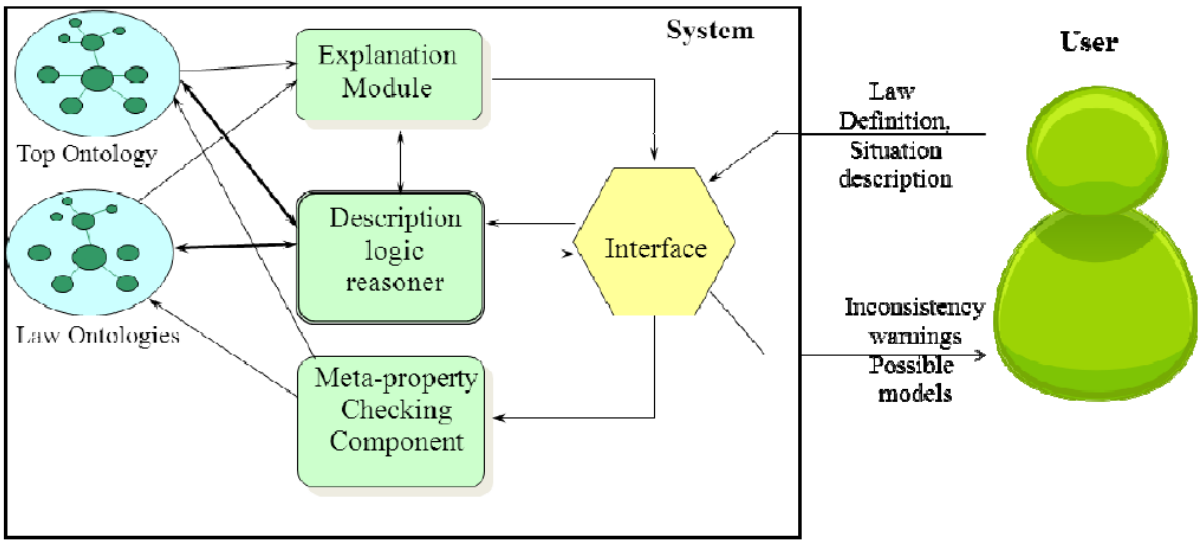


Figure 17 Architecture of a semantic law checker, adapted from Freitas et al. (2011)

2.3.6 Protégé OWL Reasoner

Protégé plug-in reasoners such as Pellet (Sirin, et al., 2007), FaCT++ (Tsarkov & Horrocks, 2006), and HermiT (Shearer & Motik & Horrocks, 2008) provide reasoning services for ontologies represented in OWL. Reasoning services including consistency test, satisfiability test, and classification are performed by constructing models of the ontology that satisfy all axioms in the ontology with completion rules. If there exists at least one model of the ontology that satisfies all the axiom the reasoner will terminate and conclude that the ontology is consistent. Otherwise, the reasoner will fail if a contradiction is detected indicating that the ontology is inconsistent. City indicators represented using ontologies need to be first evaluated to determine if the ontology and imported dependencies are logically consistent.

HermiT

HermiT is an OWL reasoner that checks if an ontology written in OWL is consistent.

HermiT "implements a 'hypertableau' calculus which reduces the number of possible models that need to be considered. HermiT also incorporates the 'anywhere blocking' strategy, which limits the sizes of models which are constructed" (Shearer & Motik & Horrocks, 2008).

Pellet

Pellet is an open-source Java OWL-DL reasoner with tableau based decision procedure that supports reasoning with individuals, user-defined datatypes, and debugging ontologies. It is compatible with Jena and OWL API libraries. Pellet provides the following reasoning services (Sirin, et al., 2007):

- "Consistency checking, which ensures that an ontology does not contain any contradictory facts.
- Concept satisfiability, which checks if it is possible for a class to have any instances.
- Classification, which computes the subclass relations between every named class to create the complete class hierarchy.
- Realization, which finds the most specific classes that an individual belongs to" (Sirin, et al., 2007)

FaCT++

FaCT++ is a reasoner based on FaCT (Fast Classification of Terminologies) which is a DL classifier with ability of satisfiability and subsumption test using tableaux algorithm (Horrocks, 1999).

OWL reasoners reviewed above are capable of evaluating the logical consistency and satisfiability of an ontology. City indicators are represented with ontologies such as GCI ontologies which were evaluated to be logically consistent (Fox, 2015b). However, information regarding types of inconsistency of city indicators need to be evaluated based on definitions of city indicators which is outside of reasoning services provided by the reasoners. As we will see in the end of the chapter, there are also cases of inconsistency that an OWL reasoner is not able to detect such as when performing transversal and longitudinal consistency analysis. Evaluation

of city indicator inconsistency with respect to indicator definition, city and theme specific knowledge is outside the scope of OWL reasoners that is mandatory for city indicator consistency analysis.

2.3.7 Information Consistency

Information consistency often refers to "text, images, and other content remaining the same regardless of how and where they are presented" (Costello et al., 2007). In addition to content of a document, inconsistencies may also occur from the following aspects of a document:

- Dependency: a document that either 1) is an exact copy of, 2) or has a quotation from a source is said to be a dependent of the source document. The content of the document may be inconsistent with the source when the source is modified and if there is no link provided between the document and the source.

- Revision: a revised document is derived from its original version. Inconsistencies may occur when changes are detected in the document's meta-information.

- Provenance: who was the creator of the document and what were the activities that generated the document. For example, two versions of documents are inconsistent if generated with different method.

- Time varying information: a piece of information may no longer be valid outside a certain time interval.

- Trust information: information that represents whether a document can be trusted. One may ask following questions regarding trust information of a document: Can the creator of the document be trusted? Does the document have dependencies and can they be trusted? Is the author an expert in the field? E.g. a piece of information cannot be trusted if the agent who generated the information is not trusted within the field where the information is published (Fox & Huang, 2003).

Traditional web contents are presented in non-machine-readable formats such as HTML, spreadsheets, or PDF which leave large amount of copies of documents on the web. When the source information is modified, revised or recreated, all copies of document containing the

information must be modified separately which leads to content inconsistency if not all documents are updated (Costello et al., 2007). Instead of find and reject inconsistent content completely, Hunter & Konieczny (2005) suggested approaches that will measure 'inconsistency' of a piece of information by measuring how much contradictions are involved (Hunter & Konieczny, 2005). The result can be used to aid applications involved with belief revision, where a new piece of information is perceived and a decision of whether it should replace the old belief needs to be made, belief merging, and negotiation. As stated in Hunter & Konieczny (2008), two approaches have been proposed previously. The first approach is to determine the minimum amount of statements mandatory to create inconsistency in a knowledge base. This amount is inversely proportional to the measure of inconsistency of the knowledgebase. Second approach is to measure the proportion of statement involved in the inconsistency within a knowledgebase. Inconsistency of knowledgebase increases when more statements are involved (Hunter & Konieczny, 2006). In Hunter & Konieczny (2006), a third approach that uses game theory technique (i.e., Shapley value) based on the previous approaches was proposed. This approaches measures inconsistency for each statement rather than the entire knowledgebase.

Information about evolution of the document is also difficult to track. For example, information about the user who modified the document and method used to generate new data is lost during process of updating the web document. Meta-information of the published data such as the identity of the publisher, date of publication, and other provenance information can also be tracked via the PROV ontology[19] (Belhajjame et al., 2012). Revisions and dependencies of the document are linked through property such as pr:wasDerivedFrom which allow users to track the source of documents.

Information about source of information, validity and trust can be represented with Knowledge Provenance (KP) proposed by Huang & Fox (2004). For any piece of information, a truth value of True, False or Unknown can be assigned. In addition to truth value, KP also is interested in the agent who asserted this piece of information, if the agent can be trusted, and does the information depend on another piece of information from a trustworthy source. Four levels of knowledge provenance models are defined in the order of most certain KP to most uncertain KP.

---

[19] 'pr:' is the prefix used for http://www.w3.org/ns/prov

The first and foundation level is static KP, which a truth value does not change over time. The model has been formally represented as an ontology[20] (Fox & Huang, 2003). Main classes in the static KP ontology are propositions (kp:KP_prop), documents (kp:Document), information sources (kp:InfoSource), trust relations (kp:TrustRelation), and signature status (kp:SigStatus) that can be either 'Verified', 'Failed', or 'NoSignature'. The classes and properties are shown in Figure 18. Other levels are: dynamic KP, where validity of information changes overtime; uncertain KP; and judgement based KP that is based on subjective judgement (Fox & Huang 2005). Each level of KP model is an extension that builds upon its previous levels.

[20] 'kp:' is the prefix used for http://ontology.eil.utoronto.ca/kp.owl

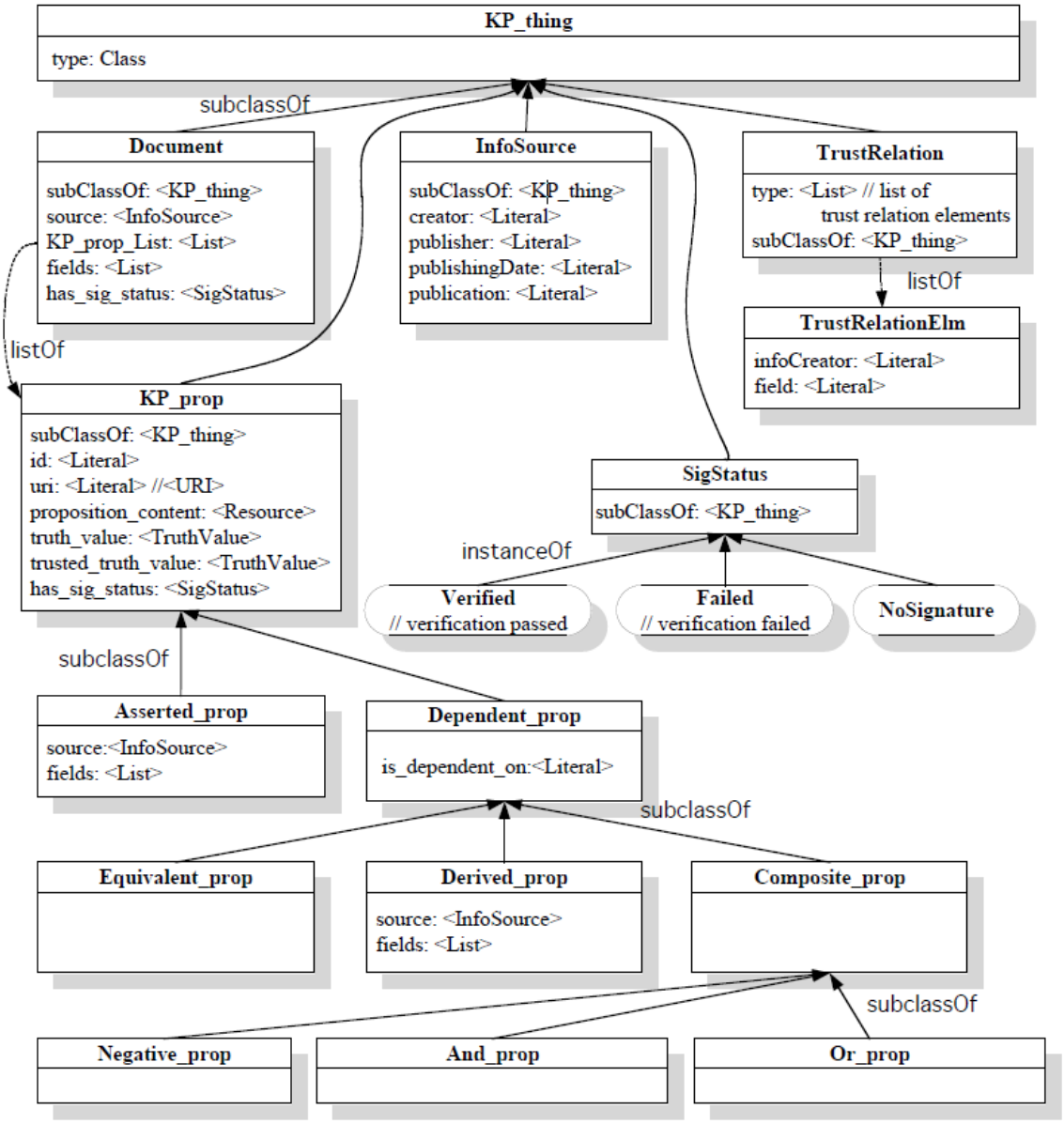


Figure 18 Classes and Attributes in KP, adpated from Fox & Huang (2003)

City indicators measure a city's properties with values. Value of a city indicator is represented with a measure (Fox, 2013) that carries information such as number, unit of measure, and meta-information of the measure. In addition to measurement consistency (e.g. numerical value and units are consistent), consistency of indicator values also need to be evaluated based on provenance, trust and validity information. As discussed in section 2.2.2, GCI Foundation

ontology implemented PROV and KP to represent provenance, trust and validity information of city indicator values. The following information are represented by GCI Foundation ontology:

- Agent who published the indicator value
- Time that the indicator value was published
- Activity that generated the indicator value
- Supporting data that the indicator value was derived from
- Time interval that indicator value effective
- Trust value of the publisher assigned by another agent

2.4 Summary

We have reviewed a number of city indicator standards and their representations. From these standards only ISO 37120 and City Anatomy indicators are formally represented in OWL with the GCI ontologies. Thus automated city indicator consistency analysis becomes possible for city indicators that are represented using the GCI ontologies. Consistency checkers and reasoners such as Hermit and Pellet are able to infer if an ontology is logically consistent and if an instance satisfies its class definition. But when evaluating city indicators consistency, there are two types of inconsistencies that are not detected by standard consistency checkers.

Inter-indicator inconsistencies may arise when performing longitudinal or transversal analysis. Consider a simple example when the same shelter indicator for two cities (e.g., Toronto and New York City) are to be compared. Separately, Toronto could publish its indicator data and city knowledge depicted in the following figure:

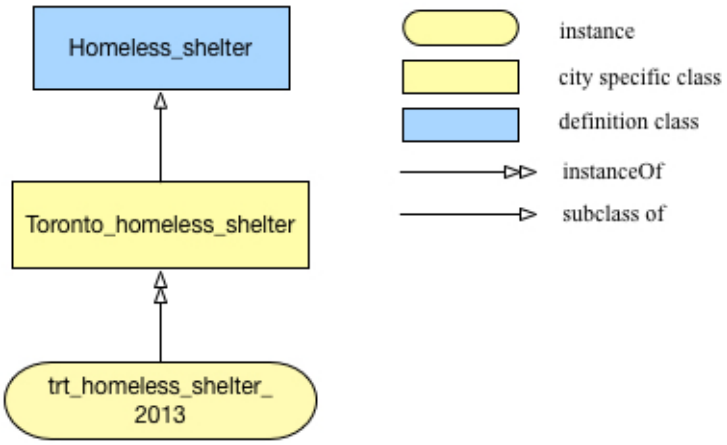


Figure 19 Published data and city knowledge Toronto

trt_homeless_shelter_2013 is an instance of Toronto's homeless shelter class (i.e., Toronto_homeless_shelter, which is a subclass of the Homeless_shelter class which represents the definition of a homeless shelter defined by ISO 37120. A standard consistency checker will determine whether Toronto_homeless_shelter class is subsumed by the Homeless_shelter class, and whether trt_homeless_shelter_2013 satisfies the Toronto_homeless_shelter class definition.

Now if we merge the indicator data and city knowledge of New York City we have:

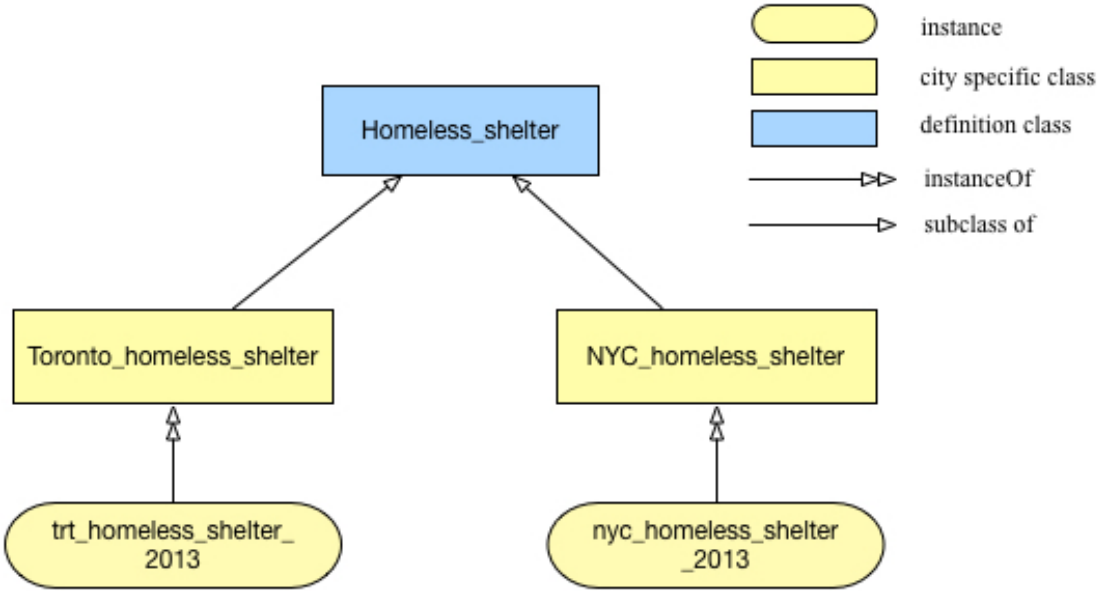


Figure 20 Merged indicator data and city knowledge

The standard consistency checker will verify each of the instances satisfy their corresponding definition of their city's shelter class, and that the city shelter classes are subsumed by the common Homeless_shelter class. But it will not compare Toronto_homeless_shelter class to NYC_homeless_shelter class. This is where transversal inconsistency may arise, namely each city is measuring different types of shelters.

The second type of inconsistency is intra-indicator inconsistency. If we have the derivation tree for a single indicator, there may inconsistencies within the tree. For example, the year the numerator was determined may be different than the year the denominator was determined.

The following identifies the general types of inconsistencies that may arise in both the inter and intra-indicator cases. They will be defined in more detail in subsequent chapters.

- Place inconsistency: geographical concepts of an indicator's definition are inconsistent. E.g. Toronto publishes an indicator where the homeless population was drawn from downtown area instead of the entire city as per ISO 37120 definition

- Temporal inconsistency: inconsistency is caused by temporal concepts such as temporal unit. E.g. City 1 publishes an indicator that measures city's performance monthly while city 2 measures the same performance annually.

- Population definition (Type) inconsistency: inconsistencies occur within the concepts that define the characteristics of the measured population. ISO 37120 defines homeless person as person who lives outdoor or in an emergency homeless shelter and a city's definition of homeless person involves person who lives in a treatment facility as well. Therefore, the city's definition of homeless person is inconsistent with ISO 37120 standards due to the additional 'treatment facility' concept included by the city's definition.

Finally, when standard consistency checkers detect an inconsistency, they do not provide indicator specific explanations as to the nature of the inconsistency. Is it unit of measure mismatch, or a temporal mismatch? Knowing the nature of the inconsistency is important.

Chapter 3
Definitional Consistency Analysis

3 Definitional Consistency Analysis

The ultimate goal of the PolisGnosis project is to determine the root cause of performance variations, whether they be longitudinal or transversal. But before we can perform root cause analysis, we need to determine whether the instance of an indicator reported by a city is consistent with the indicator's definition. If the indicator is inconsistent, root cause analysis is irrelevant because the data cannot be trusted nor compared to other cities. In this chapter we define a process for determining whether the city specific definition of an indicator and the derivation of its values published by a city is consistent with the indicator's definition.

A city's published indicator data can be inconsistent or potentially inconsistent with the indicator's definition. A city indicator's data is **definitional inconsistent** if it includes concepts/instances that are inconsistent with the indicator's definition, e.g. city's definition of homeless person. A city's indicator data may be **potentially inconsistent** with the indicator's definition if there is a possible interpretation of the indicator data that is inconsistent with the definition. For example, suppose that the homeless population of Toronto was counted across a subset of neighborhoods within Toronto. It is unknown whether these neighbourhoods contain all of the homeless. When a city publishes the data used to derive a specific indicator, it creates a set of instances that represent the indicator's value and supporting data. The published data will include instances of foundational and theme specific ontologies, and city specific ontologies such as Toronto's definition of homeless person (Fox, 2015c).

Published indicator data and indicator's definition are represented as graphs where nodes represent instances, classes or literals (e.g., integers, strings), and arcs represent properties. Let S_i be the graph that represents the indicator data and D_i be the graph that represents the indicator's definition for indicator i. Each node from S_i corresponds to a node from D_i in the sense that published city indicator data is an instance or subclass of the corresponding node in the indicator's definition. The correspondence between nodes of S_i and D_i is represented by $Cor(m_{ij}, n_{ij})$ where m_{ij} and n_{ij} are nodes of S_i and D_i respectively as shown in Figure 21. Table 4 and Table 5 list the notation used in definitional consistency analysis.

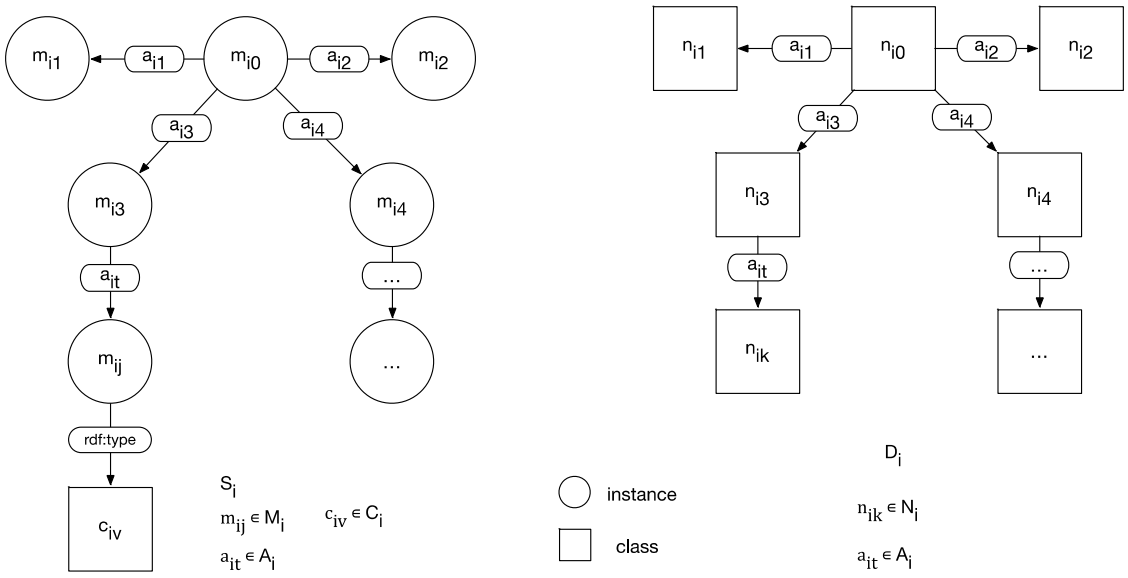


Figure 21 Published indicator data S_i and definition D_i

Indicator Definition and Theme Specific Knowledge

- Let I be the set of all indicators for a particular theme
- Let D be the set of all indicator definitions for a particular theme
- Let O^F be the Foundation ontology used in D
- Let O^T be the Theme Specific Ontology used in D
- Let $O = O^F$ union O^T
- D_i is the definition of indicator $i \in I$
- D_i is composed of a set of arcs A_i and nodes N_i
- $A_i \subseteq$ prop(O) where prop(O) is the set of properties defined in O
- $N_i \subseteq$ Class(O) ∪ Indiv(O) ∪ Literal(O) where Class(O) is the set of classes defined in O, and Indiv(O) is a set of individuals in O, and Literal(O) is a set of literals in O.
- $N^{class}_i \subseteq$ Class(O) $\subseteq N_i$

Table 4 Notation of Definition and Theme Specific Knowledge

Published City Indicator Data and City Specific Knowledge

- Let S be the set of all published indicator data for a particular theme
- Let O^C be the ontology used to define City Specific Knowledge used in S
- Let S_i be the graph that represents the data used to derive indicator i
- S_i is composed of a set of attributes A'_i, nodes C_i nodes M_i, and nodes N_i
- A'_i is a set of arcs (properties) in S_i
- M_i is a set of nodes (individuals) in S_i
- $C_i \subseteq Class(O^C) \cup Indiv(O^C) \cup Literal(O^C)$ where $Class(O^C)$ is the set of classes defined in O^C, and $Indiv(O^C)$ is a set of individuals in O^C, and $Literal(O^C)$ is a set of literals in O^C
- $C^{class}_i \subseteq Class(O^C) \subseteq C_i$

Table 5 Notation of Indicator Data and City Specific Knowledge

We represent that two nodes are connected by an arc (property) as $Tri(x, a_{it}, y)$, where both x and y can be nodes $m_{ij} \in M_i$, $c_{iv} \in C_i$, or $n_{ik} \in N_i$, while a_{it} is a property from A_i. The predicate $Tri(x, a_{it}, y)$ represents the triple with x as the subject, a_{it} as a binary property, and y as an object. x and y can be classes, instances or literals. We use the predicate Equal(x,y) to indicate that x and y are the same class or individual. We use the predicate Type(x,y) to specify that x is an instance of y (or subclass of y) where x is an individual and y is a class.

Before consistency can be analysed, we need to determine for each node in S_i, which node it corresponds to in definition D_i. The correspondence between nodes $m_{ij} \in M_i$ from graph S_i and $n_{ik} \in N_i$ from graph D_i are determined as follows:

For any $m_{ij} \in M_i$ in S_i and $n_{ik} \in N_i$ in D_i, $Cor(m_{ij}, n_{ik})$ is true if:

- There exist a direct rdf:type (instanceOf) relationship between the instance m_{ij} and definition class n_{ik}, or
- There exist a direct rdf:type relationship between the instance m_{ij} and a class n_{iv} where n_{iv} is a subclass of n_{ik}, i.e., $n_{iv} \sqsubseteq n_{ik}$, or
- There exist a node m_{ix} that corresponds to n_{iy}, i.e., $Cor(m_{ix}, n_{iy})$, which are linked to m_{ij} and n_{ik} via the property a_{it} respectively.
    - E.g. an indicator instance 15.2_trt_2013 published by Toronto corresponds the its definition class iso37120:15.2. The instance 15.2_trt_2013 links to the instance geo:Toronto via the property gci:for_city and the definition class iso37120:15.2 is

linked to gci:City class via the same property, then geo:Toronto corresponds to the class gci:City.

Predicate Cor(m_{ij},n_{ik})

$\forall m_{ij}\ n_{ik}\ n_{iv}$

Type(m_{ij}, n_{ik}) ∨

(Type(m_{ij}, n_{iv}) ∧ Subclass(n_{iv}, n_{ik})) ∨

$\exists m_{ix}\ n_{iy}\ a_{it}$ (Cor(m_{ix},n_{iy}) ∧ Tri(m_{ix},a_{it},m_{ij}) ∧Tri(n_{iy},a_{it},n_{ik}))

⊃ Cor(m_{ij},n_{ik})

Defintion 1 Correspondence

The starting point of correspondence should occur on the indicator value m_{i0} published by a city and the indicator class n_{i0} from its definition. For example, the instance '15.2_trt_2013' published by Toronto corresponds with the definition class iso37120:15.2 such that Cor('15.2_trt_2013', iso37120:'15.2') is true. As shown in Chapter 2, an om:Quantity class (e.g. gci:GCI_Quanity, gci:Population_size, etc.) is linked to om:Measure and om:Unit_of_measure in ISO 37120 indicator definition. Classes such as om:Measure are reused throughout the indicator's definition therefore correspondence is not unique between nodes in S_i and such classes in D_i. For example, let m_{iy} be the homeless population size value and m_{ix} be the value of indicator '15.2_trt_2013' where both are instances of om:Measure (n_{ik}). Thus both Cor(m_{ix}, n_{ik}) and Cor(m_{iy}, n_{ik}) are true.

All nodes $n_{ik} \in N_i$ that exist in the indicator's definition D_i should have a corresponding node $m_{ij} \in M_i$ from S_i such that Cor(m_{ij}, n_{ik}) is true. In the case where there is no correspondence detected for a definition class n_{ik}, there are missing data from the set of indicator data published by the city according to the indicator's definition. We specify the following inconsistency type to indicate that published indicator data S_i is inconsistent in terms of correspondence if for any corresponding nodes $m_{ij} \in M_i$ and $n_{ik} \in N_i$, there exists a class n_{iy} that is linked to n_{ik} via property a_{it} where there is no node m_{ix} linked to m_{ij} that corresponds to n_{iy}.

CI. Correspondence Inconsistency

$\forall m_{ij}\ n_{ik}$

$\mathrm{Cor}(m_{ij},n_{ik}) \wedge \exists n_{iy}\ a_{it}\left[\mathrm{Tri}(n_{ik}, a_{it},n_{iy}) \supset \neg \exists m_{ix}\ \left(\mathrm{Tri}(m_{ij}, a_{it},m_{ix}) \wedge \mathrm{Cor}(m_{ix},n_{iy})\right)\right]$

$\supset \mathrm{CI}(m_{ij},n_{ik})$

Defintion 2 CI. Correpondence Inconsistency

In the remainder of this section, we define the types of inconsistencies that may arise in indicator data.

3.1 Type Inconsistency

3.1.1 TC1. Class Type inconsistency

Two corresponding classes are type inconsistent if it is not the case that the two classes are equal, nor there exists a property owl:equivalentClass or owl:subclassOf between the classes, or one class is not subsumed by another class. We define a predicate Subclass(x, y) to indicate that class x is subsumed by class y.

Therefore, a class x is type inconsistent with class y if x satisfies the following:

- x is not equal to, an equivalent class, nor a subclass of y, or
- x is not subsumed by y.

We use the function card(x, a_{it}) to specify the cardinality restriction of property a_{it} on x where x is a class or individual. We define the following definition TC1 to state that class x is type inconsistent with its corresponding class y.

TC1. Class Type Inconsistency

$$\forall x\, y$$
$$\mathrm{Cor}(x, y) \wedge \neg\big(\mathrm{Equal}(x, y) \vee \mathrm{Tri}(x, \mathrm{owl{:}\ equivalentClass}, y) \vee \mathrm{Tri}(x, \mathrm{owl{:}\ subclassOf}, y)\big) \vee \neg\mathrm{Subclass}(x, y)$$
$$\supset\ \mathrm{TC1}(x, y)$$

Defintion 3 TC1. Class Type Inconsistency

For example, let x be the class that represents Toronto's definition of homeless person and y be the definition of homeless person defined by ISO 37120. ISO 37120 defines a homeless person as a person who lives outdoors or in a homeless shelter. But cities may have different definitions for specific types of homeless shelter. In this case, a Toronto homeless person is defined to live in either an emergency homeless shelter, a Violence Against Women shelter, or a treatment facility (City of Toronto, 2013) which we represent as x_1, x_2, and x_3 respectively. Assume x1 and x2 are subclass of y. Since x and y are distinct classes and there are no owl:equivalentClass nor owl:subclassOf relation asserted in between, we need to verify that x is subsumed by y. Let x' be x_1 or x_2 or x_3, the following predicates holds true

$$\mathrm{Tri}(x, \mathrm{gcis{:}\ livesIn}, x')$$

$$X' = x1 \vee x2 \vee x3$$

$$\mathrm{Tri}(y, \mathrm{gci{:}\ livesIn}, y')$$

The cardinality restrictions are 'exactly 1' for values of gci:livesIn for both x and y thus satisfies the condition

$$card(x, gci{:}\ livesIn) = card(y, gci{:}\ livesIn)$$

Then we must verify if x' and y' are equal classes or x' is a subclass of y'. x1 and x2 are subclasses of y' thus are type consistent with y'. x3 is neither an equivalent nor a subclass of y',

thus it is type inconsistent with y'. Since $\neg\text{Subclass}(x', y')$ thus $\neg\text{Subclass}(x, y)$ is true therefore $\text{TC1}(x, y)$ is also true. Thus we can conclude that class x is type inconsistent with class y.

3.1.2 TC2. Instance Type Inconsistency

The most basic definitional consistency of a published city indicator is to verify whether the indicator and its supporting data are instances of classes that define ISO 37120 indicator. That is, instance type inconsistency verifies that if the instances that make up a city's indicator are an instance of the same class, an equivalent class, a subclass of concepts defined in ISO37120, or have all necessary properties with values that satisfy the restrictions of those properties defined in the ISO 37120 definition (Fox, 2015b).

City published indicator data are represented by set of instances M_i of classes N_i and C_i in graph S_i that corresponds to classes N_i from indicator's definition D_i. Let m_{ij}, c_{iv}, and n_{ik} be nodes of M_i, C_i, and N_i respectively, where m_{ij} is an instance, c_{iv} and n_{ik} are classes. m_{ij} is instance type inconsistent if:

- There does not exist a direct rdf:type relation between m_{ij} and n_{ik}
- m_{ij} is not an instance of n_{ik}, and
- m_{ij} is an instance of c_{iv}, and c_{iv} is type inconsistent with n_{ik}

TC2. Instance Type Inconsistency

$$\forall m_{ij}\ n_{ik}$$
$$\text{Cor}(m_{ij}, n_{ik}) \wedge \neg\text{Type}(m_{ij}, n_{ik}) \wedge \exists c_{iv}\ \left(\text{Type}(m_{ij}, c_{iv}) \wedge \text{TC1}(c_{iv}, n_{ik})\right)$$
$$\supset\ \text{TC2}(m_{ij}, n_{ik})$$

Defintion 4 TC2. Instance Type Inconsistency

For example, let m_{ij} be the 15.2 indicator value published by Toronto, n_{ik} be the class iso37120:'15.2' where Cor(m_{ij},n_{ik}). Assuming there is a direct rdf:type such that Tri(m_{ij}, rdf:type, n_{ik}), or Tri(m_{ij}, rdf:type, c_{iv}) and c_{iv} is the same class, equivalent class or a subclass of n_{ik}, i.e.,

iso37120:15.2, then m_{ij} is instance type consistent n_{ik}. In case shown in Figure 22, given that $Cor(m_{ij},n_{ik})$ and m_{ij} is an instance of c_{iv}. The class c_{iv} and n_{ik} are linked to c'_{iv} and n'_{ik} respectively via property a_{it}. The instance m_{ij} is type inconsistent with n_{ik} if the class c_{iv} is inconsistent with n_{ik}, i.e., $TC1(c_{iv},n_{ik})$ is true.

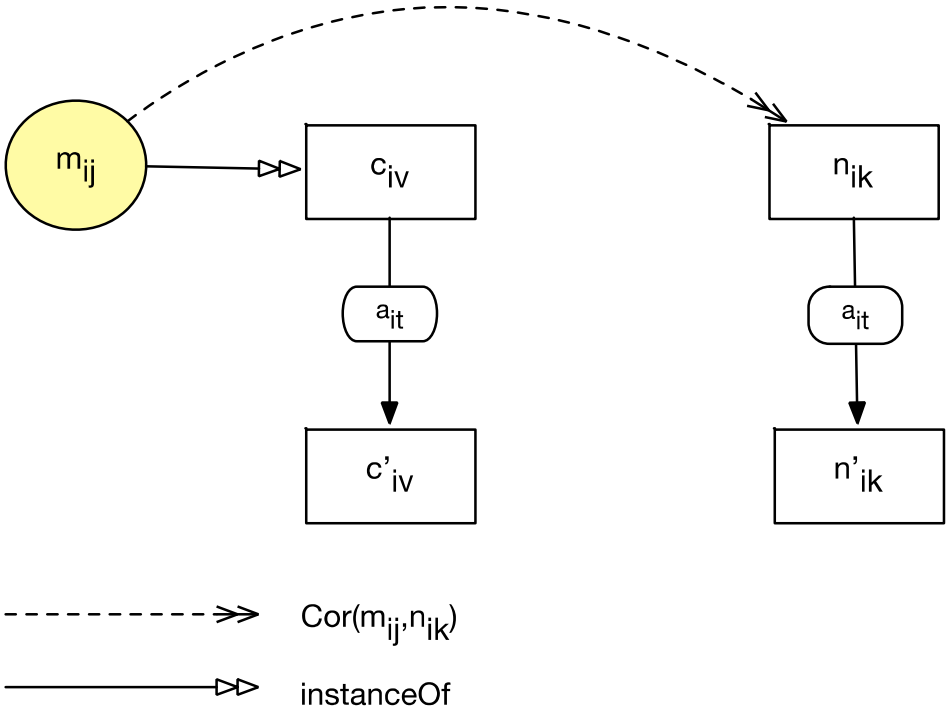


Figure 22 Evaluation of TC2(m_{ij},n_{ik}) consistency given that Cor(m_{ij}, n_{ik})

3.1.3 TC3. Property Inconsistency

An instance $m_{ij} \in M_i$ is potentially inconsistent with its corresponding definition class $n_{ik} \in N_i$ if there exist a necessary property a_{it} defined in n_{ik} that satisfies one of the following conditions:

- a_{it} does not exist in m_{ij}, or
- the cardinality of ait for m_{ij} does not satisfy the cardinality restriction defined in n_{ik}, or
- m_{ij} does not satisfy the value restriction of a_{it} defined in n_{ik}

We introduce $Nec(a_{it}, n_{ik})$ to indicate that a_{it} is a necessary property defined in class n_{ik}. The predicate $CardSat(m_{ij}, n_{ik}, a_{it})$ indicates that the instance m_{ij} satisfies the cardinality restriction of the property a_{it} defined by class n_{ik}.

TC3. Property Inconsistency

$$\forall m_{ij}\ n_{ik}\ n_{iy}$$
$$\mathrm{Cor}(m_{ij}, n_{ik}) \wedge \exists a_{it}\ \Big[\big(\mathrm{Nec}(a_{it}, n_{ik}) \wedge \mathrm{Tri}(n_{ik}, a_{it}, n_{iy})\big)$$
$$\supset \neg\exists m_{ix}\ \big(\mathrm{Tri}(m_{ij}, a_{it}, m_{ix}) \wedge \mathrm{CardSat}(m_{ij}, n_{ik}, \mathrm{a}_{it})$$
$$\wedge \neg \mathrm{TC2}(m_{ix}, n_{iy})\big)\Big]$$
$$\supset\ \mathrm{TC3}(m_{ix}, n_{iy})$$

Defintion 5 TC3. Property Inconsistency

Continuing with the example in 3.1.2, for m_{ij} to be consistent with n_{ik}, the values for each property of m_{ij} must be type consistent with the value restriction of the properties defined in n_{ik}. The class iso37120:15.2 has a property gci:for_city with a cardinality restriction of 'exactly 1' and its value restricted to class gci:City. Consider the following statements where a_{it} is the gci:for_city, m_{ix} is the instance geo:Toronto, and n_{iy} is the class gci:City. The placename instance geo:Toronto has a direct rdf:type relationship with gci:City.

$$\mathrm{Tri}(m_{ij}, \mathrm{gci{:}\ for\_city}, m_{ix})$$

$$\mathrm{Tri}(n_{ik}, \mathrm{gci{:}\ for\_city}, n_{iy})$$

$$\mathrm{Tri}(m_{ix}, \mathrm{rdf{:}\ type}, n_{iy})$$

Since m_{ix} is the only value of gci:for_city for m_{ij}, thus the cardinality of m_{ij} equals to the cardinality restriction of 'exactly 1' for n_{ik}. The predicate TC2(m_{ix}, n_{iy}) is false since m_{ix} is an instance of n_{iy}. Therefore, m_{ij} is type consistent with n_{ik}.

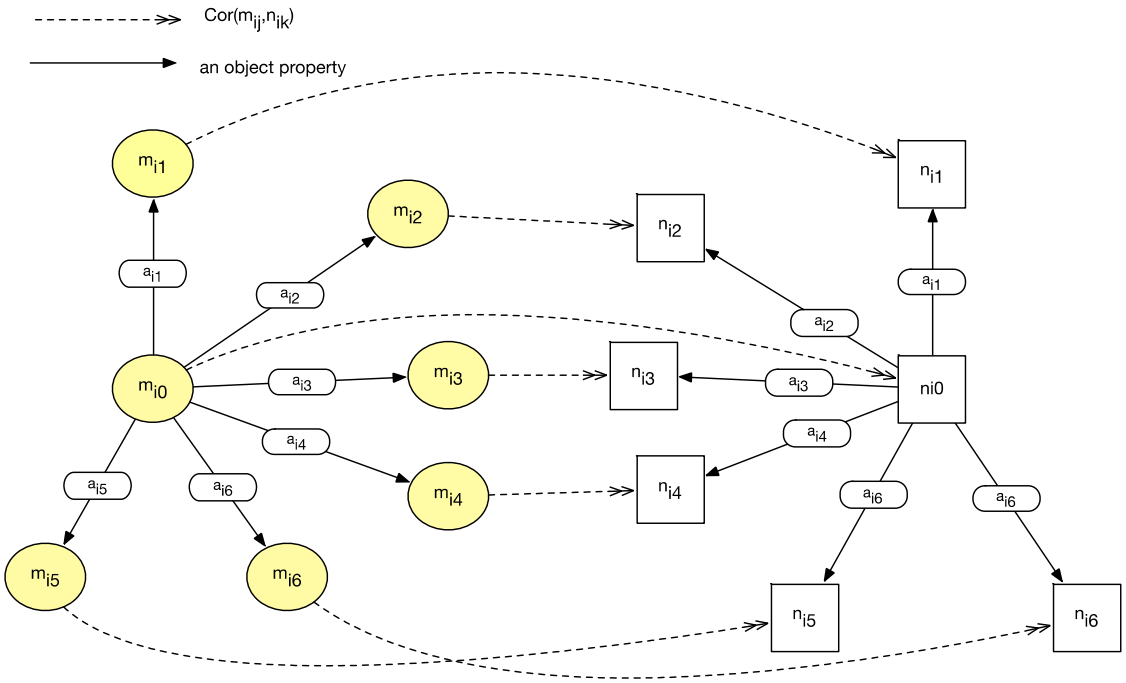


Figure 23 Evaluation of TC2(mij,nik) inconsistency given that Cor(mij, nik), case 2

In the case shown in Figure 23, given that Cor(m_{ij},n_{ik}) where j,k=1…6, m_{i0} is type inconsistent (TC2) with n_{i0} if m_{i0} does not satisfy cardinality and value restriction for any property a_{it}, or any node m_{ij} (j=1…6) is type inconsistent with n_{ik}(k=1…6) such that TC2(m_{i1},n_{i1}) or TC2(m_{i2},n_{i2}) or … or TC2(m_{i6},n_{i6}).

Therefore, an instance m_{ij} is type inconsistent with a definition class n_{ik} if it satisfies either TC2. Instance type inconsistency or TC3. Property inconsistency.

TC. Type Inconsistency

$$\forall m_{ij}\ n_{ik}$$

$$TC2(m_{ij}, n_{ik}) \vee TC3(m_{ij}, n_{ik})$$

$$\supset TC(m_{ij}, n_{ik})$$

Defintion 6 TC. Type Inconsistency

3.2 Temporal Inconsistency

City indicators measure a city's performance during a specific time interval. Data that are believed to be true at the time they are gathered may be found incorrect during other time periods (Fox, 2013). Supporting data that are generated or effective outside the time interval of the indicator are not relevant and are therefore temporally inconsistent. For example, a 15.2 indicator value measured for Toronto in the year of 2013 is temporally inconsistent with its numerator (i.e., homeless population size) if it is generated outside the time interval of 2013 since a portion of the homeless population may have exited the homeless cycle already and have settled in a permanent resident location while other citizens might become homeless due to various reasons.

An indicator value and its supporting data are represented as quantities and measures and are associated with a time interval. The quantities and measures of published indicator data must refer to the same time interval in order to be temporally consistent. That is, the published indicator value and supporting data S_i contains individuals m_{ij} and m_{ik}, which are instances of class om:Quantity or om:Measure, are potentially temporally inconsistent if they refer to the different instances of ot:TemporalEntity. An individual m_{ij} is linked to an instance of ot:TemporalEntity int_i via a property such as gci:for_time_interval, pr:generatedAtTime or kp:effective. We specify predicate Time(m_{ij}, int_i) to identify the relation between an instance m_{ij} and the time interval int_i as follows:

Predicate Time(m_{ij},int_i)

$$\forall m_{ij}\ int_i$$
$$\Big(\text{Type}\big(m_{ij}, om: Quanity\big) \wedge \text{Tri}\big(m_{ij}, \text{gci}: for\_time\_interval, int_i\big)$$
$$\wedge \text{Type}\big(int_i, ot: TemporalEntity\big)\Big) \vee$$
$$\Big(\text{Type}\big(m_{ij}, om: Measure\big)$$
$$\wedge \Big(\text{Tri}\big(m_{ij}, \text{pr}: \text{generateAtTime}, int_i\big)$$
$$\vee \text{Tri}\big(m_{ij}, \text{kp}: \text{effective}, int_i\big)\Big)$$
$$\wedge \text{Type}\big(int_i, ot: TemporalEntity\big)\Big)$$
$$\supset\ \text{Time}\big(m_{ij}, int_i\big)$$

Defintion 7 Time

3.2.1 T1. Non-Overlap Interval Inconsistency

The first type of potential temporal inconsistency deals with non-overlapping intervals of instance m_{ij} and m_{ik} in S_i. Indicator values and supporting data measured during two non-overlapping time intervals (e.g., 2013 and 2016) are not comparable in terms of time since the validity of data do not agree during any time point within the time intervals of m_{ij} and m_{ik}. A time interval int_i that does not overlap another interval int'_i at any time point can be either before or after int'_i (Allen, 1983).

Any two instances of om:Quantity or om:Measure m_{ij}, $m_{ik} \in M_i$ are potentially inconsistent if time interval measured by m_{ij}

- is before the interval int'_i used by m_{ik}, or
- Is after the interval int'_i

We use Before(inti,int'i) and After(inti,int'i) to specify that inti is before and after int'i respectively.

T1. Non-Overlap Interval Inconsistency

$$\forall m_{ij}\ m_{ik}\ int_i\ int'_i$$
$$Type(int_i, ot{:}\ Interval) \wedge Type(int'_i, ot{:}\ Interval) \wedge \mathrm{Time}(\mathrm{m}_{ij}, int_i)$$
$$\wedge \mathrm{Time}(\mathrm{m}_{ik}, int'_i)$$
$$\wedge \left(\mathrm{Before}(int_i, int'_i) \vee \mathrm{After}(int_i, int'_i)\right)$$
$$\supset\ \mathrm{T1}(m_{ij}, m_{ik})$$

Defintion 8 T1. Non-Overlap Interval Inconsistency

In the case where m_{ij} and m_{ik} are linked to int_i and int'_i, for example, represent the instance of interval 2013 and 2016 respectively. The following predicates are true.

$$\mathrm{Time}(\mathrm{m}_{ij}, int_i)$$

$$\mathrm{Time}(\mathrm{m}_{ik}, int'_i)$$

$$\mathrm{Before}(int_i, int'_i)$$

$$After(int'_i, int_i)$$

Given both int_i and int'_i are instances of ot:Interval, m_{ij} and m_{ik} satisfy the definition T1 and therefore are temporally inconsistent in terms of overlapping intervals, i.e., T1(m_{ij},m_{ik}).

This type of inconsistency is a potential temporal inconsistency since there are cases where the indicator intentionally measures data from different time intervals. For example, ISO 37120 city indicator 6.2:Percentage of Students Completing Primary Education: Survival Rate is defined as "the total number of students belonging to a school-cohort who complete the final grade of primary education (numerator) divided by the total number of students belonging to a school-cohort, i.e., those originally enrolled in the first grade of primary education (denominator)" (Fox, 2014). Its numerator measures a population of students from a year that is after the population measured by the denominator.

3.2.2 T2. Interval Equality Inconsistency

In order for instance m_{ij} to be temporally consistent with m_{ik}, the time intervals for m_{ij} and m_{ik} should be equal. The instance m_{ij} and m_{ik} are potentially inconsistent if the time intervals int_i and int'_i of m_{ij} and m_{ik} respectively are not equal. Similar to the case discussed previously, this is a type of potential inconsistency since there are indicators which take measures during different time intervals. Nevertheless, we must evaluate int_i and int'_i of m_{ij} and m_{ik} respectively to verify if the instances int_i and int'_i are equal. Specifically, given that int_i and int'_i are instances of the class ot:Interval, int_i and int'_I must have the same beginning and end. As described in owl-time, an interval int_i has a beginning and end which are both instances of class ot:Instant (Pan & Hobbs, 2004). Interval int_i is equal to int'_i if both the instant of both the beginning and end are equal. We specify interval int_i and int'_i are equal using the predicate IntEqual(int_i,int'_i).

The predicate IntEqual(int_i,int'_i) evaluates the beginning and end of a time interval, both are instances of the class ot:Instant that are characterized by an instance of ot:DateTimeDescription that contains temporal unit (ot:unitType) instances such as ot:unitYear, ot:unitMonth, ot:unitDay, etc. and data properties such as year (ot:year), month (ot:month), day (ot:day), etc. where each is associated with values that represents the datetime parameters of the DateTimeDescription accordingly.

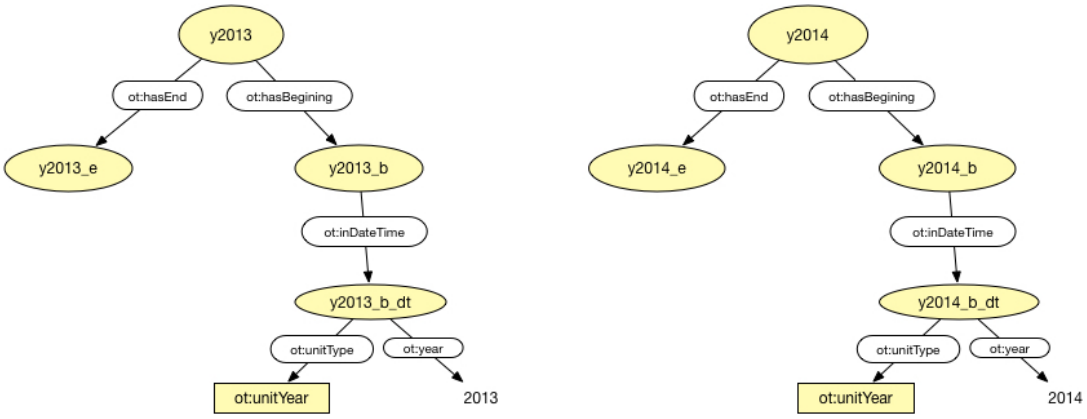


Figure 24 Time Interval Representation

We define interval equality inconsistency T2 as follows:

Any two instances of om:Quanty or om:Measure m_{ij}, $m_{ik} \in M_i$ are potentially inconsistent in terms of interval equality if the interval int_i and int'_i referred by m_{ij} and m_{ik} respectively are not equal.

T2. Interval Equality Inconsistency

$$\forall m_{ij}\ m_{ik}\ int_i\ int'_i$$
$$\mathrm{Type}(int_i, ot{:}\ Interval) \wedge \mathrm{Type}(int'_i, ot{:}\ Interval) \wedge \mathrm{Time}(m_{ij}, int_i) \wedge \mathrm{Time}(m_{ik}, int'_i) \wedge \neg IntEqual(int_i, int'_i)$$
$$\supset\ \mathrm{T2}(m_{ij}, m_{ik})$$

Defintion 9 T2. Interval Equality Inconsistency

Given the same example in the previous section where m_{ij} and m_{ik} are linked to 2013 and 2016 respectively, m_{ij} and m_{ik} are evaluated with the predicate Before(int_i,int'_i) omitted. The intervals int_i and int'_i will be evaluated against the predicate IntEqual(int_i,int'_i). Since int_i and int'_I begins and ends with instants that are not equal, thus IntEqual(int_i,int'_i) returns false which in turn satisfies the predicate T2(m_{ij},m_{ik}). Therefore, according to definition T2, m_{ij} and m_{ik} are inconsistent in terms of interval equality.

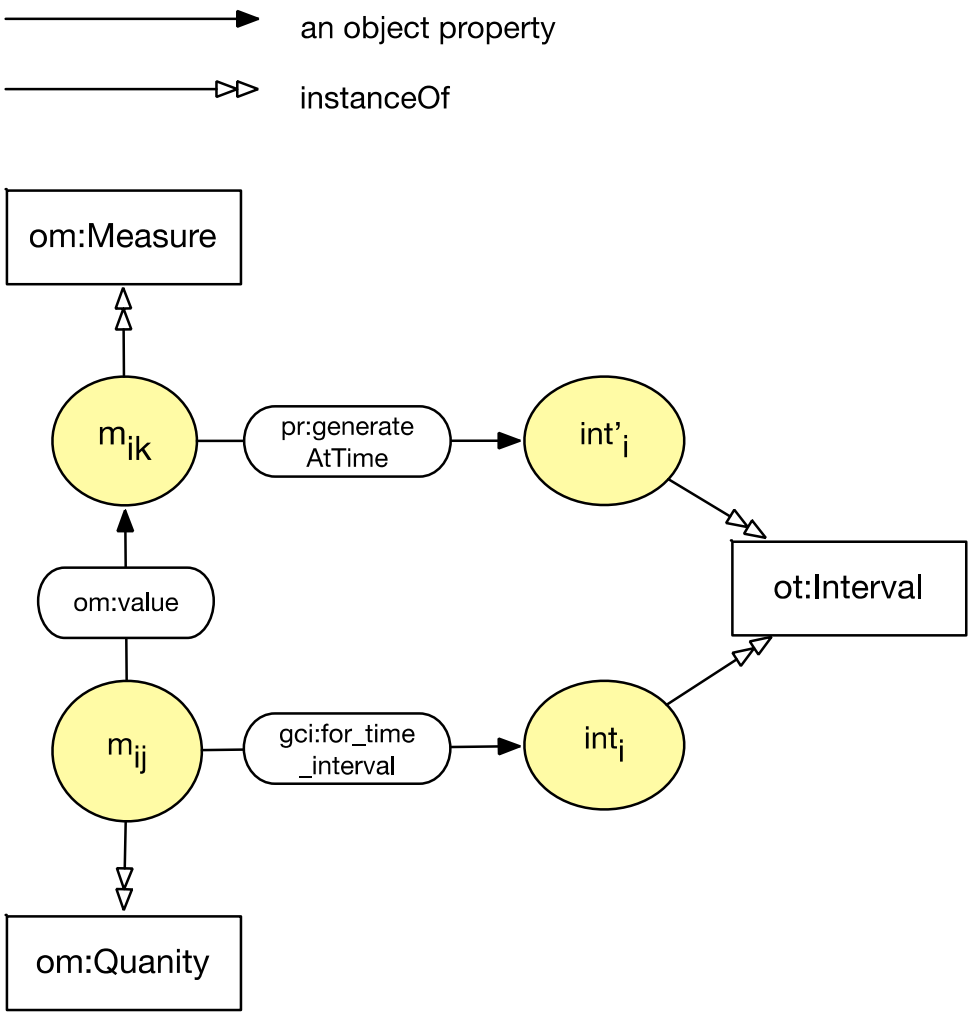


Figure 25 Instances m_{ij} and m_{ik} where T2(m_{ij},m_{ik})

3.2.3 T3. Subinterval Inconsistency

Individuals m_{ij} and m_{ik} can be related to intervals int_i and int'_i that are linked by relations such as during, overlap, starts, finishes, and meets. In each case a portion of interval int_i overlaps with int'_i. Supporting data with intervals with such relations can only be guaranteed to be valid during the portion where the intervals overlap. Information during other portion of the interval is unknown. Thus the supporting data of an indicator is potentially inconsistent with the indicator if the interval referred is a (partial) subinterval of the interval referred by the indicator.

An instance m_{ij} is potentially subinterval inconsistent with m_{ik} if it is related to a time interval int_i that

- is during the interval int'$_i$ for m_{ik}, or
- overlaps with int'$_i$, or
- starts interval int'$_i$, or
- ends interval int'$_i$, or
- meets interval int'$_i$

The first case of potential subinterval inconsistency occurs when an interval is during another. In Figure 3 it was mentioned that an interval int$_i$ is during interval int'$_i$ if the beginning and end of interval int$_i$ is within that of interval int'$_i$. Let int$_i$ represent the interval March 1st to May 31st of 2013 and int'$_i$ be the entire year of 2013. The time interval int$_i$ is during int'$_i$ and int'$_i$ contains int$_i$ as shown in Figure 26 below.

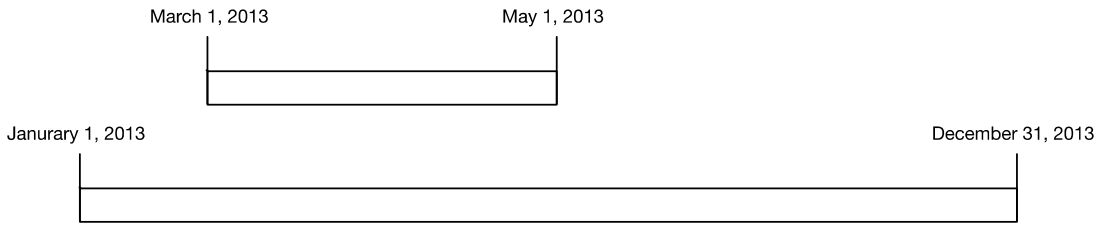


Figure 26 An interval is during another

Suppose interval int$_i$ represents the interval January 2012 to June 2013 which overlaps the time points of interval int'$_i$ which represents the year 2013. Since there are no information indicating the validity of the measure for June to December of 2013, thus an instance m_{ij} linked to int$_i$ is potentially inconsistent with m_{ik} that is linked to int'$_i$.

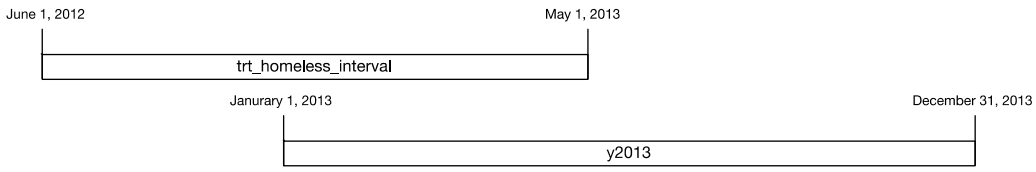


Figure 27 An interval overlaps with another

We use the following predicates to specify interval relationships

During(int$_i$,int'$_i$), Overlaps(int$_i$,int'$_i$), Starts(int$_i$,int'$_i$), Finishes(int$_i$,int'$_i$)

The corresponding inverse predicates are shown below respectively

Contains(int'$_i$,int$_i$), OverlappedBy(int'$_i$,int$_i$), StartedBy(int'$_i$,int$_i$), FinishedBy(int'$_i$,int$_i$)

T3. Subinterval Inconsistency

$$\forall m_{ij}\ m_{ik}\ int_i\ int'_i$$
$$\mathrm{Type}(int_i, ot: Interval) \wedge \mathrm{Type}(int'_i, ot: Interval) \wedge \mathrm{Time}(\mathrm{m}_{ij}, int_i)$$
$$\wedge \mathrm{Time}(\mathrm{m}_{ik}, int'_i)$$
$$\wedge \Big(\big(During(int_i, int'_i) \vee Contains(int'_i, int_i) \big)$$
$$\vee \big(Overlaps(int_i, int'_i) \vee OverlappedBy(int'_i, int_i) \big)$$
$$\vee \big(Starts(int_i, int'_i) \vee StartedBy(int'_i, int_i) \big)$$
$$\vee \big(Finishes(int_i, int'_i) \vee FinishedBy(int'_i, int_i) \big)$$
$$\vee \big(Meets(int_i, int'_i) \vee MetBy(int'_i, int_i) \big) \Big)$$
$$\supset \mathrm{T3}(m_{ij}, m_{ik})$$

Defintion 10 T3. Subinterval Inconsistency

Suppose int$_i$ and int'$_i$ represents the following intervals March to May 2016 and the entire year of 2016 respectively with the following statement:

$$Tri(\mathrm{int}_i, ot: hasBeginning/ot: inDateTime, ins_b)$$

$$Tri(\mathrm{int}_i, ot: hasEnd/ot: inDateTime, ins_e)$$

$$Tri(\mathrm{int}'_i, ot: hasBeginning/ot: inDateTime, ins'_b)$$

$$Tri(\mathrm{int}'_i, ot: hasEnd/ot: inDateTime, ins'_e)$$

$$Tri(ins_b, ot: unitType, ot: unitMonth)$$

$$Tri(ins_b, ot: year, 2016)$$

$$Tri(ins_b, ot: month, 03)$$

$$Tri(ins_e, ot{:}\, unitType, ot{:}\, unitMonth)$$

$$Tri(ins_e, ot{:}\, year, 2016)$$

$$Tri(ins_e, ot{:}\, month, 05)$$

$$Tri(ins'_b, ot{:}\, unitType, ot{:}\, unitMonth)$$

$$Tri(ins_{,b}, ot{:}\, year, 2016)$$

$$Tri(ins'_b, ot{:}\, month, 01)$$

$$Tri(ins'_e, ot{:}\, unitType, ot{:}\, unitMonth)$$

$$Tri(ins'_e, ot{:}\, year, 2016)$$

$$Tri(ins'_e, ot{:}\, month, 12)$$

The interval int_i is therefore during int'_i, that is,

$$During(int_i, int'_i) \vee Contains(int'_i, int_i)$$

Suppose m_{ij} is the homeless population size effective during int_i and m_{ik} is the instance of iso37120:15.2 published by Toronto for interval int'_I, then

$$Time(m_{ij}, int_i) \wedge Time(m_{ik}, int'_i)$$

Therefore the m_{ij} and m_{ik} are potentially inconsistent in terms of subinterval inconsistency since int_i is during int'_I according to definition T3.

3.2.4 T4. Temporal Granularity Inconsistency

Temporal granularity inconsistency arises when time points associated with int_i and int'_i possess different temporal units. As seen in the previous section, time points ins_b and ins_e were linked to the instance ot:unitMonth which represents the temporal unit ‘month’. The values of ot:month were 03 and 05 respectively for ins_b and ins_e since int_i represents the interval March to May of 2016. Interval int_i would be inconsistent if ins_b and ins_e have different temporal units, for

example, if ins_b has a temporal unit ot:unitMonth while ins_e is measured in ot:unitDay, then the interval int_i starts from March 2016 and ends on a day in May, say, May 31st, 2016. Intervals int_i and int'_i will also become incomparable if time points were using different temporal units. Thus we define the following definition T4 to specify temporal granularity inconsistency:

Any two instances m_{ij} and $m_{ik} \in M_i$ are potentially inconsistent in terms of temporal granularity if the time interval int_i and int'_i have different temporal units.

For any time point ins and ins', the predicate Gra(ins, ins') is true if the temporal units of the instants are not equal.

Predicate Gra(ins,ins')

$$\forall ins\ ins'$$
$$Type(ins, ot{:}\ Instant) \wedge Type(ins', ot{:}\ Instant) \wedge Tri(ins, ot{:}\ unitType, tunit)$$
$$\wedge Tri(ins', ot{:}\ unitType, tunit') \wedge \neg Equal(tunit, tunit')$$
$$\supset Gra(ins, ins')$$

Defintion 11 Granularity

For an instant ins and interval int_i, we specify the following predicates to represent that ins is the beginning and end of int_i.

$$Beginning(int_i, ins),\ End(int_i, ins)$$

Let ins_b, ins'_b, ins_e, and ins'_e be the time points that represent the beginning and end of int_i and int'_i respectively. An individual m_{ij} is potentially temporal granularity inconsistent with m_{ik} if the intervals int_i and int'_i consist of time points with different temporal units.

T4. Temporal Granularity Inconsistency

$$\forall m_{ij}\ m_{ik}\ int_i\ int'_i$$
$$\text{Type}(int_i, ot: Interval) \wedge \text{Type}(int'_i, ot: Interval) \wedge \text{Time}(m_{ij}, int_i)$$
$$\wedge \text{Time}(m_{ik}, int'_i) \wedge$$
$$Beginning(int_i, ins_b) \wedge End(int_i,\ ins_e) \wedge Beginning(int'_i, ins'_b)$$
$$\wedge End(int'_i,\ ins'_e) \wedge$$
$$\big(Gra(ins_b, ins'_b) \vee Gra(ins_e, ins'_e) \vee Gra(ins_b, ins_e)$$
$$\vee Gra(ins'_b, ins'_e)\big)$$
$$\supset \text{T4}(m_{ij}, m_{ik})$$

Defintion 12 T4. Temporal Granularity Inconsistency

Consider the example from the previous section with temporal unit of ins_b and ins_e modified to be ot:unitYear instead of ot:unitMonth.

$$Tri(ins_b, ot: unitType, ot: unitYear)$$

$$Tri(ins_b, ot: year, 2016)$$

$$Tri(ins_e, ot: unitType, ot: unitYear)$$

$$Tri(ins_e, ot: year, 2016)$$

Interval inti now represents the year of 2016 with beginning and end time points with 2016 as the values of ot:year property. The interval int'_i was intended to represent the same time interval but using the temporal unit ot:unitMonth, i.e., 01 and 12 were asserted for ot:month on both beginning and end time points ins'_b and ins'_e. There is no information on the values of smaller temporal units such as ot:day, ot:hour, etc. Therefore int'_i may not have covered all time points in the year 2016 as int_i. Since we have the predicate $Gra(ins_b, ins'_b)$ and $Gra(ins_e, ins'_e)$ therefore int_i and int'_i are inconsistent in terms of temporal granularity according to definition T4.

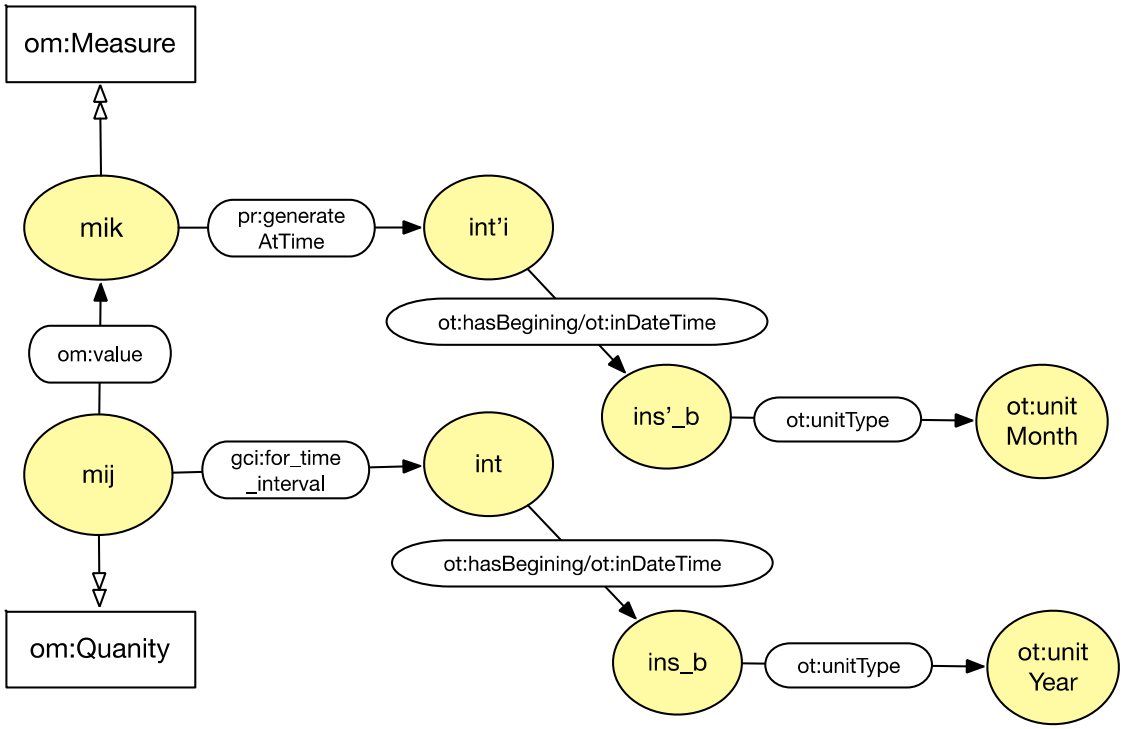


Figure 28 Instances mij and mik where T4(mij,mik)

We have defined four types of temporal inconsistencies, namely non-overlapping interval inconsistency (T1), interval equality inconsistency (T2), subinterval inconsistency (T3) and temporal granularity inconsistency (T4). An instance m_{ij} is temporally inconsistent (TI) with m_{ik} if it is inconsistent with m_{ik} in terms of one of the inconsistency types defined.

TI. Temporal Inconsistency

$$\forall m_{ij}\ m_{ik}$$

$$T1(m_{ij}, m_{ik}) \vee T2(m_{ij}, m_{ik}) \vee T3(m_{ij}, m_{ik}) \vee T4(m_{ij}, m_{ik})$$

$$\supset TI(m_{ij}, m_{ik})$$

Defintion 13 TI. Temporal Inconsistency

3.3 Place Inconsistency

Every city indicator measures performance of a city. Place (i.e., toponym) concepts such as city, province, area, etc. are one of the foundational concepts involved in the definition of a city indicator. An indicator is related to a placename instance (e.g., city) by a 'for_city' property. This placename instance is also incorporated in the representation of supporting data (e.g., Populations) of the indicator. The property 'located in' is used for 'Population' class to describe the location where the target population resides in. The population must refer to a placename instance that is spatially equivalent with the city referred by the indicator in order to be consistent in terms of geographical place.

Place inconsistencies deal with the placenames referred by two instances m_{ij} and $m_{ik} \in M_i$ in the published indicator data. An instance m_{ij} may be related to an instance of geo:Feature city via properties gci:for_city, gci:located_in. m_{ij} is an instance of om:Quantity or gci:Population. We specify Place(m_{ij}, $city_i$) predicate to represent that m_{ij} is linked to $city_i$ as follows:

predicate Place(m_{ij},$city_i$)

$$\forall m_{ij}\ city_i$$
$$Type(city_i, geo{:}\ Feature)\ \wedge$$
$$\left(\begin{array}{c}\left(\mathrm{Type}\left(m_{ij}, \mathrm{om}{:}\ Quantity\right) \wedge \mathrm{Tri}\left(m_{ij}, \mathrm{gci}{:}\ \mathrm{for\_city}, \mathrm{city}_i\right)\right) \vee \\ \left(\mathrm{Type}\left(m_{ij}, \mathrm{gci}{:}\ Population\right) \wedge \mathrm{Tri}\left(m_{ij}, \mathrm{gci}{:}\ \mathrm{located\_in}, \mathrm{city}_i\right)\right)\end{array}\right)$$
$$\supset \mathrm{place}\left(m_{ij}, \mathrm{city}_i\right)$$

Defintion 14 Place

The instances m_{ij} and m_{ik} are inconsistent in terms of place if 1) m_{ij} and m_{ik} are not referring to the same placename instance defined in Geonames, or 2) m_{ij} refers to a placename instance spatially located in the placename instance referred by m_{ik}, or 3) m_{ij} and m_{ik} are linked to the same placename intances with different time intervals. We will discuss each inconsistency type in the following sections.

3.3.1 G1. Place Equality Inconsistency

All references to a city in the supporting data of a single indicator must refer to the same city. For example, the 15.2 indicator for Toronto is measured by measuring the homeless population and city's overall population. Both populations should be located in Toronto. Therefore, if an individual in a published indicator data graph S_i refers to a place instance $city_i$, an instance m_{ik} is inconsistent with m_{ij} if it refers to a different place instance than $city_i$, given that both m_{ij} and m_{ik} are instances of the set of nodes M_i from S_i We specify this type of place inconsistency with the following definition G1:

Instances m_{ij} and $m_{ik} \in M_i$ from S_i are inconsistent if place instances referred by m_{ij} and m_{ik} are not equal. We specify the predicate PlaceEqual($city_i$, $city'_i$) to indicate that $city_i$ and $city'_i$ are the same placename instance.

G1. Place Equality Inconsistency

$$\forall m_{ij}\ m_{ik}\ city_i\ city'_i$$
$$Place(m_{ij}, city_i) \wedge Place(m_{ik}, city'_i) \wedge \neg PlaceEqual(city_i, city'_i)$$
$$\supset \mathrm{G1}(m_{ij}, m_{ik})$$

Defintion 15 G1. Place Equality Inconsistency

Consider the scenario described above, let m_{ij} be the instance of indicator 15.2 which has a gci:for_city property with a value geo:Toronto and m_{ik} be the instance of homeless population measured by m_{ij} that has a property gci:located_in geo:NewYorkCity. Both geo:Toronto and geo:NYC are instances of geo:Feature and gci:City that represent Toronto and New York City respectively[21]. Let $city_i$ and $city'_i$ represent the two cities respectively, the measure of m_{ij} is therefore inconsistent with m_{ik} in terms of place equality since the individuals refer to different cities.

[21] Actual IRI of instances are geo:6167865 and geo:5128581 respectively. We use geo:Toronto and geo:NYC in this thesis for simplicity.

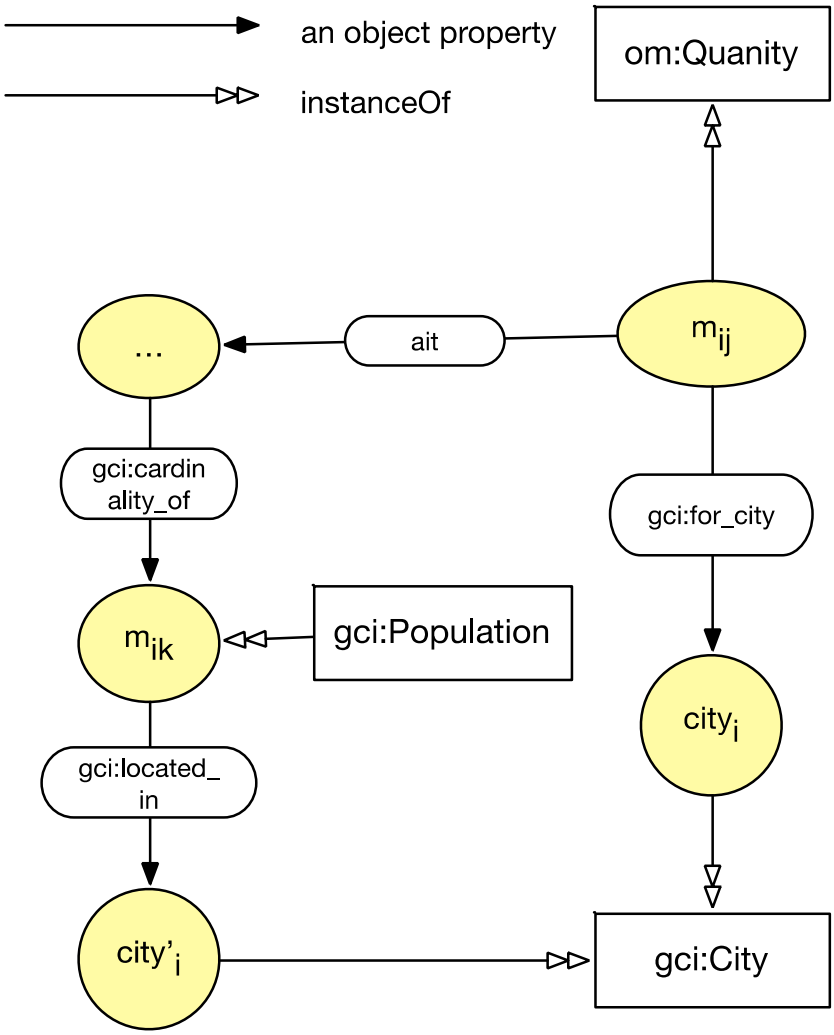


Figure 29 Instances m_{ij} and m_{ik} where G1(mij,mik)

3.3.2 G2. SubPlace Inconsistency

Subplace inconsistency refers to the situation where the placename referred by an instance m_{ik} is an area within the instance of m_{ij}. For example, the population measured by an indicator should be related to place instances $city'_i$ which include all areas within $city_i$ which is referred by the indicator m_{ij}. The measure may not be complete if $city'_i$ is only an area within $city_i$ since not all populations in $city_i$ have been considered. The instance m_{ij} is potentially inconsistent with m_{ik} if $city'_i$ referred by m_{ik} is an area within the $city_i$ referred by m_{ij}. Therefore we specify subplace inconsistency with following definition G2 where we specify the predicate Subplace($city'_i$, $city_i$) for individual $city'_i$ and $city_i$, both are instances of geo:Feature, to represent that $city'_i$ is spatially located in $city_i$:

Any two instances m_{ij} and $m_{ik} \in M_i$ are potentially subplace inconsistent if instance of placename referred by m_{ik} is an area within city referred by m_{ij}

G2. Subplace Inconsistency

$$\forall m_{ij}\ m_{ik}\ city_i\ city'_i$$
$$Place(m_{ij}, city_i) \wedge Place(m_{ik}, city'_i) \wedge Subplace(city'_i, city_i)$$
$$\supset G2(m_{ij}, m_{ik})$$

Defintion 16 G2. Subplace Inconsistency

For example, the homeless population size of Toronto could be an aggregation of homeless population size from different areas in Toronto. The population could be drawn from downtown area, North York area, etc. Therefore, the measured population should be related to all placename instances city'$_i$ that are areas within Toronto such that Subplace(city$_i$,geo:Toronto). Let m_{ij} be the instances of 15.2 and m_{ik} be the homeless population and city'$_i$ be downtown Toronto while city$_i$ is geo:Toronto. Assume m_{ij}, m_{ik}, city$_i$, city'$_i$ are related via the predicates Place(m_{ij}, city$_i$) therefore Place(m_{ik}, city'$_i$) and city'$_i$ is spatially located in city$_I$, i.e., Subplace(city'$_i$, city$_i$). The instance m_{ij} is potentially subplace inconsistent with m_{ik} since the placename city$_i$ referred is an area within the instance city'$_i$ (i.e., geo:Toronto) referred by m_{ik}.

3.3.3 G3. Dynamic Place inconsistency

A city can be related to a time interval via the property kp:effective. Therefore there exist different 'versions' of the same city. For example, the geographical definition of Toronto changed in 1998 after its amalgamation with adjacent municipalities (Fox, 2013). Dynamic placenames were introduced in Fox (2013) where place instances were linked to temporal intervals. Thus the city referred to by the indicator must be related to the same time interval with the city referred by the indicator's supporting data. We define dynamic place inconsistency with the following definition G3:

Any two instances m_{ij} and $m_{ik} \in M_i$ are dynamic place inconsistent if city$_i$ referred by m_{ij} has an effective time interval int$_i$ that is not equal to the effective interval int'$_i$ for city'$_i$ referred by m_{ik}. We define the predicate Revision(city'$_i$, city$_i$) to specify that city'$_i$ is a revision of city$_i$ meaning

that city'$_i$ is initially the same placename as city$_i$ but is effective during an time interval that is different from the time interval of city$_i$.

Predicate Revision($city'_i, city_i$).

$$\forall city_i\ city'_i\ int_i\ int'_i$$
$$Type(int_i, ot{:}\ Interval) \wedge Type(int'_i, ot{:}\ Interval)$$
$$\wedge PlaceEqual(city'_i, city_i) \wedge Tri(city_i, kp{:}\ effective, int_i)$$
$$\wedge Tri(city'_i, kp{:}\ effective, int'_i) \wedge \neg IntEqual(int_i, int'_i)$$
$$\supset \text{Revision}(m_{ij}, m_{ik})$$

Defintion 17 Revision

G3. Dynamic Place inconsistency

$$\forall m_{ij}\ m_{ik}\ city_i\ city'_i$$
$$Place(m_{ij}, city_i) \wedge Place(m_{ik}, city'_i) \wedge Revision(city'_i, city_i)$$
$$\supset \text{G3}(m_{ij}, m_{ik})$$

Defintion 18 G3. Dynamic Place inconsistency

For example, if m$_{ij}$ is the instance of 15.2 which is linked to Toronto in 2013, it is inconsistent in terms of dynamic place if m$_{ik}$ is the instance of homeless population that is linked to Toronto associated with the year 1998. Let inti be the interval 2013 and int'I be 1998, dynamic place inconsistency in shown in Figure 30 below.

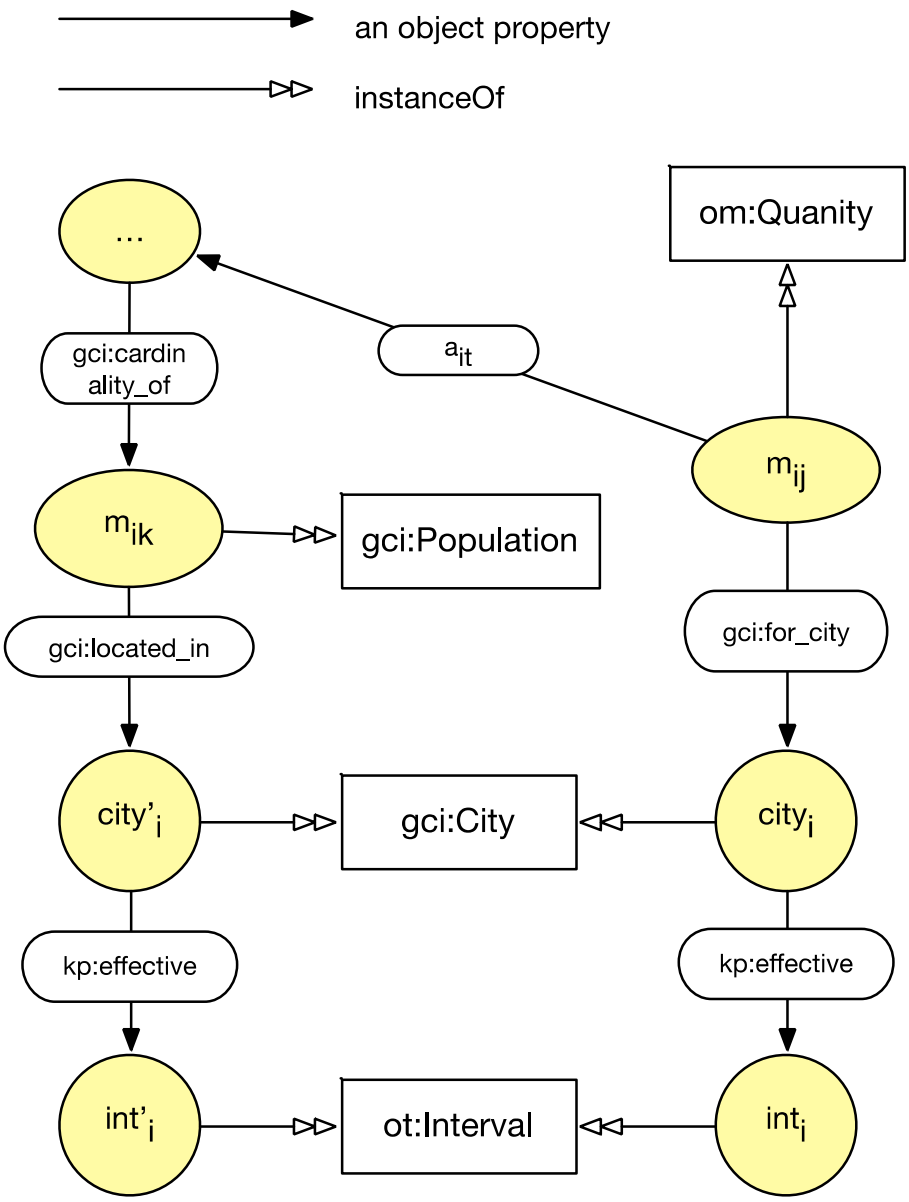


Figure 30 Dynamic place inconsistency

3.3.4 G4. Dynamic Place Temporal Inconsistency

As shown above, cities can be linked to time intervals to represent different 'versions' of the city. Thus the time interval of an indicator must be during the time interval referred by the place instance measured by the indicator.

An instance $m_{ij} \in M_i$ is potentially inconsistent with its measured city $city_i$ if m_{ij} is linked to a time interval int_i that is not during (or contained by) the effective interval int'_i for $city_i$.

G4. Dynamic Place Temporal Inconsistency

$$\forall m_{ij}\ city_i\ int_i\ int'_i$$
$$\mathrm{Type}(int_i, ot{:}\ Interval) \wedge \mathrm{Type}(int'_i, ot{:}\ Interval) \wedge \mathrm{Time}(\mathrm{m}_{ij}, int_i)$$
$$\wedge \mathrm{Tri}(\mathrm{city}_i, kp{:}\ effective, int'_i) \wedge \mathrm{Place}(\mathrm{m}_{ij}, \mathrm{city}_i)$$
$$\wedge \neg(During(int_i, int'_i) \vee Contains(int'_i, int_i))$$
$$\supset \mathrm{G4}(m_{ij}, city_i)$$

Defintion 19 G4. Dynamic Place Temporal Inconsistency

For example, Toronto publishes indicator value for 15.2 for 2013 is dynamic place temporally inconsistent if it measures the instance geo:Toronto[1967-1998] which represents Toronto before the amalgamation occurred in 1998.

We have defined four types of place inconsistencies, namely place equality inconsistency (G1), subplace inconsistency (G2), dynamic place inconsistency (G3), and dynamic place inconsistency (G4). An instance m_{ij} is place inconsistent (PI) with m_{ik} if it is inconsistent with m_{ik} in terms of one of the inconsistency types defined.

PI. Place Inconsistency

$$\forall m_{ij}\ m_{ik}$$
$$G1(m_{ij}, m_{ik}) \vee G2(m_{ij}, m_{ik}) \vee G3(m_{ij}, m_{ik}) \vee G4(m_{ij}, m_{ik})$$
$$\supset PI(m_{ij}, m_{ik})$$

Defintion 20 PI. Place Inconsistency

3.4 Measurement Inconsistency

Measurement consistency considers consistency of unit of measure of the published indicator data. Consistency of units is evaluated based on two aspects of the indicator. First is to evaluate

the units used by the composition of indicator, e.g., the indicator and its numerator and denominator. Second is to evaluate if instances of om:Quantity and their actual values (instances of om:Measure) are applying to the same units. It is also of interest if a multiple or submultiple of the unit referred by the indicator's definition is referred by indicator's supporting data.

Therefore, the set of individuals $M_i \subseteq S_i$ is measurement inconsistent with the indicator if individuals m_{ij} and $m_{ik} \in M_i$ are linked to distinct instances of om:Unit_of_measure $unit_i$ and $unit'_i$ respectively, where m_{ij} and m_{ik} are quantities or measures, or $unit_i$ is a multiple or submultiple unit of $unit'_i$. We use the predicate Unit(m_{ij}, $unit_i$) to indicate that an individual m_{ij} is related to $unit_i$.

Predicate Unit(m_{ij}, $unit_i$)

$$\forall m_{ij}\ n_{ik}\ unit_i$$
$$\left(Type(m_{ij}, om{:}\ Quantity) \vee Type(m_{ij}, om{:}\ Measure)\right)$$
$$\wedge Type(unit_i, n_{ik}) \wedge Subclass(n_{ik}, om{:}\ Unit\_of\_measure)$$
$$\wedge Tri(m_{ij}, om{:}\ unit\_of\_measure, unit_i)$$
$$\supset \text{Unit}(m_{ij}, unit_i)$$

Defintion 21 Unit

3.4.1 M1. Quantity Measure Inconsistency

As depicted in Figure 31, both a Quantity and its Measure must refer to the same unit of measure.

M1: Any two instances m_{ij} and $m_{ik} \in$ Mi are measurement inconsistent if an instance of Quantity m_{ij} has a unit of measure $unit_i$ that is different from the Measure's unit of measure $unit'_i$.

M1. Quantity Measure Inconsistency

$$\forall m_{ij}\ m_{ik}\ unit_i\ unit'_i$$
$$Type(m_{ij}, om{:}\ Quantity) \wedge Type(m_{ik}, om{:}\ Measure)$$
$$\wedge Tri(m_{ij}, om{:}\ value, m_{ik}) \wedge \mathrm{Unit}(m_{ij}, unit_i)$$
$$\wedge \mathrm{Unit}(m_{ik}, unit'_i) \wedge \neg Equal(unit_i, unit'_i)$$
$$\supset \mathrm{M1}(m_{ij}, m_{ik})$$

Defintion 22 M1. Quantity Measure Inconsistency

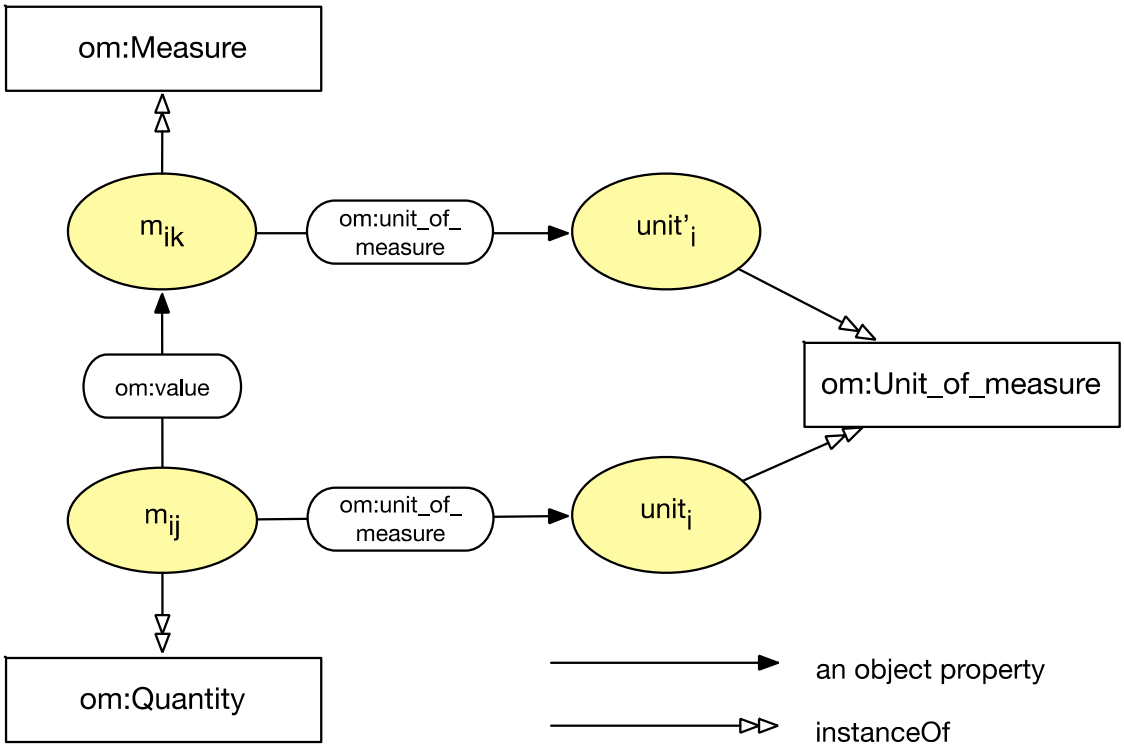


Figure 31 Instances m_{ij} and m_{ik} where M1(m_{ij},m_{ik})

3.4.2 M2. Indicator Unit Component Inconsistency

The supporting data of an indicator is related to the instance of the indicator via properties such as numerator and denominator for a ratio indicator, or factors for a multiplication indicator. The units of measure of such indicators should be linked to the unit of measure of the supporting data via the same properties.

Any two instances of om:Quantity m_{ij} and $m_{ik} \in M_i$ where m_{ij} is connected to m_{ik} via a property a_{it}(e.g., numerator, denominator), m_{ij} and m_{ik} has a unit of measure $unit_i$ and $unit'_i$ respectively. The instance m_{ij} is inconsistent with m_{ik} if definition of $unit_i$ and $unit'_i$ are not connected by a_{it}.

M2. Indicator Unit Component Inconsistency

$$\forall m_{ij}\ m_{ik}\ unit_i\ unit'_i$$
$$Type(m_{ij}, om{:}\ Quantity) \wedge Type(m_{ik}, om{:}\ Quantity) \wedge Type(unit_i, n_i)$$
$$\wedge Type(unit'_i, n'_i) \wedge \mathrm{Unit}(m_{ij}, unit_i) \wedge \mathrm{Unit}(m_{ik}, unit'_i) \wedge$$
$$\exists a_{it}\ \left(Tri(m_{ij}, a_{it}, m_{ik}) \supset \neg Tri(n_i, a_{it}, n'_i)\right)$$
$$\supset \mathrm{M2}(m_{ij}, m_{ik})$$

Defintion 23 M2. Indicator Unit Component Inconsistency

Suppose m_{ij} is the instance of 15.2 indicator and m_{ik} and m_{iv} are the instances of homeless population size and city population size respectively. We need to evaluate if the unit referred by m_{ij}, i.e., gci:population_ratio_unit is compatible with the unit referred by m_{ik} and m_{iv}, i.e., gci:population_cardinality_unit. Let $unit_i$ be gci:'population_ratio_unit', which has both its numerator and denominator as 'gci:population_cardinality_unit' represented by $unit'_i$. Since the m_{ij} is linked to m_{ik} via the property om:numerator and m_{iv} with om:denominator, the unit of measure associated with m_{ik} and m_{iv} must be $unit'_i$ which is linked to $unit_i$ through om:numerator and om:denominator. The instance m_{ij} will be inconsistent if one of m_{ik} and m_{iv} has a different unit than $unit'_i$.

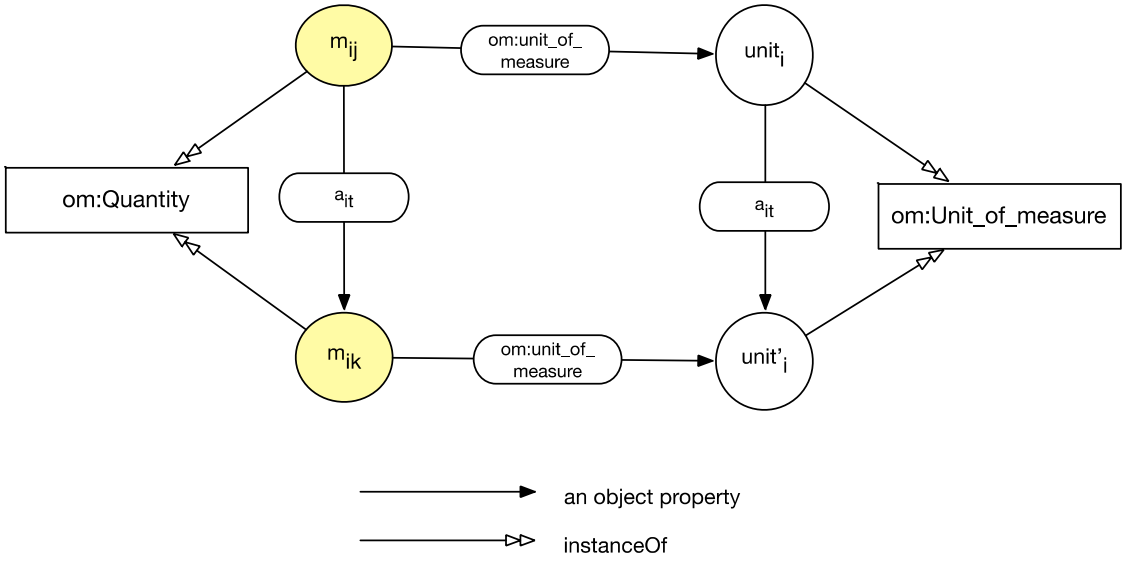


Figure 32 Indicator component inconsistency where T2(m_{ij},m_{ik})

3.4.3 M3. Singular Unit Inconsistency

Another case of measurement inconsistency is when an instance m_{ij} has a unit of measure $unit_i$ that is a multiple or submultiple of the unit defined in its corresponding definition class n_{ik}. It is therefore inconsistent with its corresponding class n_{ik} in terms of singular unit. We specify singular unit inconsistency with the following definition

M4: Given Cor(m_{ij},n_{ik}), $m_{ij} \in S_i$ and n_{ik}, $n_{iv} \in D_i$, are inconsistent if the unit of measure $unit_i$ used by m_{ij} is an instance of a class that is a singular unit of n_{iv} which is a subclass of om:Unit_of_measure that is related to n_{ik} via the property om:unit_of_measure.

We specify the predicate Su($unit_i$,n_{iv}) to represent that $unit_i$ is an instance of a class that is a multiple or submultiple unit of class n_{iv} given that n_{iv} is a subclass of om:Unit_of_measure.

M3. Singular Unit Inconsistency

$$\forall m_{ij}\ n_{ik}\ n_{iv}\ unit_i$$
$$Cor(m_{ij}, n_{ik}) \wedge \text{Unit}(m_{ij}, unit_i) \wedge \text{Tri}(n_{ik}, om{:}\ unit\_of\_measure, n_{iv})$$
$$\wedge Subclass(n_{iv}, om{:}\ Unit\_of\_measure) \wedge Su(unit_i, n_{iv})$$
$$\supset \text{M3}(m_{ij}, m_{ik})$$

Defintion 24 M3. Singular Unit Inconsistency

Published indicator and its supporting data may refer to unit of measure that is a multiple or submultiple of the unit specified in the definition. Let $unit_i$ be an instance of the class gci:kilopc and niv be the class gci:'population_cardinality_unit'(pc). The class n_{iv} is related to gci:kilopc via the property om:'singular_unit'. Suppose m_{ij} is instance of its corresponding definition class of homeless population size n_{ik}, while n_{iv} is the unit defined by n_{ik}. The instance m_{ij} is inconsistent with n_{ik} if the predicate Unit(m_{ij},$unit_i$) is true since it is using the unit kilopc instead of pc as defined by n_{ik}.

This type of measurement inconsistency is also captured by the evaluation of type inconsistency TC2 as discussed earlier since a unit of measure other than pc defined by n_{ik} was referenced. In the case of a unit such as kilopc was used by m_{ij}, we wish to return the nature of inconsistency such that the unit is a multiple or submultiple of the unit of measure specified by the definition.

We have defined three types of measurement inconsistencies, namely unit of quantities and measures inconsistency (M1), quantity component inconsistency (M2), and singular unit inconsistency (M3). An instance m_{ij} is inconsistent in terms of measurement (MI) with m_{ik} if it is inconsistent in terms of one of the measurement inconsistency types defined.

MI. Measurement Inconsistency

$\forall m_{ij}\ m_{ik}$

$$M1(m_{ij}, m_{ik}) \vee M2(m_{ij}, m_{ik}) \vee M3(m_{ij}, m_{ik})$$

$\supset MI(m_{ij}, m_{ik})$

Defintion 25 MI. Measurement Inconsistency

3.5 Summary

In this chapter we defined types of definitional inconsistencies of a published city indicator with respect to the indicator's definition. Definitional consistency refers to whether an indicator is consistent with respect to provided definition, (i.e., ISO 37120 standards) and is also internally consistent. Indicator value and supporting data are considered to be inconsistent with its definition if it satisfies at least one of the following inconsistency types. Prolog implementation of each inconsistency types can be found in Appendix III.

Correspondence Inconsistency: where there are no correspondence detected between nodes in the indicator's definition and city published indicator data. This means that not all components in the definition are covered by the published indicator data.

Inconsistency	**Description**
CI. Correspondence Inconsistency	for any corresponding nodes $m_{ij} \in M_i$ and $n_{ik} \in N_i$, there exists a class n_{iy} that is linked to n_{ik} via property a_{it} where there is no node m_{ix} linked to m_{ij} that corresponds to n_{iy}.

TC. Type Inconsistency: the type of all instances from the published indicator are instances from, an equivalent class, a subclass, or subsumed by classes defined by the indicator's definition.

Type Inconsistency	**Description**

TC1. Class Type Inconsistency	a class X is type inconsistent with class Y if X is not the same, an equivalent class, nor a subclass of Y, or X is not subsumed by Y.
TC2. Instance Type Inconsistency	Let m_{ij}, c_{iv}, and n_{ik} be nodes of M_i, C_i, and N_i respectively, m_{ij} is instance type inconsistent if: • there does not exist a direct rdf:type relation between m_{ij} and n_{ik}, and • m_{ij} is not an instance of n_{ik}, and • m_{ij} is an instance of c_{iv}, and c_{iv} is type inconsistent with n_{ik}
TC3. Property Inconsistency	An instance $m_{ij} \in M_i$ is potentially inconsistent with its corresponding definition class $n_{ik} \in N_i$ if there exist a necessary property a_{it} defined in n_{ik} that satisfies one of the following conditions • a_{it} does not exist in m_{ij}, or • the cardinality of ait for m_{ij} does not satisfy the cardinality restriction defined in n_{ik}, or • m_{ij} does not satisfy the value restriction of a_{it} defined in n_{ik}

TI. Temporal Inconsistency: an indicator is inconsistent with its definition in terms of temporal entities. Temporal inconsistencies occur when supporting data are measured for a time interval that is not equal to, is a subinterval of, or has different temporal unit with the time interval measured by the indicator.

Temporal Inconsistency	**Description**
T1. Interval Overlap Inconsistency	Any two instances of om:Quantity or om:Measure m_{ij}, $m_{ik} \in M_i$ are potentially inconsistent if time interval measured by m_{ij} • is before the interval int'_i used by m_{ik}, or • Is after the interval int'_i
T2. Interval Equality Inconsistency	Any two instances of om:Quanty or om:Measure m_{ij}, $m_{ik} \in M_i$ are potentially inconsistent in terms of interval equality if the interval inti and int'_i referred by m_{ij} and m_{ik} respectively are not equal

T3. Subinterval Inconsistency	An instance m_{ij} is potentially subinterval inconsistent with m_{ik} if it is related to a time interval int_i that • is during the interval int'_i for m_{ik}, or • overlaps with int'_i, or • starts interval int'_i, or • ends interval int'_i, or • meets interval int'_i
T4. Temporal Granularity Inconsistency	Any two instances m_{ij} and $m_{ik} \in M_i$ are potentially inconsistent in terms of temporal granularity if the time interval int_i and int'_i have different temporal units.

PI. Placename Inconsistency: an indicator's geographical concepts are inconsistent with the indicator's definition. Place inconsistencies include 1) Place equality inconsistency where a population was drawn from a place different than the city specified by the indicator, 2) Subplace inconsistentcy, where a population is drawn from areas within the city specified by the indicator, or 3) Dynamic Placename inconsistency which considers the case where population was drawn from the city that is referred to a different time.

Placename Inconsistency	**Description**
G1. Place Equality Inconsistency	Instances m_{ij} and $m_{ik} \in M_i$ from S_i are inconsistent if place instances referred by m_{ij} and m_{ik} are not equal.
G2. SubPlace Inconsistency	Any two instances m_{ij} and $m_{ik} \in M_i$ are potentially subplace inconsistent if instance of placename referred by m_{ik} is an area within city referred by m_{ij}
G3. Dynamic Place Inconsistency	Any two instances m_{ij} and $m_{ik} \in M_i$ are dynamic place inconsistent if $city_i$ referred by m_{ij} has an effective time interval int_i that is not equal to the effective interval int'_i for $city'_i$ referred by m_{ik}.

G4. Dynamic Place Temporal Inconsistency	An instance $m_{ij} \in M_i$ is potentially inconsistent with its measured city $city_i$ if m_{ij} is linked to a time interval int_i that is not during (or contained by) the effective interval int'_i for $city_i$.

MI. Measurement Inconsistency: Units of measure used to measure the values of the indicator and its supporting data are internally consistent and consistent with respect to the indicator's definition. Measurement inconsistency may occur due to different units of measure used for quantities and its measure, or the use of multiples or submultiple of the unit defined by the indicator's definition.

Measurement Inconsistency	**Description**
M1. Quantity Measure Inconsistency	Any two instances m_{ij} and $m_{ik} \in Mi$ are measurement inconsistent if an instance of Quantity m_{ij} has a unit of measure $unit_i$ that is different from the Measure's unit of measure $unit'_i$.
M2. Indicator Unit Component Inconsistency	Any two instances of om:Quantity m_{ij} and $m_{ik} \in M_i$ where m_{ij} is connected to m_{ik} via a property a_{it}(e.g., numerator, denominator), m_{ij} and m_{ik} has a unit of measure $unit_i$ and $unit'_i$ respectively. The instance m_{ij} is inconsistent with m_{ik} if definition of $unit_i$ and $unit'_i$ are not connected by a_{it}.
M3. Singular Unit Consistency	Given Cor(m_{ij},n_{ik}), $m_{ij} \in S_i$ and n_{ik}, $n_{iv} \in D_i$, are potential inconsistent if the unit of measure $unit_i$ used by m_{ij} is an instance of a class that is a singular unit of n_{iv} which is a subclass of om:Unit_of_measure that is related to n_{ik} via the property om:unit_of_measure.

In the next chapter we will discuss transversal and longitudinal consistency analysis which evaluates published indicator instances and city specific definition between different cities and a city over different time periods.

Chapter 4
Transversal and Longitudinal Consistency

4 Transversal and Longitudinal Consistency

When comparing indicator data published by the same city at two different times or two cities at the same time on the semantic web, it is crucial to ensure they are consistent with each other. We call this longitudinal and transversal consistency respectively. For example, the ISO 37120 15.2 indicator is longitudinally inconsistent if the geospatial dimensions of the city have changed over time, which means that homeless population included are from different locations. The indicator is transversally inconsistent if the definition of a homeless person differs between two cities.

The following formally defines the different types of longitudinal and transversal inconsistencies that may arise.

4.1 Transversal Consistency Analysis

Consider the evaluation of city indicator values and supporting data used to derive them, published by City 1 and City 2, represented as instances of classes from the indicator's definition, theme general knowledge, and city specific knowledge. We assume theme and city specific ontologies used to represent the indicator data provided by City 1 and City 2, are logically consistent. We also assume that published indicator values and supporting data provided by both City 1 and City 2 are definitional consistent with the indicator's definition.

Similar to definitional consistency analysis, published indicator data from both cities are represented as graphs where nodes represent instances, classes or literals, and arcs represent properties. Let S^{city1}_i be the graph that represents the indicator data published by City 1 and S^{city2}_i be the graph that represents indicator data published by City 2 for an indicator i.

Indicator values and supporting data published by a city are evaluated against corresponding data published by another city. That is, nodes m^{city1}_{ij} in S^{city1}_i will be evaluated with respect to corresponding nodes m^{city2}_{ik} in S^{city2}_i. Inter-indicator correspondence between nodes of S^{city1}_i and S^{city2}_i is represented by $Cor\_I(m^{city1}_{ij}, m^{city2}_{ik})$. Table 6 below lists the notation used in transversal consistency analysis.

Published City Indicator Data and City Specific Knowledge

- Let S be the set of all published indicator data
- Let O^{C1} be the City Specific ontology in S from City 1
- Let O^{C2} be the City Specific ontology in S from City 2
- Let $S^{city1}{}_i$ be the graph that represents the data used to derive indicator i for City 1
- Let $S^{city2}{}_i$ be the graph that represents the data used to derive indicator i for City 2
- $S^{city1}{}_i$ is composed of a set of nodes $A^{city1}{}_i$, $C^{city1}{}_i$, $M^{city1}{}_i$ and N_i, where
  - $A^{city1}{}_i$ is a set of properties in $S^{city1}{}_i$
  - $M^{city1}{}_i$ is a set of individuals in $S^{city1}{}_i$
  - $C^{city1}{}_i \subseteq class(O^{C1}) \cup indiv(O^{C1}) \cup literal(O^{C1})$, where
    - $class(O^{C1})$ is the set of classes defined in O^{C1},
    - $indiv(O^{C1})$ is a set of individuals in O^{C1}, and
    - $literal(O^{C1})$ is a set of literals in O^{c1}
- $S^{city2}{}_i$ is composed of a set of nodes $A^{city2}{}_i$, $C^{city2}{}_i$, $M^{city2}{}_i$, and N_i, where
  - $A^{city2}{}_i$ is a set of properties in $S^{city2}{}_i$
  - $M^{city2}{}_i$ is a set of individuals in $S^{city2}{}_i$
  - $C^{city2}{}_i \subseteq class(O^{C2}) \cup indiv(O^{C2}) \cup literal(O^{C2})$ where
    - $class(O^{C2})$ is the set of classes defined in O^{C2}, and
    - $indiv(O^{C2})$ is a set of individuals in O^{C2}, and
    - $literals(O^{C2})$ is a set of literals in O^{c2}
- $C^{city1c}{}_i \subseteq class(O^{C1}) \subseteq C^{city1}{}_i$
- $C^{city2c}{}_i \subseteq class(O^{C2}) \subseteq C^{city2}{}_i$

Table 6 Notation for transversal consistency analysis

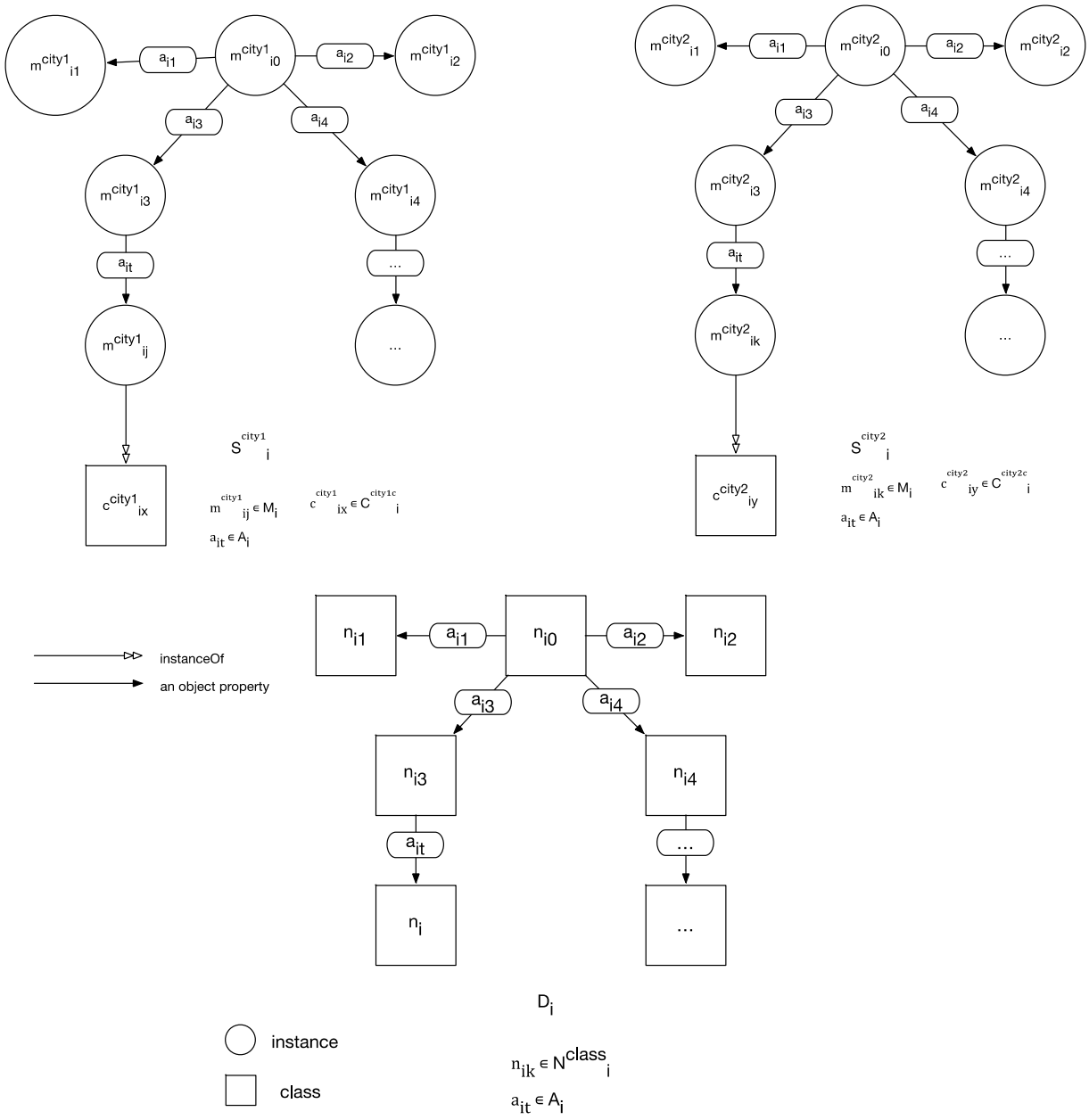


Table 7 Indicator value and supporting data from different cities comply to definition D_i

As mentioned in Chapter 3, determining correspondence between instances of two graphs can be difficult. Determining correspondence can be simplified by identifying the subset of instances we wish to determine consistency. We refer to these as instances of Primary classes. We denote Primary classes as Prim(n_{ik}) where $n_{ik} \in N^{class}{}_i \subseteq D_i$. Examples of primary classes include indicator and population size. Examples of secondary instances, are instances of om:Measure and om:Unit_of_measure. We denote Secondary classes as Sec(n_{ik}) where $n_{ik} \in N^{class}{}_i \subseteq D_i$. Inter-

indicaotr correspondences are only established for instances of Primary classes. We assume in the remainder of this chapter that all individuals $m_{ij} \in M_i$, are instances of $Prim(n_{ik})$ unless stated otherwise.

Predicate $Cor\_I(m^{city1}_{ij}, m^{city2}_{ik})$

$\forall\ m^{city1}_{ij}, m^{city2}_{ik}, n_{ik}$

$(Type(m^{city1}_{ij}, n_{ik}) \wedge Type(m^{city2}_{ij}, n_{ik}) \wedge Prim(n_{ik})) \vee$

$\exists\ m^{city1}_{ix}\ m^{city2}_{iy}\ a_{it}$

$(Cor\_I(m^{city1}_{ix}, m^{city2}_{iy}) \wedge Tri(m^{city1}_{ix}, a_{it}, m^{city1}_{ij}) \wedge Tri(m^{city2}_{iy}, a_{it}, m^{city2}_{ik}))$

$\supset Cor\_I(m^{city1}_{ij}, m^{city2}_{ik})$

Defintion 26 Inter-indicator correspondence

Inter-indicator correspondence inconsistency may exist between the indicator data published by the cities if no correspondence $Cor\_I(m^{city1}_{ij}, m^{city2}_{ik})$ can be found for m^{city1}_{ij} and m^{city2}_{ik}. This means that one city is providing more information than the other.

Inter_CI. Inter-indicator Correspondence Inconsistency

$\forall\ m^{city1}_{ij}, m^{city2}_{ik}, n_{ik}$

$Cor\_I(m^{city1}_{ij}, m^{city2}_{ik}) \wedge$

$\exists\ m^{city1}_{ix}\ m^{city2}_{iy}\ a_{it}$

$(\quad Tri(m^{city2}_{ik}, a_{it}, m^{city2}_{iy}) \supset$

$\neg\exists\ m^{city1}_{ix}\ (Tri(m^{city1}_{ij}, a_{it}, m^{city1}_{ix}) \wedge Cor\_I(m^{city1}_{ix}, m^{city2}_{iy}))\quad)$

$\supset Inter\_CI(m^{city1}_{ij}, m^{city2}_{ik})$

Defintion 27 Inter_CI. Inter-indicator Correpondence Inconsistency

In the following, we define different types of transversal inconsistencies. Similar to definitional consistency analysis, categories of inconsistencies include transversal type inconsistency, temporal inconsistency, geographical, and measurement inconsistency.

4.1.1 Trans_TC. Transversal Type Inconsistency

Transversal type inconsistency evaluates type inconsistency between corresponding instances representing indicator data published by City 1 and City 2. Although both set of instances from S^{city1}_i and S^{city2}_i are definitional consistent, we still need to evaluate if both City 1 and City 2 are measuring the same population since definition of concepts such as homeless person, shelters may differ between cities.

An instance $m^{city1}_{ij} \in M^{city1}_i$ is type inconsistent with its corresponding instance $m^{city2}_{ik} \in M^{city2}_i$ if m^{city1}_{ij} is an instance of a class $c^{city1}_i \in C^{city1}_i$ and m^{city2}_{ik} is an instance of a class $c^{city2}_i \in C^{city2}_i$ such that c^{city1}_i is not equivalent to c^{city2}_i.

Trans_TC. Transversal Type Inconsistency

$\forall m^{city1}_{ij}\ m^{city2}_{ik}$

$\quad Cor\_I(m^{city1}_{ij}, m^{city2}_{ik}) \wedge$

$\quad \exists c^{city1}_{i}\ c^{city2}_{i}$

$\quad (\quad Type(m^{city1}_{ij}, c^{city1}_{i}) \wedge Type(m^{city2}_{ik}, c^{city2}_{i}) \wedge$

$\quad \neg Equal(c^{city1}_{i}, c^{city2}_{i}) \quad)$

$\supset Trans\_TC(m^{city1}_{ij}, m^{city2}_{ik})$

Defintion 28 Trans_TC. Transversal Type Inconsistency

According to the ISO 37120 definition of 15.2 indicator represented with the GCI ontologies, a homeless population is defined by a homeless person. But a homeless person is defined differently between Toronto and New York City according to city specific knowledge provided by the cities (City of Toronto, 2013; Coalition of Homeless, 2016). The definition of a homeless person is represented using the classes 'Toronto_homeless_person' and NYC_homeless_person' for Toronto and New York City respectively. 'Toronto_homeless_person' was described in Chapter 3 as a homeless person who lives outdoor, in an emergency homeless shelter, a VAW shelter or a treatment facility. 'NYC_homeless_person' is defined to be homeless person who lives in a single adult shelter or a family shelter. Both cities satisfy the definition of homelessness outlined by ISO 37120 (i.e., are definitional consistent) but disagree on the specific types of homeless shelters that characterize their homeless population. Therefore, the set of classes that restrict the property gcis:livesIn need to be compared between Toronto and NYC in order to verify the definition of homeless person is transversally consistent.

We define Toronto homeless shelter and NYC homeless shelter as follow:

Toronto homeless shelter = emergency shelter ∨ VAW shelter ∨ treatment facility

NYC homeless shelter = single adult shelter ∨ family shelter

Let m^{city1}_{ij} be an instance of Toronto homeless person and m^{city2}_{ik} be an instance of NYC homeless person with the following triples:

Tri($m^{city1}_{ij,}$ gcis:livesIn, m^{city1}_{ix})

Tri($m^{city2}_{ik,}$ gcis:livesIn, m^{city2}_{iy})

Tri(Toronto homeless person, gcis:livesIn, Toronto homeless shelter)

Tri(NYC homeless person, gcis:livesIn, NYC homeless shelter)

Where m^{city1}_{ix} and m^{city2}_{iy} are instances of Toronto and NYC homeless shelters respectively. The cardinality restriction of gcis:livesIn for both classes are exactly 1. Given that Cor_I(m^{city1}_{ij}, m^{city2}_{ik}), we first evaluate the class Toronto_homeless_person and NYC_homeless_person. Since the two classes are not the same class, the value restriction of properties will be evaluated for equivalency. The classes are linked to Toronto homeless shelter and NYC homeless shelter via the property gcis:livesIn.

Let c^{trt}_{iu} and c^{nyc}_{iv} represent homeless person class for Toronto and NYC respectively. Let c^{trt}_{ix} be the class represent the class 'outdoor or Toronto homeless shelter' and c^{nyc}_{iy} be the NYC homeless shelter class. Transversal consistency analysis will then evaluate the restrictions of property a_{it} which is gcis:livesIn in this case. We have *card(*c^{trt}_{iu}*,gci:livesIn) = card(*c^{nyc}_{iv}*, gci:livesIn)* since both classes have exactly 1 as cardinality restriction for gcis:livesIn. The value restrictions of gcis:livesIn are classes c^{trt}_{ix} and c^{nyc}_{iy} which are neither the same class nor equivalent class since different types of homeless shelters are referred by Toronto and NYC. Therefore Trans_TC(m^{city1}_{ix}, m^{city2}_{iy}) is true and the instance m^{city1}_{ij} and m^{city2}_{ik} are transversally type inconsistent.

4.1.2 Trans_TI. Transversal Temporal Inconsistency

In Definitional consistency analysis, we have determined whether the quantities and measures of published indicator data refer to the same time interval. In transversal consistency analysis, temporal inconsistency is determined between the indicator values published by City 1 and City 2. The time interval referred by the indicators and supporting data should be the same between

the two cities being compared. For example, if the 15.2 Homeless ratio indicator data published by Toronto was generated in the year 2013 then the indicator data being compared published by New York City should also be generated in 2013 and are valid throughout the entire year. Corresponding individuals m^{city1}_{ij} and m^{city2}_{ik}, which are instances of the indicator iso37120:15.2, are temporally inconsistent if they refer to the different time intervals. That is, T2(m^{city1}_{ij}, m^{city2}_{ik}) must be false.[22]

As shown in the Figure 33 below, the time interval that the indicators referenced to are represented with the instances y2013_trt and y2013_nyc respectively. Both are instances of the class ot:Interval with a beginning (ot:hasBeginning) and end (ot:hasEnd) that link to an instance of ot:Instant. The date-time representation (instances of ot:DateTimeDescription) of both beginning and end of the two time intervals must have the same temporal unit and values for property ot:year. E.g., both have 'unitYear' as value for 'unitType' and the value of property ot:year is '2013'. Let the m^{city1}_{ij} be an instance of data published by Toronto and m^{city2}_{ik} be that of NYC, T2(m^{city1}_{ij}, m^{city2}_{ik}) is evaluated to be false since the intervals are equal.

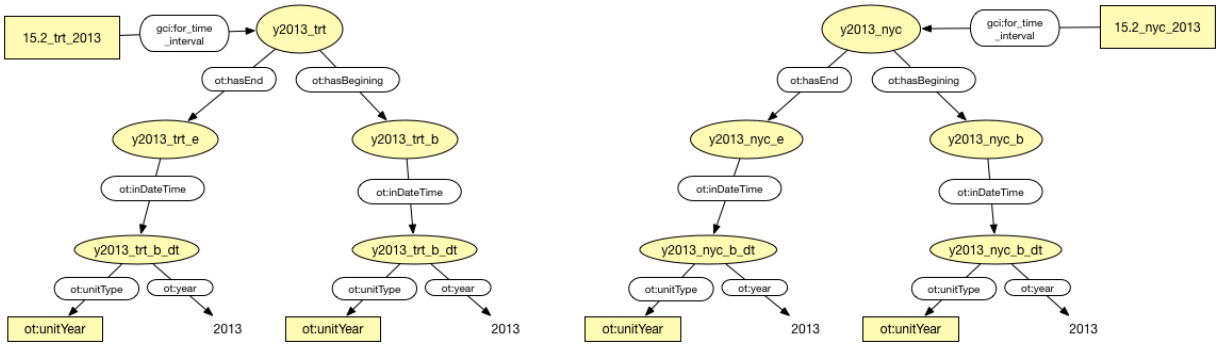


Figure 33 Time interval linked by indicator of Toronto and NYC

The same type of temporal inconsistencies defined in definitional consistency analysis can be applied between m^{city1}_{ij} and m^{city2}_{ik}, temporal inconsistencies T1, T3, and T4 can be applied to the indicator instances in order to verify if the time intervals are inconsistent in terms of overlap, subintervals, or temporal granularity inconsistency.

[22] It could be the case that a comparison of two cities at different time intervals is desired. Never the less, an inconsistency would be determined.

Trans_TI. Transversal Temporal Inconsistency

$\forall m^{city1}_{ij}\ m^{city2}_{ik}$

$Cor\_I(m^{city1}_{ij}, m^{city2}_{ik}) \wedge$

$(T1(m^{city1}_{ij}, m^{city2}_{ik}) \vee T2(m^{city1}_{ij}, m^{city2}_{ik}) \vee T3(m^{city1}_{ij}, m^{city2}_{ik}) \vee T4(m^{city1}_{ij}, m^{city2}_{ik}))$

$\supset Trans\_TI(m^{city1}_{ij}, m^{city2}_{ik})$

Defintion 29 Trans_TI. Transversal Temporal Inconsistency

4.1.3 Trans_PI. Transversal Place Inconsistency

Transversal place inconsistency identifies geographical inconsistencies between published indicators from two different cities. Place inconsistencies deal with the placename referred to by any two corresponding instances $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$ in the published indicator data. Instead of ensuring consistency, transversal consistency analysis requires placenames being measured are inconsistent. The values of 'for_city' of m^{city1}_{ij} and m^{city2}_{ik} were compared to ensure that the indicators are measuring performance for different cities which means $G1(m^{city1}_{ij}, m^{city2}_{ik})$ must be true. In addition, m^{city1}_{ij} and m^{city2}_{ik} are inconsistent in terms of place if m^{city1}_{ij} and m^{city2}_{ik} are linked to placename instances with different feature codes or different time intervals.

Trans_G1. Feature Code Inconsistency

Indicators measured for different types of cities may be incomparable due to different administrative level and urbanization of the cities. E.g., the homeless population ratio will be significantly different when comparing a capital city to a farm village. In Geonames (http://www.geonames.org), a set of feature codes were used to distinguish different types of cities and administrative division. For example, feature code 'P.PPLC' represents the capital of a political entity, and 'P.PPLF' represents a farm village. We specify the predicate *Adm(*$city_i$, *$code_i$)* for individual *$city_i$* which is an instances of geo:Feature, and $code_i$ which is an instance of geo:featureCode to represent that *$city_i$* or its administrative division has a feature code $code_i$.

Predicate Adm($city_i$, $code_i$)

∀ $city_i$ admin $code_i$

Type($city_i$,geo:Feature) ∧

(Tri($city_i$,geo:featureCode,code) ∨

(Tri($city_i$,geo:parentCountry,admin) ∨ Tri($city_i$,geo:parentADM1,admin))

∧ Tri(admin,geo:featureCode,code))

⊃ Adm($city_i$, $code_i$)

Defintion 30 Administration (Feature Code)

Any two corresponding instances $m^{city1}_{ij} \in M^{city1}_i$ and $m^{city2}_{ik} \in M^{city2}_i$ are potentially inconsistent if instance of placename $city^{city1}_i$ referred by m^{city1}_{ij} has an admin division feature that is different from the placename $city^{city2}_i$ referred by m^{city2}_{ik}

Trans_G1. Feature Code Inconsistency

∀ m^{city1}_{ij} m^{city2}_{ik} $city^{city1}_i$ $city^{city2}_i$ $code^{city1}_i$ $code^{city2}_i$

Cor_I(m^{city1}_{ij}, m^{city2}_{ik}) ∧

Place(m^{city1}_{ij}, $city^{city1}_i$) ∧ Place(m^{city2}_{ik}, $city^{city2}_i$) ∧

Adm($city^{city1}_i$ $code^{city1}_i$) ∧ Adm($city^{city2}_i$ $code^{city2}_i$) ∧

¬Equal($code^{city1}_i$, $code^{city2}_i$)

⊃ Trans_G1(m^{city1}_{ij}, m^{city2}_{ik})

Defintion 31 Trans_G1. Feature Code Inconsistency

Suppose we have instances of indicators m^{city1}_{ij} and m^{city2}_{ik} where City 1 and City 2 are Toronto and Ottawa respectively. The definition of feature codes for Toronto and Ottawa are listed in Table 8 below. The feature codes are not equal meaning that the cities have different administrative characteristics (Ottawa is the capital of Canada). Therefore city indicators measured for Toronto is potentially transversally inconsistent with city indicator measured for Ottawa.

City	Feature	Label	Definition
Toronto	P.PLA	seat of a first-order administrative division	seat of a first-order administrative division (PPLC takes precedence over PPLA)
Ottawa	P.PLC	capital of a political entity	

Table 8 Feature Code Definition from Geonames

Trans_G2. Transversal Dynamic Place Inconsistency

As discussed in Chapter 3, the definition of a city may change over time. The placename instances measured by city indicators published by both cities must be evaluated to verify if they are related to time intervals that are effective during the time interval of the indicators. For example, if the population of Toronto was drawn from Toronto after the amalgamation in 1998, then for New York City the population must be drawn from New York City that is effective after 1998. Dynamic place inconsistency G3, defined in Chapter 3, cannot be applied to evaluate transversal dynamic place inconsistency since Revision(m^{city1}_{ij}, m^{city2}_{ik}) forces m^{city1}_{ij} and m^{city2}_{ik} to measure the same city. Inconsistency G4 ensures that the indicator is linked to an interval that is during the effective time interval of the city being measured. Recall that indicator value and supporting data published by both City 1 and City 2 are definitional consistent, Thus effective time interval of placename instances measured by m^{city1}_{ij}, m^{city2}_{ik} must include the time interval of the indicators. However, if temporal inconsistency type T2(m^{city1}_{ij}, m^{city2}_{ik}) was detected to be true for instances m^{city1}_{ij} and m^{city2}_{ik} then the placename instances $city^{city1}_{i}$ and $city^{city2}_{i}$ may be linked to time intervals that do not overlap.

Any two instances $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$ are potentially inconsistent in terms of transversal dynamic place inconsistency if the m^{city1}_{ij} and m^{city2}_{ik} are transversal temporally inconsistent according to $T2(m^{city1}_{ij}, m^{city2}_{ik})$ given that both M^{city1}_{i} and M^{city2}_{i} are definitional consistent.

Trans_G2. Transversal Dynamic Place Inconsistency

$$\forall\ m^{city1}_{ij}\ m^{city2}_{ik}$$

$$\text{Cor\_I}(m^{city1}_{ij}, m^{city2}_{ik}) \wedge T2(m^{city1}_{ij}, m^{city2}_{ik})$$

$$\supset \text{Trans\_G2}(m^{city1}_{ij}, m^{city2}_{ik})$$

Defintion 32 Trans_G2. Transversal Dynamic Place Inconsistency

The set of individuals $M^{city1}_{i} \subseteq S^{city1}_{i}$ is placename inconsistent with $M^{city2}_{i} \subseteq S^{city2}_{i}$ if any two instances $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$ satisfy one of the following inconsistency types: Trans_G1.Admin Division inconsistency or Trans_G2. Transversal Dynamic Place inconsistency. In addition, m^{city1}_{ij} and m^{city2}_{ik} must not measure the same city.

Trans_PI. Place Inconsistency

$$\forall\ m^{city1}_{ij}\ m^{city2}_{ik}$$

$$\text{Cor\_I}(m^{city1}_{ij}, m^{city2}_{ik}) \wedge$$

$$(\neg G1(m^{city1}_{ij}, m^{city2}_{ik}) \vee \text{Trans\_G1}(m^{city1}_{ij}, m^{city2}_{ik}) \vee \text{Trans\_G2}(m^{city1}_{ij}, m^{city2}_{ik}))$$

$$\supset \text{Trans\_PI}(m^{city1}_{ij}, m^{city2}_{ik})$$

Defintion 33 Trans_PI. Place Inconsistency

4.1.4 Transversal Measurement Inconsistency

Indicator's supporting data and city specific knowledge published by two cities may disagree on the units of measure used. E.g., Toronto may use 'population cardinality unit' (pc) as the unit of measure of its Homeless population size measure while New York City uses 'kilo-pc' which measures the population size in 1000 times of the unit 'pc'. Consistency of units should be checked for the indicator itself (e.g., 15.2 Homeless population size ratio), population size (a Quantity), and population size values (a Measure) to ensure that the units are consistent throughout indicator value and supporting data published by cities.

In definitional consistency analysis, the unit of measure of the indicator data were evaluated to be definitional consistent with the indicator's definition. For example, 15.2 indicator value published by Toronto links to an instance gci:population_ratio_unit. Supporting data homeless population size was linked to an instance gci:population_cardinality_unit (pc) and city population size was linked to '100 000th of pc'. In the case of Transversal consistency analysis indicators published by both cities were evaluated to be definitional consistent. Therefore both indicator values and supporting data are referring to the same instance gci:population_ratio_unit, pc, or '100 000th of pc'. Thus the unit of measure of quantities or measure are transversally consistent in terms of measurement if indicator value and supporting data from City 1 and City 2 are both definitional consistent.

4.2 Longitudinal Consistency Analysis

Longitudinal consistency analysis evaluates if an indicator is consistent over different time intervals for the same city. For example, is the 15.2 indicator of Toronto published in 2013 consistent with the same indicator published in 2015?

In Longitudinal consistency analysis, we have indicator values and the supporting data used to derive them published by a city at time intervals int_i and int'_i, represented as instances of classes from indicator's definition, theme general knowledge and city specific knowledge. Separate city specific knowledge is provided by the city for int_i and int'_i. Similar to Transversal consistency analysis, we assume ontologies that represent indicator definition, city specific and theme

specific knowledge are logically consistent and published indicator value and supporting data are definitional consistent.

An indicator value and its supporting data, published by a city at interval int, are evaluated against corresponding data published at int' by the same city. Similar to transversal consistency analysis, the correspondence between nodes of S^{int}_i and $S^{int'}_i$ is represented using Cor_I(m^{int}_{ij}, $m^{int'}_{ik}$) where m^{int}_{ij} and $m^{int'}_{ij}$ are primary instances of S^{int}_i and $S^{int'}_i$ respectively. Table 9 below lists the notation used in longitudinal consistency analysis. Inter-indicator correspondence inconsistency (Inter_CI) may exist between the indicator data published by the cities if no correspondence Cor_I(m^{int}_{ij}, $m^{int'}_{ik}$) can be found for nodes m^{int}_{ij} and $m^{int'}_{ik}$.

Published City Indicator Data and City Specific Knowledge

- Let S be the set of all published indicator data
- Let O^{int} be the City Specific ontology used in S at int
- Let $O^{int'}$ be the City Specific ontology used in S at int'
- Let S^{int}_i be the graph that represents the data used to derive indicator i for the city at int
- Let $S^{int'}_i$ be the graph that represents the data used to derive indicator i for the city at int'
- S^{int}_i is composed of a set of attributes A^{int}_i, nodes C^{int}_i and nodes M^{int}_i
  - A^{int}_i is a set of properties in S^{int}_i
  - M^{int}_i is a set of individuals in S^{int}_i
  - $C^{int}_i \subseteq class(O^{int}) \cup indiv(O^{int}) \cup literal(O^{int})$ where
    - $class(O^{int})$ is the set of classes defined in O^{int},
    - $indiv(O^{int})$ is a set of individuals in O^{int} and
    - $literal(O^{int})$ is a set of literals in O^{int}.
- $S^{int'}_i$ is composed of a set of attributes $A^{int'}_i$, nodes $C^{int'}_i$ and nodes $M^{int'}_i$
  - $A^{int'}_i$ is a set of properties in $S^{int'}_i$
  - $M^{int'}_i$ is a set of individuals in $S^{int'}_i$
  - $C^{int'}_i \subseteq class(O^{int'}) \cup indiv(O^{int'}) \cup literal(O^{int'})$ where
    - $class(O^{int'})$ is the set of classes defined in $O^{int'}$, and
    - $indiv(O^{int'})$ is a set of individuals in $O^{int'}$ and
    - $literal(O^{int'})$ is a set of literals in $O^{int'}$.
- $C^{intc}_i \subseteq class(O^{int}) \subseteq C^{int}_i$
- $C^{int'c}_i \subseteq class(O^{int'}) \subseteq C^{int'}_i$

Table 9 Notation for longitudinal consistency analysis

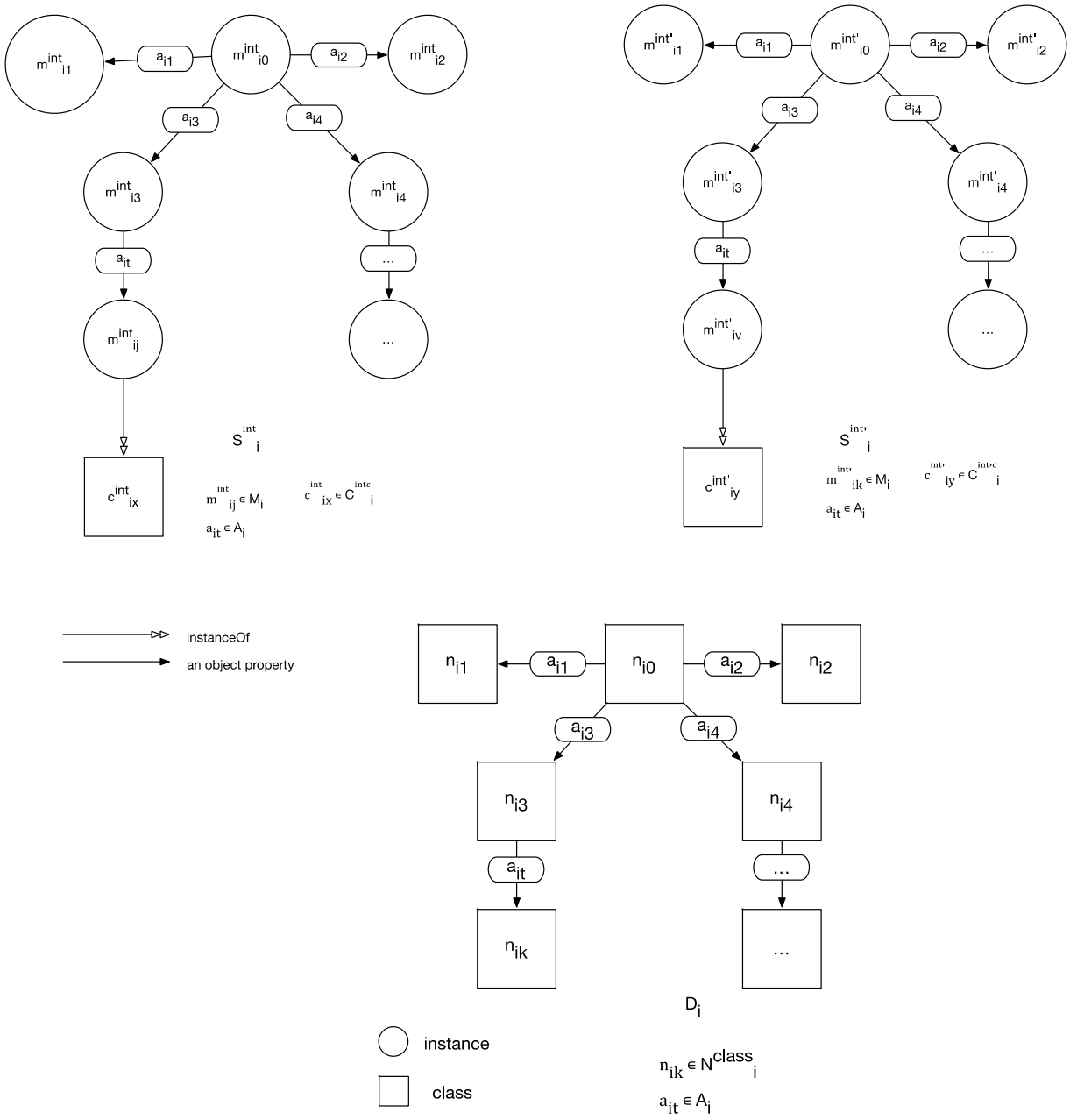


Figure 34 Indicator value and supporting data published by a city at different time comply with definition D_i

In the following sections we define a number of longitudinal inconsistency types. We assume all instances $m^{int}_{ij} \in M^{int}_i$ and $m^{int'}_{ik} \in M^{int'}_i$ are corresponding instances from interval int and int'.

4.2.1 Long_TC. Longitudinal Type Inconsistency

City specific knowledge such as homeless person and homeless shelter may have changed over time. Indicator data may simply be published differently at different intervals. Similar to transversal consistency analysis, the corresponding classes defined in the city specific knowledge for the two time intervals will be evaluated.

Type consistency in Longitudinal consistency analysis is essentially the same evaluation process as in transversal consistency analysis that evaluates m^{int}_{ij} and $m^{int'}_{ik}$

Long_TC. Longitudinal Type Inconsistency

$\forall m^{int}_{ij}\ m^{int'}_{ik}$

Trans_TC(m^{int}_{ij}, $m^{int'}_{ik}$)

$\supset$ Long_TC(m^{int}_{ij}, $m^{int'}_{ik}$)

Defintion 34 Long_TC. Longitudinal Type Inconsistency

For example, suppose Toronto has changed its definition of homeless shelters by replacing treatment facilities with family shelters which was not part of the definition in 2013. In this case the 15.2 indicator of Toronto published in 2015 is inconsistent with its previous version published in 2013 and Long_TC(m^{int}_{ij}, $m^{int'}_{ik}$).

4.2.2 Long_TI. Longitudinal Temporal Inconsistency

Longitudinal consistency evaluation compares temporal concepts of an indicator published in different years. In particular, it compares the temporal units(T4) of the two intervals and the duration.

Long_T1. Duration Inconsistency

Duration inconsistency arises when the durations of the two intervals differ. Instances m^{int}_{ij} and $m^{int'}_{ik}$ are inconsistent if their time intervals int and int' have different duration. We specify the

predicate Dur(int, intDur) to represent that interval int has a duration intDur which is an instance of ot:DurationDescription.

Long_T1. Duration Inconsistency

$\forall\ m^{int}_{ij}\ m^{int'}_{ik}\ int_i\ int'_i\ intDur_i\ intDur'_i$

$Cor\_I(m^{int}_{ij}, m^{int'}_{ik}) \wedge$

$Type(int_i, ot:Interval) \wedge Type(int'_i, ot:Interval) \wedge$

$Time(m^{int}_{ij}, int_i) \wedge Time(m^{int'}_{ik}, int'_i) \wedge$

$Dur(int_i, intDur_i) \wedge Dur(int'_I, intDur'_i) \wedge \neg Equal(intDur_i, intDur'_i)$

$\supset Long\_T1(m^{int}_{ij}, m^{int'}_{ik})$

Defintion 35 Long_T1. Duration Inconsistency

Definitional temporal inconsistency evaluates if the supporting data refer to the same time intervals as the indicator. Longitudinal duration inconsistency evaluates if two indicators are measured for time intervals with different durations. For example, Toronto's 15.2 indicator would be longitudinally inconsistent if the indicator was measured semi-annually in 2013 but annually in 2015. In this case $Long\_T1(m^{int}_{ij}, m^{int'}_{ik})$ is true where m^{int}_{ij} and $m^{int'}_{ik}$ are indicator values published by Toronto for 2013 and 2015 respectively.

The set of individuals $M^{int}_i \subseteq S^{int}_i$ and $M^{int'}_i \subseteq S^{int'}_i$ is temporal inconsistent if any two corresponding instances $m^{int}_{ij} \in M^{int}_i$ and $m^{int}_{ik} \in M^{int'}_i$ are inconsistent in terms of one of the following inconsistency types: Long_T1.Duration Inconsistency or T4.Temporal granularity inconsistency

Long_TI. Longitudinal Temporal Inconsistency

$\forall m^{int}_{ij}\ m^{int'}_{ik}$

$\quad\quad$ Long_T1(m^{int}_{ij}, $m^{int'}_{ik}$) ∨ T4(m^{int}_{ij}, $m^{int'}_{ik}$)

⊃ Long_TI(m^{int}_{ij}, $m^{int'}_{ik}$)

Defintion 36 Long_TI. Longitudinal Temporal Inconsistency

4.2.3 Long_PI. Longitudinal Place Inconsistency

An indicator is longitudinal place inconsistent if the placename referred at different time intervals differ. In addition, cities may have changed over time. Any changes in the geographical location associated to the indicator or population that is part of the indicator data would make indicators inconsistent over time. Examples of geographical changes include city boundary changes or relocation of a city.

Long_G1. Longitudinal Geometry Inconsistency

The geometrical shape of a city's boundary may change overtime, e.g., due to amalgamation. A city's geometry can be represented as instances of geom:Geometry class linked to a place instance via the property geom:geometry (Norton, et al., 2012).

Corresponding instances $m^{int}_{ij} \in M^{int}_{i}$ and $m^{int'}_{ik} \in M^{int'}_{i}$ are inconsistent due to geometry inconsistency if the place instances $city^{int}$ and $city^{int'}$ are linked to different instances of geom:Geometry. We use predicate Geo($city^{int}$, geo^{int}) to represent that $city^{int}$ has a geometry geo^{int} which is an instance of geom:Geometry.

Long_G1. Longitudinal Geometry Inconsistency

$\forall\ m^{int}_{ij}\ m^{int'}_{ik}\ int_i\ int'_i\ city^{int}\ city^{int'}$

$Cor\_I(m^{int}_{ij}, m^{int'}_{ik}) \wedge$

$Type(int_i, ot:Interval) \wedge Type(int'_i, ot:Interval) \wedge$

$Type(city^{int}, sc:City) \wedge Type(city^{int'}, sc:City) \wedge$

$Type(geo^{int}, geom:Geometry) \wedge Type(geo^{in't}, geom:Geometry) \wedge$

$Time(m^{int}_{ij}, int_i) \wedge Time(m^{int'}_{ik}, int'_i) \wedge Place(m^{int}_{ij}, city^{int}) \wedge Place(m^{int'}_{ik}, city^{int'}) \wedge$

$Geo(city^{int}, geo^{int}) \wedge Geo(city^{int'}, geo^{int'}) \wedge \neg Equal(geo^{int}, geo^{int'})$

$\supset Long\_G1(m^{int}_{ij}, m^{int'}_{ik})$

Defintion 37 Long_G1. Longitudinal Geometry Inconsistency

Long_G2. Longitudinal Coordinates Inconsistency

The geographical coordinates (i.e., longitude and latitude) may also change for a city over time. Natural catastrophes or wars may result the relocation of a city or political entity. Geographical coordinates can be represented using properties from Basic Geo (WGS84 lat/long) Vocabulary (Brickley, 2006), i.e., wgs84[23]:long and wgs84:lat. Changes in a reference point, which is an instance of class wgs84:point, defined by geographical coordinates (e.g., the center) within the geometrical shape imply geographical inconsistency between indicators being compared in terms of geographical coordinates.

Corresponding instances $m^{int}_{ij} \in M^{int}_i$ and $m^{int'}_{ik} \in M^{int'}_i$ are inconsistent due to coordinate inconsistency if the place instances $city^{int}$ and $city^{int'}$ are linked to different reference points p^{int}

[23] http://www.w3.org/2003/01/geo/wgs84_pos#

and $p^{int'}$ within the city boundary. We use predicate Poi($city^{int}$, p^{int}) to represent that $city^{int}$ has a reference point p^{int} within its boundary.

Long_G2. Longitudinal Coordinates Inconsistency

$\forall\ m^{int}_{ij}\ m^{int'}_{ik}\ int_i\ int'_i\ city^{int}\ city^{int'}$

Cor_I(m^{int}_{ij}, $m^{int'}_{ik}$) ∧

Type(int_i,ot:Interval) ∧ Type(int'_i,ot:Interval)∧

Type($city^{int}$,sc:City) ∧ Type($city^{int'}$,sc:City)∧

Type(p^{int},wgs84:point) ∧ Type($p^{in't}$,wgs84:point)∧

Time(m^{int}_{ij},int_i) ∧ Time($m^{int'}_{ik}$,int'_i) ∧ *Place(m^{int}_{ij}, $city^{int}$)* ∧ *Place($m^{int'}_{ik}$, $city^{int'}$)* ∧

Poi($city^{int}$, p^{int}) ∧ Poi($city^{int'}$, $p^{int'}$) ∧ ¬Equal(p^{int}, $p^{int'}$)

⊃ Long_G2(m^{int}_{ij}, $m^{int'}_{ik}$)

Defintion 38 Long_G2. Longitudinal Coordinates Inconsistency

Other types of longitudinal place inconsistency include feature code inconsistency (Trans_G1) and dynamic place inconsistency (G3) which were both described previously. In section 4.1.3 we defined feature code inconsistency which evaluates the feature code associated with the city or its administrative divisions. Inconsistency Trans_G1 defined previously can be applied to m^{int}_{ij} and $m^{int'}_{ik}$ to detect changes in the characteristic of the city or its administrative division over time. Placename instances $city^{int}$ and $city^{int'}$ linked to m^{int}_{ij} and $m^{int'}_{ik}$ respectively are also linked to different time intervals. E.g Toronto [1967-1998] and Toronto[1998-] are considered as different placename instances. Recall that inconsistency G3(m^{int}_{ij}, $m^{int'}_{ik}$) is true if $city^{int}$ is a revision of $city^{int'}$, i.e., same city linked to different time intervals, which is intended in longitudinal consistency analysis. Thus the negation of G3, i.e., ¬G3(m^{int}_{ij}, $m^{int'}_{ik}$) implies

longitudinal place inconsistency between m^{int}_{ij} and $m^{int'}_{ik}$ since $city^{int}$ and $city^{int'}$ are not revisions of the same city.

Long_PI. Longitudinal Place Inconsistency

$$\forall\, m^{int}_{ij}\, m^{int'}_{ik}$$

$$\text{Long\_G1}(m^{int}_{ij}, m^{int'}_{ik}) \vee \text{Long\_G2}(m^{int}_{ij}, m^{int'}_{ik}) \vee$$

$$\text{Trans\_G1}(m^{int}_{ij}, m^{int'}_{ik}) \vee \neg \text{G3}(m^{int}_{ij}, m^{int'}_{ik})$$

$$\supset \text{Long\_PI}(m^{int}_{ij}, m^{int'}_{ik})$$

Defintion 39 Long_PI. Longitudinal Place Inconsistency

The set of individuals $M^{int}_{i} \subseteq S^{int}_{i}$ is longitudinal placename inconsistent with $M^{int'}_{i} \subseteq S^{int'}_{i}$ if any two instances $m^{int}_{ij} \in M^{int}_{i}$ and $m^{int'}_{ik} \in M^{int'}_{i}$ satisfy inconsistency types Long_G1 (Longitudinal Geometry Inconsistency), Long_G2 (Longitudinal Coordinates Inconsistency), Trans_G1 (Feature Code Inconsistency), or do not satisfy G3 (Dynamic Place Inconsistency).

4.2.4 Longitudinal Measurement Inconsistency

A city should use the same unit of measure for indicator data published during different time periods. E.g., the unit of measure for population size should remain as ‘pc’ for indicator instances published in different years. Similar to transversal measurement inconsistency, if published indicator value and supporting data were evaluated to be definitional consistent, then the unit of measures are guaranteed to be consistent.

4.3 Summary

In this chapter we defined types of transversal and longitudinal inconsistencies of published city indicators. Transversal consistency analysis evaluates city indicators published by different cities at the same time interval, and longitudinal inconsistency evaluates city indicators published by the same city at different times. The inconsistency types defined in this chapter are summarized as follow. Note that only inconsistency types defined specifically for transversal and

longitudinally are shown here. Prolog implementation of each inconsistency types can be found in Appendix III.

Correspondence Inconsistency: No correspondence have been detected between Primary nodes in the sets of city published indicator value and supporting data.

Inconsistency	**Description**
Inter_CI. Correspondence Inconsistency	For any corresponding nodes $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$, or $m^{int}_{ij} \in M^{int}_{i}$ and $m^{int'}_{ik} \in M^{int'}_{i}$ in the case of longitudinal consistency analysis, there exists a class m^{city2}_{iy} or $m^{int'}_{iy}$ that is linked to m^{city2}_{ik} or $m^{int'}_{ik}$ via property a_{it} where there is no instance m^{city1}_{ix} or m^{int}_{ix} linked to m^{city1}_{ij} or m^{int}_{ij} that corresponds to m^{city2}_{iy} or $m^{int'}_{iy}$.

Transversal and Longitudinal Type Inconsistency: Instances of published indicator data are type inconsistent if the instances are of types of classes that are not equivalent.

Type Inconsistency	**Description**
Trans_TC. Transversal Instance Type Inconsistency	An instance $m^{city1}_{ij} \in M^{city1}_{i}$ is type inconsistent with its corresponding instance $m^{city2}_{ik} \in M^{city2}_{i}$ if m^{city1}_{ij} is an instance of a class $c^{city1}_{i} \in C^{city1}_{i}$ and m^{city2}_{ik} is an instance of a class $c^{city2}_{i} \in C^{city2}_{i}$ such that c^{city1}_{i} is type inconsistent with c^{city2}_{i}.
Long_TC. Longitudinal Type Inconsistency	Type consistency in Longitudinal consistency analysis is the same evaluation process as in transversal consistency analysis that evaluates m^{int}_{ij} and $m^{int'}_{ik}$

Transversal and Longitudinal Temporal Inconsistency: Temporal instances of the two indicators represent the same time interval. E.g., temporal instances have same temporal unit. The values for year, month, etc. must be the same in transversal consistency analysis. Temporal inconsistency is evaluated to ensure that time intervals are not equal using inconsistency type T2 defined in definitional consistency analysis for Transversal consistency analysis. For longitudinal consistency analysis, duration inconsistency needs to be evaluated.

Temporal Inconsistency	**Description**
Long_T1. Duration Inconsistency	An instance m^{int}_{ij} and $m^{int'}_{ik}$ are inconsistent if their time intervals int and int' have different duration. We specify the predicate Dur(int, intDur) to represent that interval int has a duration intDur which is an instance of ot:DurationDescription.

Transversal and Longitudinal Placename Inconsistency: Placename concepts are representing the same type geographical area, e.g., city, district, slum, etc. Transversal place inconsistencies include feature code inconsistency where the cities have different administrative features, or transversal dynamic placename inconsistency which considers the case where cities measured by the indicators were linked to different time intervals. Longituidinal place inconsistencies deal with Longitudinal geometry inconsistency and coordinates inconsistency which deals with the city boundary geotery and geographical coordinates respectively.

Placename Inconsistency	**Description**
Trans_G1. Feature Code Inconsistency	Any two instances $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$ are potential inconsistent if instance of placename $city^{city1}_{i}$ referred by m^{city1}_{ij} has an admin division feature that is different from the placename $city^{city2}_{i}$ referred by m^{city2}_{ik}
Trans_G2. Transversal Dynamic Place Inconsistency	Any two instances $m^{city1}_{ij} \in M^{city1}_{i}$ and $m^{city2}_{ik} \in M^{city2}_{i}$ are inconsistent if the Placename $city^{city1}_{i}$ referred by m^{city1}_{ij} is related to a different time interval as the placename $city^{city2}_{i}$ referred by m^{city2}_{ik}. Given that both M^{city1}_{i} and M^{city2}_{i} are defitionally consistent with indicator's definition D_i, m^{city1}_{ij} and m^{city2}_{ik} are inconsistent according to $T2(m^{city1}_{ij}, m^{city2}_{ik})$.

Long_G1. Longitudinal Geometry Inconsistency	Corresponding instances $m^{int}_{ij} \in M^{int}_{i}$ and $m^{int'}_{ik} \in M^{int'}_{i}$ are inconsistent due to geometry inconsistency if the place instances $city^{int}$ and $city^{int'}$ are linked to different instances of geom:Geometry.
Long_G2. Longitudinal Coordinates Inconsistency	Corresponding instances $m^{int}_{ij} \in M^{int}_{i}$ and $m^{int'}_{ik} \in M^{int'}_{i}$ are inconsistent due to coordinate inconsistency if the place instances $city^{int}$ and $city^{int'}$ are linked to different reference points p^{int} and $p^{int'}$ within the city boundary.

Transversal and Longitudinal Measurement Inconsistency: Unit of measure used by both indicators must be the same. For both transversal and longitudinal analysis, since both sets of indicator value and supporting data are definitional consistent, thus unit of measures must be equal if they are transversal or longitudinal type consistent.

Chapter 5
Implementation and Example

5 Implementation and Example

In this chapter we describe the implementation of City Indicator Consistency Checker (CICC). We then describe an example for each of the definitional, transversal and longitudinal inconsistency types using the CICC. For our example, we use city indicator data and definitions for ISO 37120 15.2 homeless population ratio published by Toronto for 2013, 2015 and New York City for 2013. We also use fictional city specific ontologies and the GCI-Shelter theme ontology from the PolisGnosis project.

5.1 City Indicator Consistency Checker

CICC is implemented in SWI-Prolog (Wielemaker, Schrijvers, Triska, & Lager, 2012) version 7.2.2[24]. The Figure 35 below shows the architecture of CICC. It performs definitional, transversal or longitudinal consistency analysis based on user inputs through a web interface. The web interface allows users to uploaded their supporting data (instance file) in TTL (turtle) format, and their city specific ontology in OWL/RDF format. A second set of input files are required if transversal/longitudinal consistency analysis is to be performed. Users will then select the types of consistency analysis to be performed. The resulting analysis will then be returned to the user.

[24] http://www.swi-prolog.org/versions.txt

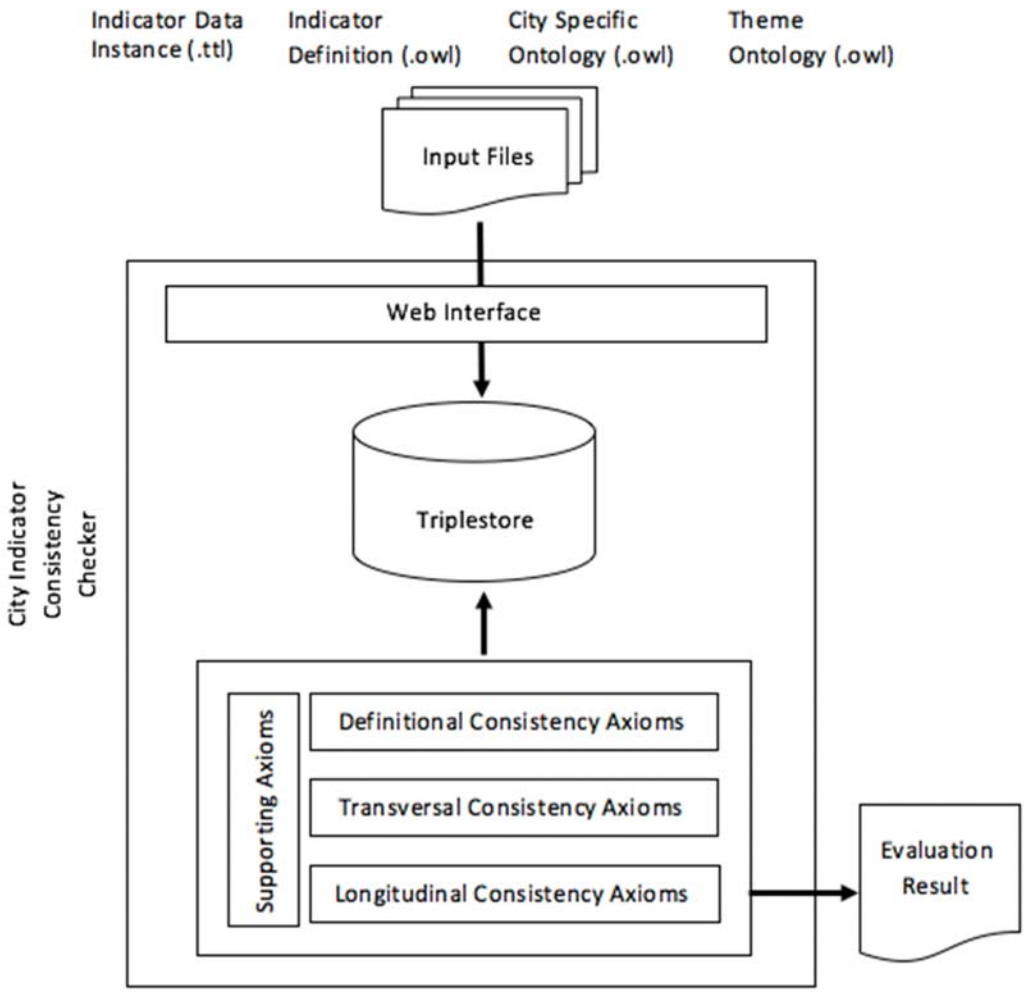


Figure 35 CICC Architecture

The CICC performs its analysis on following user input files:

- Indicator definition: The file contains the definition of an indicator represented in OWL/RDF, etc. The indicator definition should consist of classes and properties that represent an indicator's IRI, supporting data and relationships between them. E.g., iso37120:15.2 has numerator isos:15.2_homeless_population_size. The definition of 15.2 indicator is shown in Figure 9.
- Theme Knowledge: This file contains knowledge of theme specific knowledge represented using ontologies. E.g., GCI Shelters ontology, which contains classes that

represent the shelter themed knowledge used by 15.2 indicator's definition such as gcis:Homeless_person and gcis:Homeless_shelter, etc.

- City instance file: This file should contain instances of the indicator being evaluated and its supporting data. E.g., 15.2_trt_2013, trt_homeless_pop_size_2013, etc.
- City definition file: This file should contain city specific knowledge of the city being evaluated, which contains classes such as Toronto homeless person and axioms that define the class.
- Second instance file: For transversal consistency analysis, the indicator instances should be published by a different city (city 2) during the same year, whereas for longitudinal consistency, the instances should be published by the same city during a different year (time 2). This section should be left as blank in case of Definitional consistency analysis.
- Second definition file: This files contains city specific definition of city 2 for transversal and time 2 for longitudinal consistency analysis. This section should be left as blank in case of Definitional consistency analysis.

CICC imports and stores the information in a triplestore. All indicator values, supporting data and definitions represented using ontologies such as the GCIO are stored as RDF triples. This means that Prolog axioms can reason about instances, classes and properties as input data. For example, the triple (trt_homeless_pop, gci:located_in, geo:Toronto) can be queried using Prolog predicate rdf(trt_homeless_pop, gci:located_in, geo:Toronto). User interface of the CICC are shown below. Instance files in TTL format and theme and city specific ontologies in OWL/RDF are loaded into Prolog using rdf_load from Prolog's Semantic Web library. Prefixes of each ontology were registered using rdf_register_prefix. Inconsistency types described in Chapter 3 and 4 are evaluated using a set of Prolog predicates described in Appendix II. The resulting output is displayed in text format. Inconsistent instances and corresponding classes are indicated with the types of inconsistencies detected. Figure 37 below shows an example of result output text.

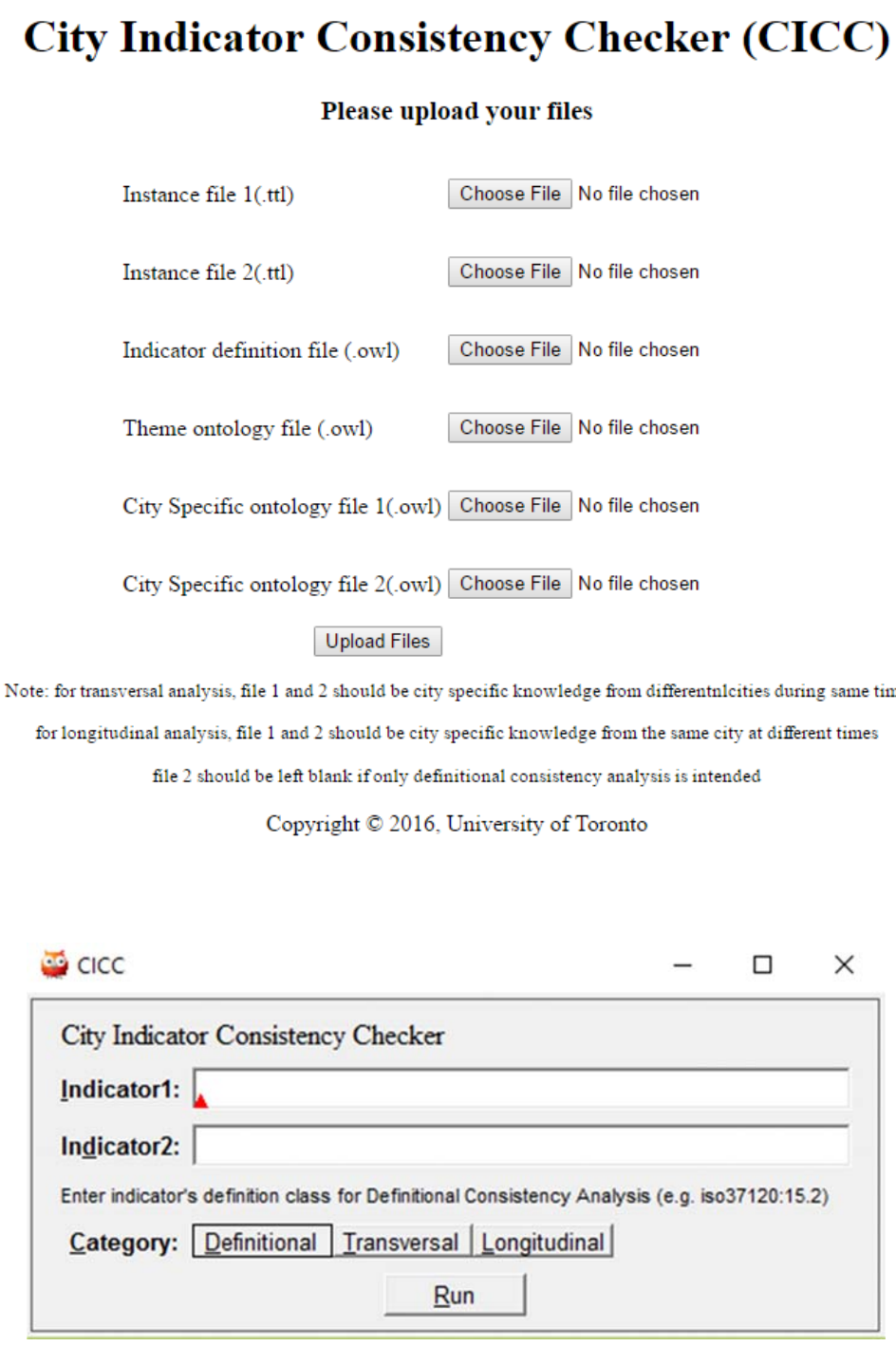
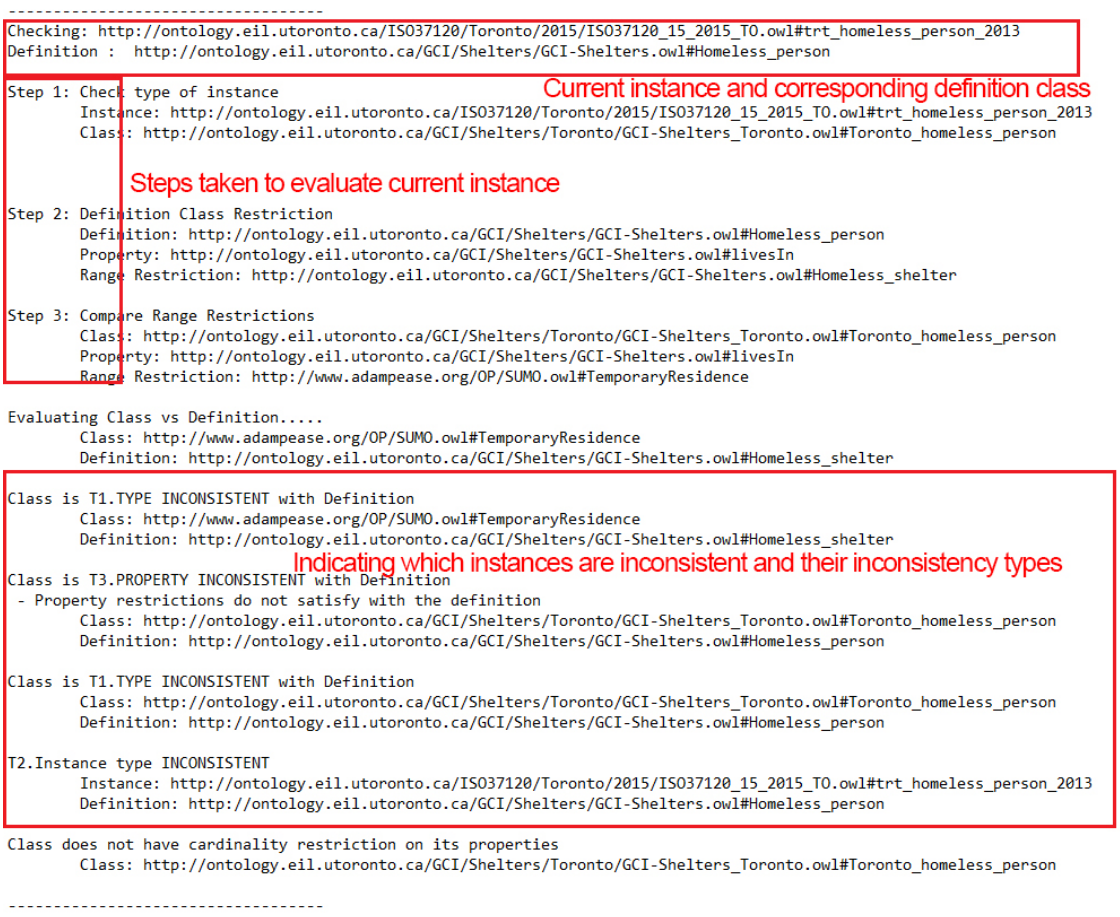


Figure 36 CICC User Interface




```
----------------------------------
Checking: http://ontology.eil.utoronto.ca/ISO37120/Toronto/2015/ISO37120_15_2015_TO.owl#trt_homeless_person_2013
Definition :  http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

Step 1: Check type of instance
        Instance: http://ontology.eil.utoronto.ca/ISO37120/Toronto/2015/ISO37120_15_2015_TO.owl#trt_homeless_person_2013
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person

Step 2: Definition Class Restriction
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person
        Property: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#livesIn
        Range Restriction: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_shelter

Step 3: Compare Range Restrictions
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person
        Property: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#livesIn
        Range Restriction: http://www.adampease.org/OP/SUMO.owl#TemporaryResidence

Evaluating Class vs Definition.....
        Class: http://www.adampease.org/OP/SUMO.owl#TemporaryResidence
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_shelter

Class is T1.TYPE INCONSISTENT with Definition
        Class: http://www.adampease.org/OP/SUMO.owl#TemporaryResidence
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_shelter
Class is T3.PROPERTY INCONSISTENT with Definition
 - Property restrictions do not satisfy with the definition
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

Class is T1.TYPE INCONSISTENT with Definition
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

T2.Instance type INCONSISTENT
        Instance: http://ontology.eil.utoronto.ca/ISO37120/Toronto/2015/ISO37120_15_2015_TO.owl#trt_homeless_person_2013
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

Class does not have cardinality restriction on its properties
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person

----------------------------------
```

Figure 37 CICC Output Text

Chapters 3 and 4 provide examples for each type of inconsistency. In this section we demonstrate definitional, transversal and longitudinal consistency analysis on complete sets of indicator instances coupled with their city specific ontologies. We reuse the indicator definitions and theme knowledge developed by the PolisGnosis project. Specifically, we import the ISO 37120 city indicator definitions for the education, shelter and innovation themes. These in turn, use the GCI Foundation ontology and GCI theme ontologies such as GCI-Education, GCI-Shelter, and GCI-Innovation. Fictional data and ontologies, including city specific knowledge for Toronto and New York City, were created to demonstrate the consistency analysis process[25]. Appendix I

[25] Published indicators for the City of Toronto do not contain sufficient detail to be used to evaluate the CICC. See Fox & Pettit (2015) for more details.

lists OWL files for all indicators' definitions and ontologies mentioned as well as the Prolog axioms. The prefixes registered are shown in Table 10 below.

Prefix	**Full URI**
gci	'http://ontology.eil.utoronto.ca/GCI/Foundation/GCI-Foundation.owl#'
gcis	'http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#'
isos	'http://ontology.eil.utoronto.ca/GCI/ISO37120/Shelters.owl#'
iso37120	'http://ontology.eil.utoronto.ca/ISO37120.owl#'
ot	'http://www.w3.org/2006/time#'
geo	'http://www.geonames.org/ontology/ontology_v3.1.rdf#'
gd	'http://www.linkedgeodata.org/ontology/'
sc	'http://schema.org/'
kp	'http://ontology.eil.utoronto.ca/trust.owl#'
om	'http://www.wurvoc.org/vocabularies/om-1.8/'
gs	'http://ontology.eil.utoronto.ca/govstat.owl#'
pr	'http://www.w3.org/ns/prov#'
tr	'http://ontology.eil.utoronto.ca/trust.owl#'
gn	'http://sws.geonames.org/'
sumo	'http://www.adampease.org/OP/SUMO.owl#'
org	'http://ontology.eil.utoronto.ca/organization.owl#'
ic	'http://ontology.eil.utoronto.ca/icontact.owl#'

gcis_trt	'http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#'
gcis_nyc	'http://ontology.eil.utoronto.ca/GCI/Shelters/NewYork/GCI-Shelters_NewYork.owl#'
t2013	'http://ontology.eil.utoronto.ca/ISO37120/Toronto/2013/ISO37120_15_2013_TO.owl#'
n2013	'http://ontology.eil.utoronto.ca/ISO37120/NYC/2013/ISO37120_15_2013_NY.owl#'
t2015	'http://ontology.eil.utoronto.ca/ISO37120/Toronto/2015/ISO37120_15_2015_TO.owl#'

Table 10 Prefix Registration

Toronto has published a set of ISO 37120 indicator values for 2013 including 15.2 homeless population ratio (City of Toronto, 2014) in PDF format. As shown in Figure 38 below, in 2013 the 15.2 indicator value was 190 per 100,000 population with a homeless population size value of 5,253. These values were used to create instances in TTL format according to 15.2 indicator definition and GCI Shelter ontology. Since Toronto does not have a city specific ontology, a fictional city specific ontology GCI Shelter-Toronto[26] was created for the purpose of evaluating the CICC. Classes Toronto_homeless_person and Toronto_homeless_shelter were created based on descriptions from Toronto's "2013 Street Needs Assessment Results" (City of Toronto, 2013). Similarly, 15.2 indicator data was created based on the document "New York City Homeless Municipal Shelter Population, 1983-Present" (Coalition for the Homeless, 2016). A city specific ontology GCI Shelter-NYC[27] was also created. 15.2 indicator value and city specific ontology for Toronto 2015 was created with arbitrary values and a modified version of GCI Shelter-Toronto. All files are listed in Appendix I.

[26] Prefix: gcis-trt

[27] Prefix: gcis-nyc

TORONTO Toronto's Results for Global City Indicators Under ISO 37120

Theme/Indicator Under ISO 37120	Indicator Result/ Rate	Absolute Value	Year of Data	Comments
14.5 - Violent crime rate per 100,000 population (supporting indicator)	1,016 violent crime incidents (per 100,000 population)	28,162 violent crime incidents (in total)	2013	Based on reported crime
Section 15 – Shelter				
15.1 - Percentage of city population living in slums (core indicator)	1.95% of population living in overcrowded or illegal housing	54,000 estimated population living in overcrowded or illegal housing (in total)	2013	• Overcrowding as defined by Canada's National Occupancy Standard, are households with more than 2 persons per bedroom and is estimated at 50,000 people for Toronto. • Those with unsecured tenure/in illegal rooming houses, is estimated at 4,000 people • The other criteria in the ISO definition for slums of a lack of durable housing or no access to water or sanitation, are not present in Toronto.
15.2 - Number of homeless per 100,000 population (supporting indicator)	190 homeless people (per 100,000 population)	5,253 homeless people (in total)	2013	Based on 2013 Street Needs Assessment Survey
15.3 - Percentage of households that exist without registered legal titles (supporting indicator)	0.37% of households without registered title	4,100 estimated households without registered title (in total)	2012	Estimated households in illegal rooming houses

Figure 38 ISO Indicator Data for Toronto 2013, adapted from City of Toronto (2014)

5.2 Definitional Inconsistency Example

We will use the ISO 37120 Shelter theme indicator 15.2 'Homeless population ratio' indicator published by Toronto for the year 2013 as our example. The Figure 39 below depicts the structure of the 15.2 indicator instance and its supporting data following the ISO 37120 definition described in Chapter 2. In order to represent this indicator, '15.2_trt_2013' was created as an instance of the class iso37120:'15.2'. It is linked to the individual geo:Toronto which is an instance of sc:'City' via gci:'for_city'. An instance of ot:'Interval', 'y2013' which represents the year 2013 is linked to '15.2_trt_2013' via gci:'for_time_interval'. The actual measurement of the indicator, '15.2_trt_2013_value', was created as an instance of om:Measure class. The instance of om:Measure consists of the measurement's numeric value and its unit of

measure which is the instance gci:’population_cardinality_unit’ (pc). The indicator’s supporting data is represented as follows: ’15.2_trt_2013’ has a numerator ‘trt_homeless_pop_size_2013’ which is an instance of isos:’15.2_homeless_population_size’ that represents the homeless population size of Toronto for 2013. It is a cardinality of (gci:‘cardinality_of’) ‘trt_homeless_pop_2013’ which is the homeless population in Toronto for 2013. ‘trt_homeless_pop’ has a property gci:’located_in’ that points to the instance geo:’Toronto’ and it is defined by (gci:’defined_by’) ‘trt_homeless_person_2013’ which is an instance of isos:’15.2_Homeless_person’.

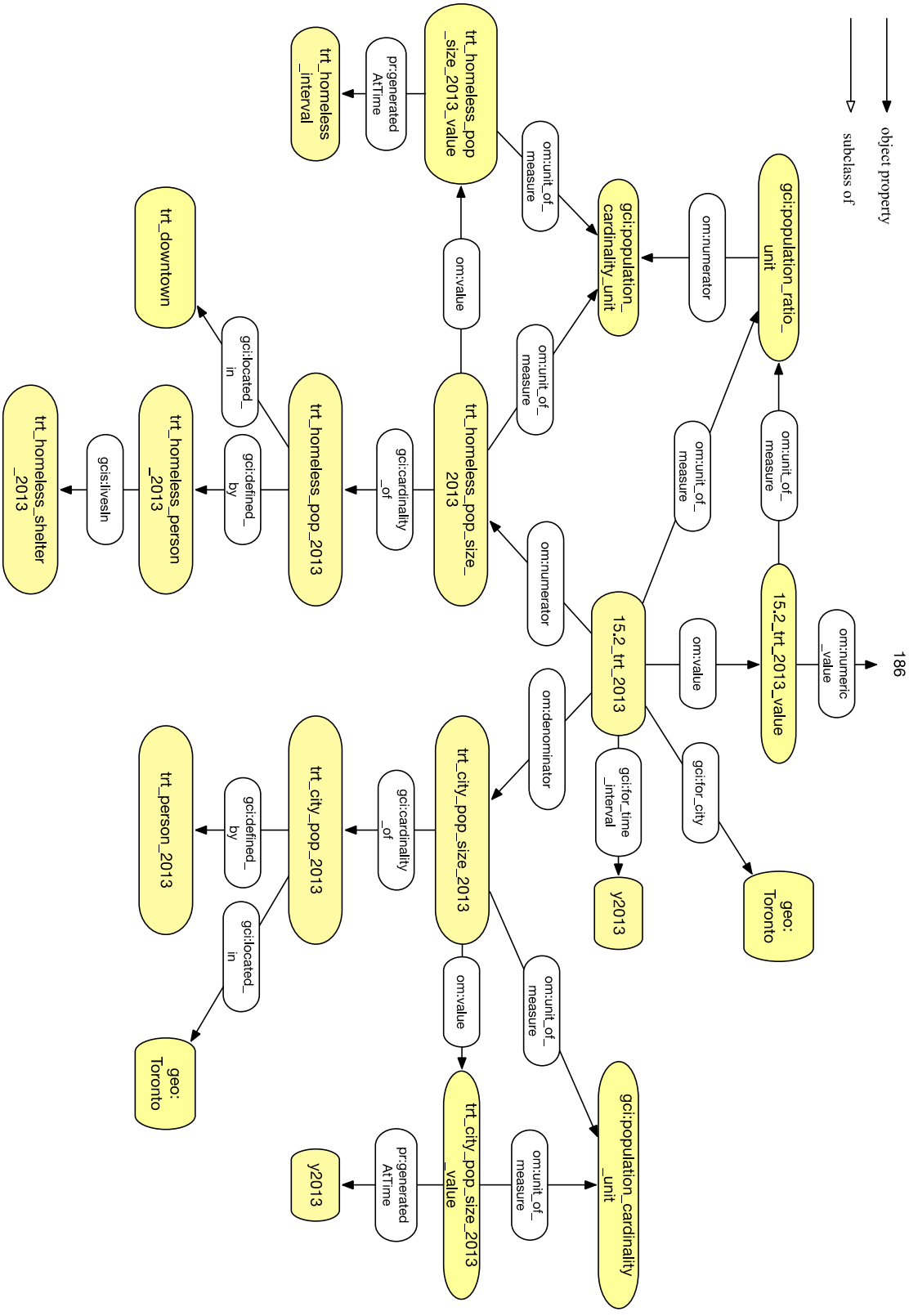


Figure 39 Published 15.2 indicator value and supporting data for Toronto 2013

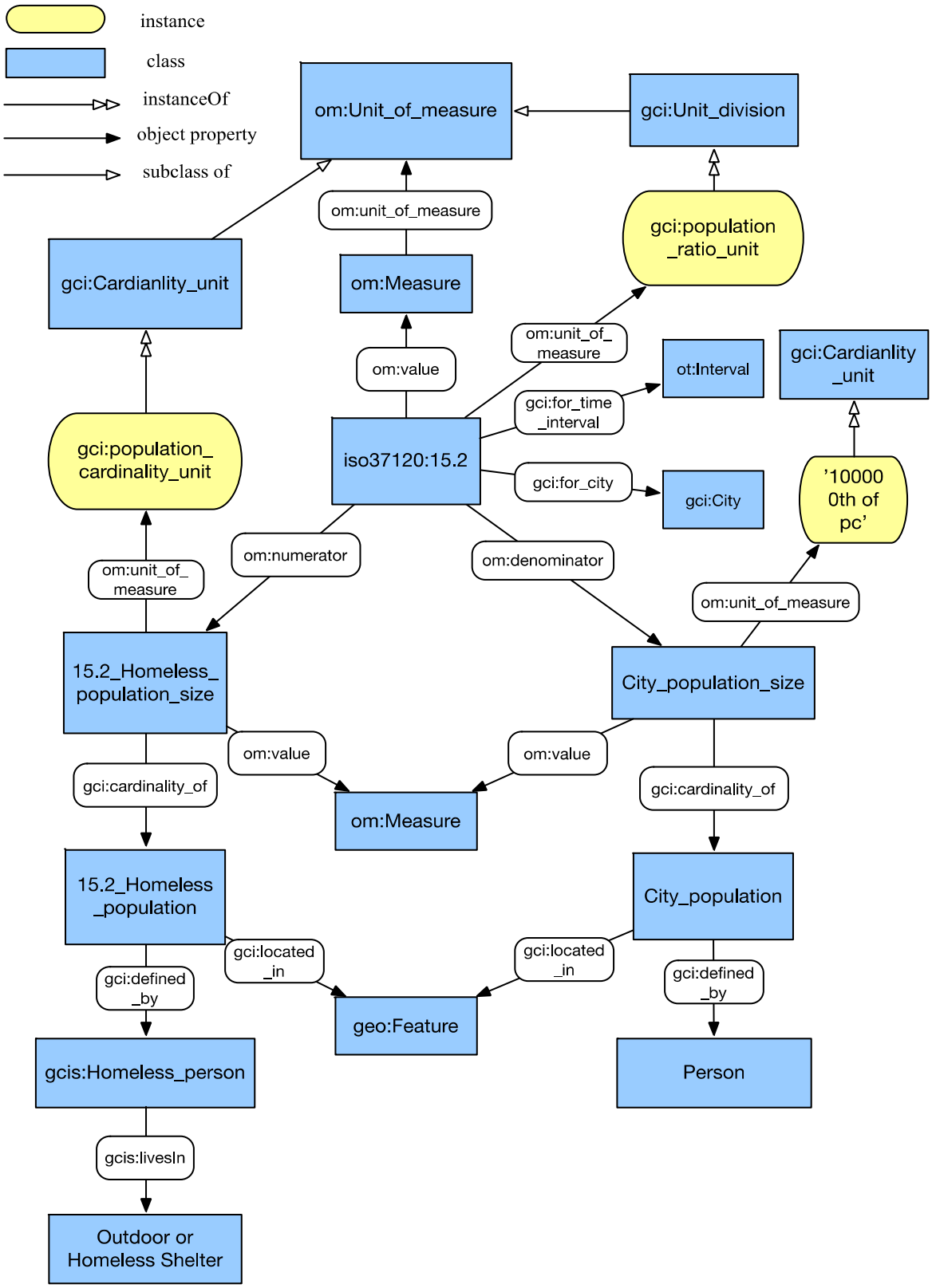


Figure 40 ISO 37120 15.2 Indicator and Definition

For the purpose of demonstration, the following inconsistencies have been embedded in the example indicator data.

Inconsistency	**Instance or Class**	**Description**
TC2. Type Inconsistency	y2013	y2013 should be an instance of ot:Interval. Instead, it is an instance of ot:DateTimeDescription
TC1. Class Type Inconsistency	gci-trt:Toronto_homeless_shelter	Type inconsistent with gcis:Homeless_shelter class
TC1. Class Type Inconsistency	gci-trt:Toronto_homeless_person	Type inconsistent with gcis:Homeless_person class since gci-trt:Toronto_homeless_shelter is type inconsistent with gcis:Homeless_shelter class
TC2. Instance Type Inconsistency	trt_homeless_person_2013	gci-trt:Toronto_homeless_person is type inconsistent with gcis:Homeless_person class
T2. Interval Equality Inconsistency	trt_homeless_pop_size_2013_value	This value was generated at an interval that covers only part of 2013
G2. Subplace Inconsistency	trt_homeless_pop_2013	The homeless population was drawn from (gci:located_in) some areas in Toronto
M2. Indicator Component	City_pop_size_2013	The city population size is measured in ‘pc’ instead of 100 000th of pc. M3 also applies.

Measurement Inconsistency		

Table 11 List of definitional inconsistencies in example

Users will first enter the named of an instance and a definition class in the text field shown in Figure 36 which act as a starting instance and its corresponding definition class. In the case of our example, the inputs are instance 15.2_trt_2013 and class iso37120:15.2. Correspondence between supporting data instances and classes from definitions are detected using method defined in chapter 3. Next we outline the evaluation process of inconsistencies listed above performed by the CICC for definitional consistency analysis[28]. Transversal and Longitudinal consistency analysis examples are provided in section 5.3 and 5.4.

Step 1. Instance: 15.2_trt_2013

Correspondence Identification

Corresponding definition class: iso37120:15.2

Type Inconsistency Evaluation

Result: TC2. Type inconsistent

Description: the following inconsistencies have been detected in supporting data as shown in Figure 41 below. Thus the 15.2_trt_2013 is type inconsistent. Details about inconsistency evaluation for following instances are introduced in steps 2, 7, and 8.

- TC2(y2013, ot:Interval)
- TC1(gci-trt:Toronto_homeless_shelter, gcis:Homeless_shelter)
- TC1(gci-trt:Toronto_homeless_person, gcis:Homeless_person)
- TC2(trt_homeless_person_2013, gcis:Homeless_person)

[28] For expository purpose, the steps outlined in this chapter differ from actual sequences of evaluation from the CICC. Please refer to Appendix II for Prolog axioms used in CICC.

```
Class is T1.TYPE INCONSISTENT with Definition
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_shelter
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_shelter

Class is T3.PROPERTY INCONSISTENT with Definition
 - Property restrictions do not satisfy with the definition
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

Class is T1.TYPE INCONSISTENT with Definition
        Class: http://ontology.eil.utoronto.ca/GCI/Shelters/Toronto/GCI-Shelters_Toronto.owl#Toronto_homeless_person
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person

T2.Instance type INCONSISTENT
        Instance: http://ontology.eil.utoronto.ca/ISO37120/Toronto/2015/ISO37120_15_2015_TO.owl#trt_homeless_person_2013
        Definition: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl#Homeless_person
```

Figure 41 CICC output for TC2(trt_homeless_person_2013, gcis:Homeless_person)

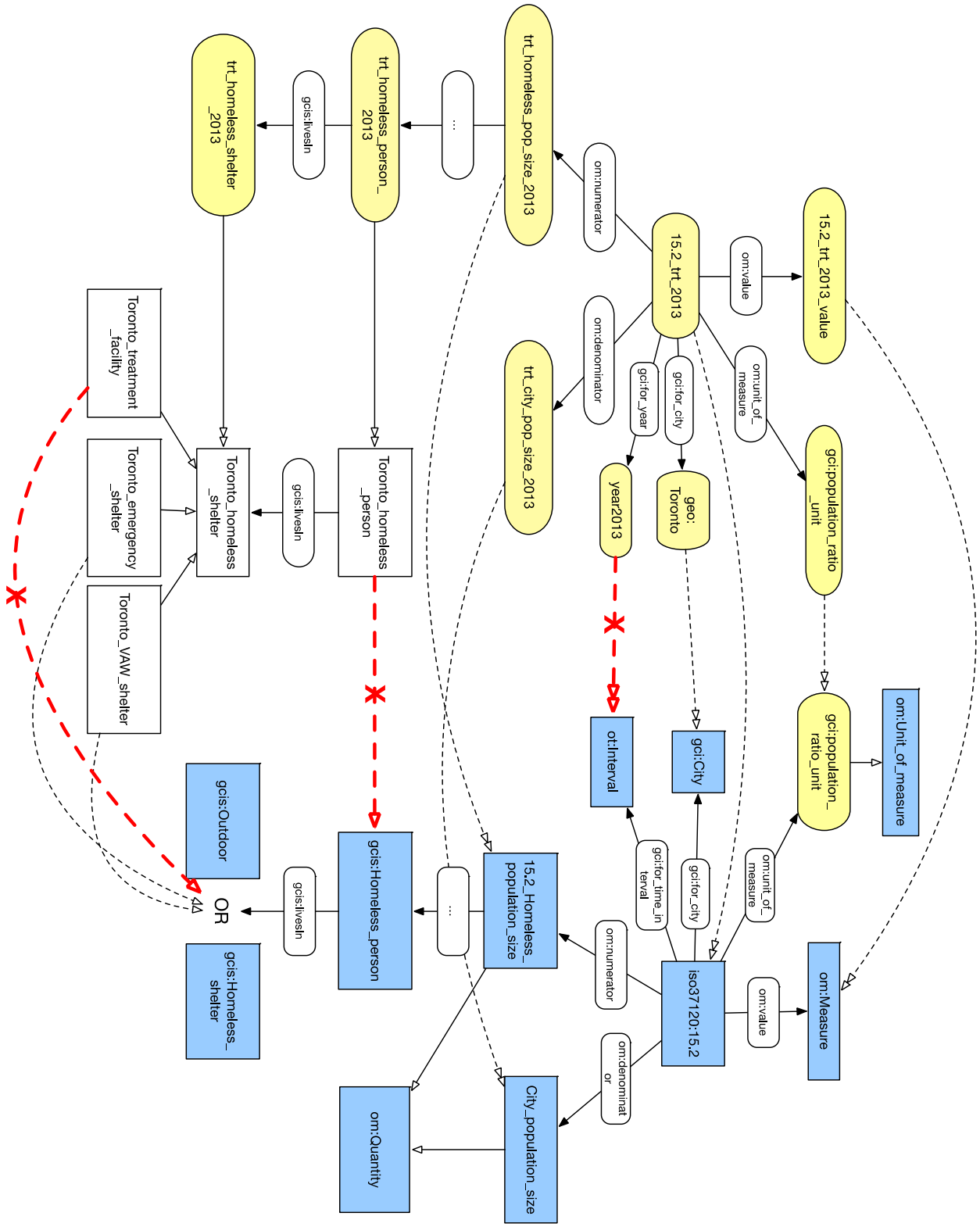


Figure 42 Type Consistency with City Definition

The values for each property must be instances of the class specified in the range restriction of the properties defined in iso37120:'15.2' as listed in Table 12 below. The 'Property' column lists the properties associated with the class iso37120:15.2. The 'Value' column lists instances linked to the 15.2_trt_2013 via these properties. 'Cardinality' and 'Restriction' columns depict cardinality and range restrictions of the corresponding property defined by iso37120:15.2 which must be satisfied by values of 15.2_trt_2013.

Property	**Cardinality**	**Restriction**	**Value**
gci:'for_city'	exactly 1	sc:City	geo:'Toronto'
gci:'for_time_interval'	exactly 1	ot:Interval	y2013
om:'unit_of_measure'	exactly 1	gci:'population_ratio_unit'	gci:'population_ratio_unit'
om:value	exactly 1	om:Measure	'15.2_trt_2013_value'
om:numerator	exactly 1	isos:'15.2_Homeless_population_size'	'trt_homeless_pop_size_2013'
om:denominator	exactly 1	isos:'City_population_size'	'trt_city_pop_size_2013'

Table 12 Properties, values, and restrictions of 15.2_trt_2013

CICC evaluates each of the instances listed above. For simplicity, we show only steps for instances where inconsistencies have been detected in this example.

Step 2. Instance: y2013

Correspondence Identification

Definition class: ot:Interval

Type Inconsistency Evaluation

Result: TC2. Type inconsistent

Description: year2013 is an instance of ot:DateTimeDescription rather than ot:Interval.

Step 3. Instance: 15.2_trt_2013_value

Correspondence Identification

Definition class: om:Measure

Type Inconsistency Evaluation

Result: No inconsistency detected

Description: 15.2_trt_2013_value is an instance of om:Measure

Temporal Inconsistency Evaluation

Result: No inconsistency detected

Description: 15.2_trt_2013_value is linked to y2013.

Measurement Inconsistency Evaluation

Result: No inconsistency detected

Description: 15.2_trt_2013_value is linked to gci:population_ratio_unit.

The CICC will then evaluate the numerator and denominator of 15.2_trt_2013. We will demonstrate the evaluation of its numerator, i.e., trt_homeless_pop_size_2013

Step 4. Instance: trt_homeless_pop_size_2013

Correspondence Identification

Definition class: isos:15.2_Homeless_population_size

Type Inconsistency Evaluation

Result: No Type inconsistency detected

Description: trt_homeless_pop_size_2013 is an instance of isos:15.2_Homeless_population_size

Measurement Inconsistency Evaluation

Result: No Type inconsistency detected

Description: trt_homeless_pop_size_2013 is linked to gci:population_cardinality_unit which is a numerator of gci:population_ratio_unit

Step 5. Instance: trt_homeless_pop_size_2013_value

Correspondence Identification

Definition class: om:Measure

Type Inconsistency Evaluation

Result: No Type inconsistency detected

Description: trt_homeless_pop_size_2013_value is an instance of om:Measure

Temporal Inconsistency Evaluation

Linked Interval: y2011

Referenced interval: y2013

Result: T2. Interval Equality Inconsistency, T1. Non-overlap Interval Inconsistency

Description: trt_homeless_pop_size_2013_value is linked to y2011 which is not equal to y2013 referred by the indicator (Figure 43). T1 also applies since y2011 is before (ot:intervalBefore) y2013.

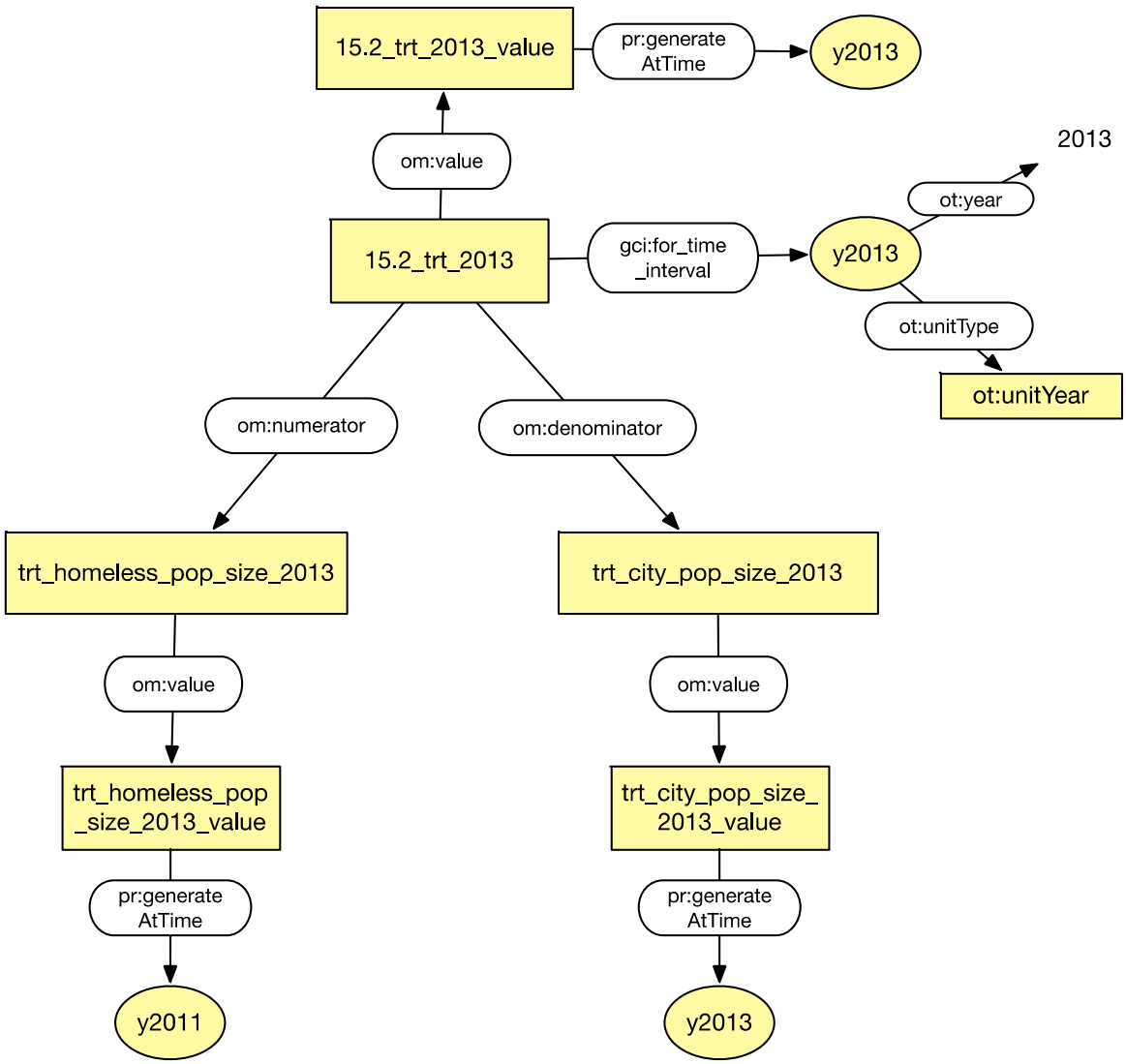


Figure 43 Temporal inconsistency example

Step 6. Instance: trt_homeless_pop_2013

Correspondence Identification

Definition class: isos:15.2_Homeless_population

Type Inconsistency Evaluation

Result: No Type inconsistency detected

Description: trt_homeless_pop _2013 is an instance of isos:15.2_Homeless_population

Place Inconsistency Evaluation

Linked Place: trt_downtown

Referenced Place: geo:Toronto

Result: G1. Place Equality Inconsistency, G2. Subplace Inconsistency

Description: trt_homeless_pop_2013 is linked to trt_downtown which is not equal to geo:Toronto referred by the indicator. It is a subarea located in Toronto. The indicator is potentially inconsistent if a placename referred by its supporting data is an area within the city referred by the indicator since the instance homeless_pop_2013 may refer to areas that include only a portion of the areas within the city referenced by 15.2_trt_2013, i.e., geo:Toronto. Thus homeless_pop_2013 is potentially inconsistent with 15.2_trt_2013 due to subplace inconsistency G2.

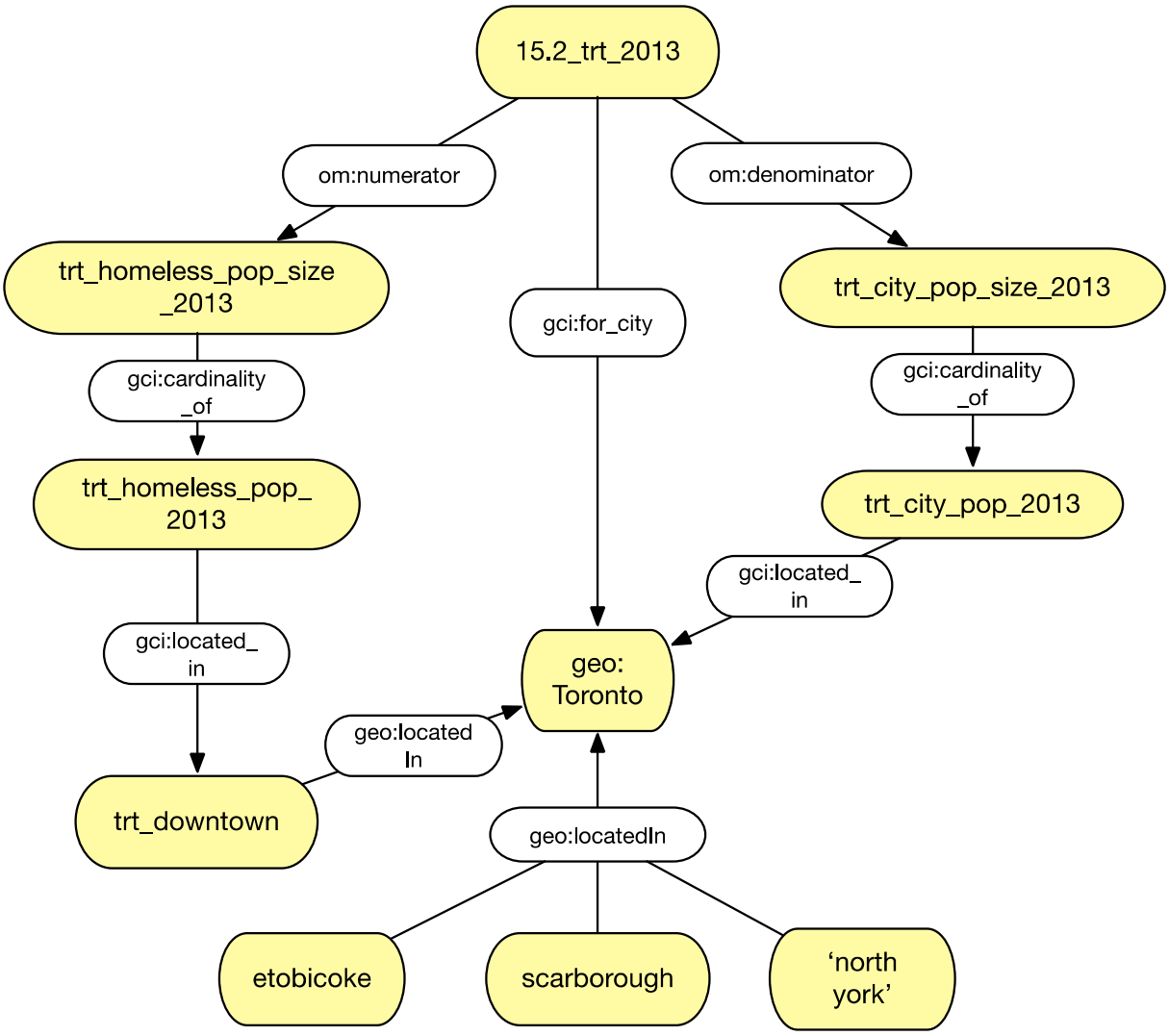


Figure 44 Subplace inconsistency example

Step 7. Instance: trt_homeless_person_2013

Correspondence Identification

Definition class: gcis:Homeless_person

Type Inconsistency Evaluation

Result: TC2. Instance Type Inconsistent

Description: the following inconsistencies exist:

- TC1(gci-trt:Toronto_homeless_shelter, gcis:Homeless_shelter)
- TC1(gci-trt:Toronto_homeless_person, gcis:Homeless_person)
- TC2(trt_homeless_person_2013, gcis:Homeless_person)
- TC3(trt_homeless_person_2013, gcis:Homeless_person)

Toronto homeless person is type inconsistent with the definition of homeless person defined by ISO 37120 since Toronto's homeless shelter types do not comply the homeless shelter types defined by ISO 37120. The value of gcis:livesIn for trt_homeless_person_2013 is trt_homeless_shelter_2013, it is type inconsistent the definition gcis:Homeless_shelter. Thus T3 also applies to trt_homeless_person_2013 and gcis:Homeless_person. Evaluation of trt_homeless_shelter_2013 is shown below.

Step 8. Instance: trt_homeless_shelter_2013

Correspondence Identification

Definition class: gcis:Homeless_shelter

Type Inconsistency Evaluation

Result: TC2. Instance Type Inconsistent

Description: the following inconsistencies exist

- TC1(gci-trt:Toronto_homeless_shelter, gcis:Homeless_shelter)
- TC2(trt_homeless_shelter_2013, gcis:Homeless_shelter)

Since gci-trt:Toronto_homeless_shelter was defined to be the union of emergency shelter, VAW shelter and treatment facility, each of the classes will be evaluated against gcis:Homeless_shelter class. Both emergency shelter and VAW shelter are subclass of gcis:Homeless_shelter but treatment facility is neither a subclass nor an equivalent class of gcis:Homeless_shelter.

The denominator of the indicator, i.e., Toronto's city population size of 2013 is evaluated with same process. In this example, we show only the measurement inconsistency occurred which was detected for the instance trt_city_pop_size_2013 which is shown below.

Step 9. Instance: trt_city_pop_size_2013

Correspondence identification

Definition class: gcis:City_population_size

Type Inconsistency evaluation

Result: TC3. Property Inconsistent

Description: the unit of measure of trt_city_pop_size_2013 is gci:population_cardinality_unit instead of '100 000th of pc' defined by the definition.

Measurement Inconsistency Evaluation

Result: M2. Indicator Unit Component Inconsistency

Description: 15.2_trt_2013 is linked to gci:population_ratio_unit. The denominator of the gci:population_ratio_unit is '100 000th of pc'. This does not satisfy the unit of measure used by trt_city_pop_size_2013 which is the denominator of 15.2_trt_2013.

As shown in Figure 45, gci:'population ratio unit', which has both its numerator and denominator as 'population cardinality unit'. Since the indicator '15.2_trt_2013' has homeless_pop_size_2013 as its numerator and city_pop_size_2013 as denominator, the unit of measure associated with the population sizes must be gci:population_cardinality_unit and '100 000th of pc' respectively. In this example, city_pop_size_2013 is linked to pc instead of 100 000th of pc therefore it is inconsistent in terms of M2.

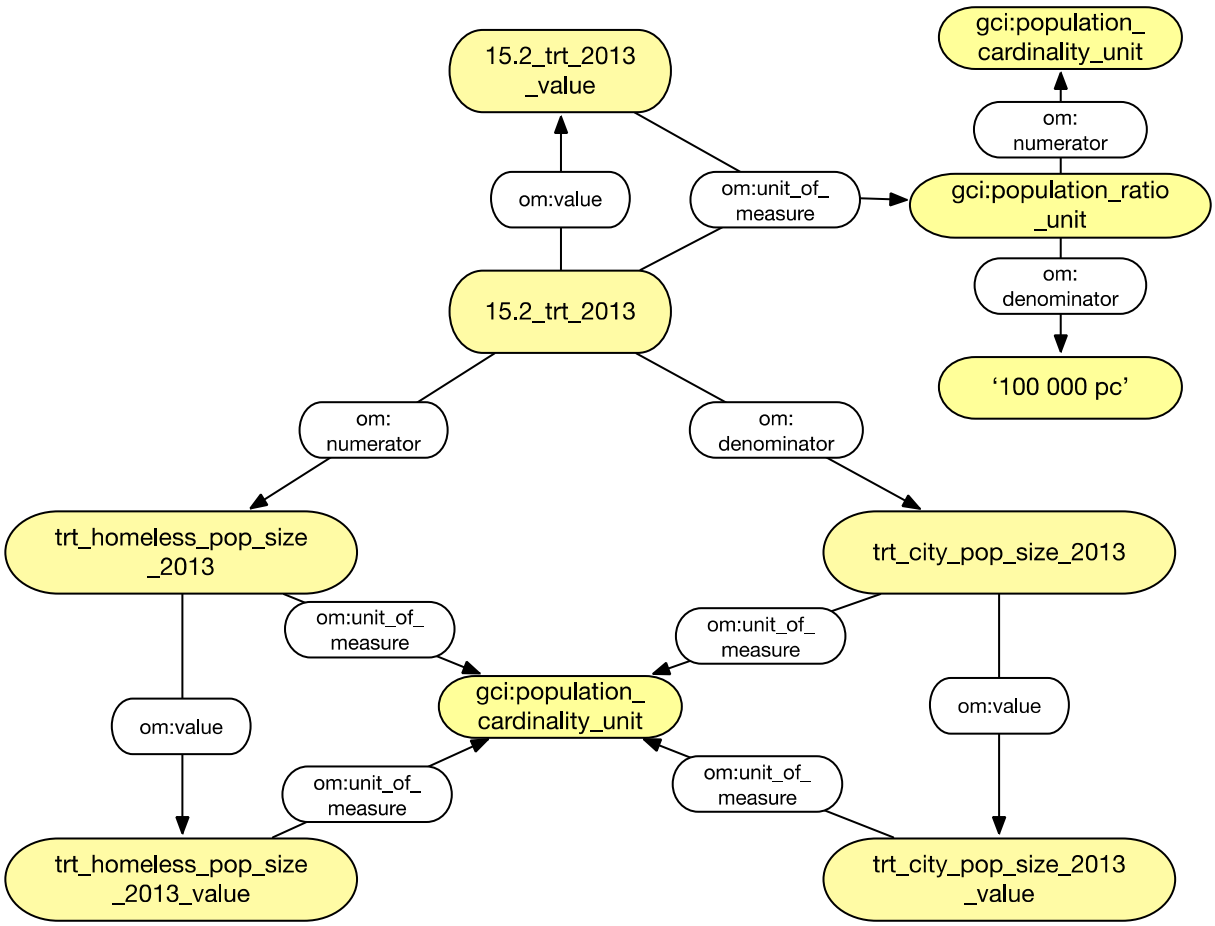


Figure 45 Indicator Unit Component Inconsistency example

Result: M3. Singular Unit Inconsistency

Description: Since '100 000th of pc' is linked to 'pc' via property om:singular_unit. Inconsistency M3 also applies to the population size of Toronto in this case.

5.3 Transversal Inconsistency Example

We will use the 15.2 Global City Indicator as an example to illustrate our transversal consistency evaluator. We define 15.2_trt_2013 and 15.2_nyc_2013 as instances of 15.2 city indicator for Toronto and New York City respectively. We have already introduced 15.2_trt_2013 indicator instance previously. Modification has been made so that it is now definitional consistent with the 15.2 indicator's definition. Specifically, Toronto's homeless person is defined to be a person who lives in an emergency shelter. Similarly, 15.2_nyc_2013 is an instance of the class iso37120:15.2

with its values of properties modeled to represent the 15.2 city indicator for New York City and it is definitional consistent with the indicator's definition represented by the GCI Ontologies. In New York City, the homeless population census report generated by NYC Department of Homeless Service included families and single adults live in homeless shelters. (Coalition for the Homeless, 2013). In order to distinguish the time intervals referred by two cities, y2013_trt and y2013_nyc has been asserted to the indicators where y2013_trt is a subinterval of y2013_nyc.

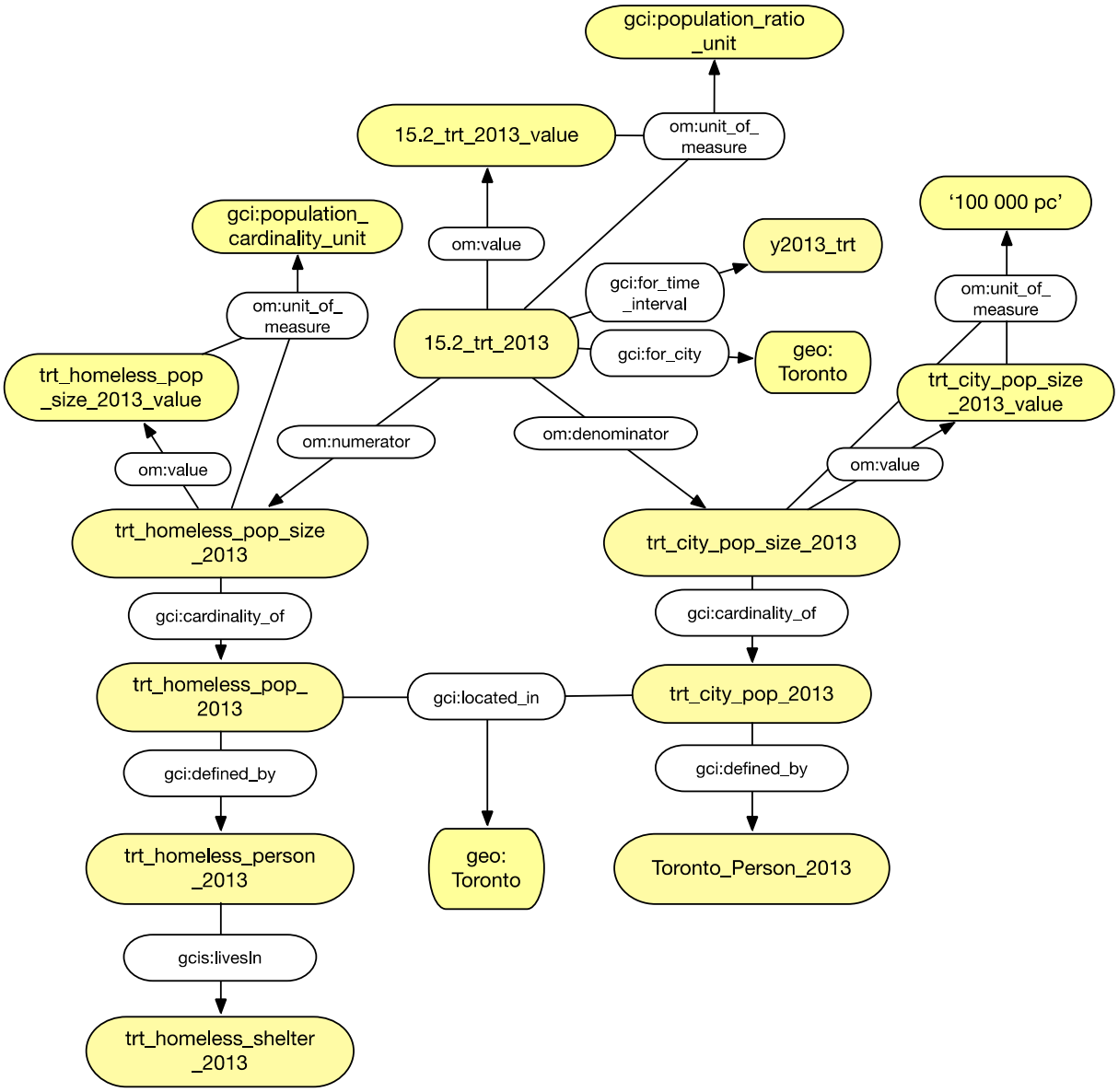


Figure 46 15.2 City Indicator data for Toronto 2013

Figure 47 15.2 City Indicator data for New York City 2013

The following inconsistencies have been embedded into the indicator data for purpose of demonstration.

Inconsistency	**Instance or Class**	**Description**
Trans_TC. Type Inconsistency	Toronto_homeless_person, NYC_homeless_person	The homeless person definitions used different types of shelters therefore are transversally type inconsistent

T2. Interval Equality Inconsistency	15.2_trt_2013, 15.2_nyc_2013	Time intervals referred by the indicators published by the two cities are different
T3. Subinterval Inconsistency	15.2_trt_2013, 15.2_nyc_2013	Interval referred by Toronto is a subinterval of that of NYC
Trans_G1. Feature Code Inconsistency	15.2_trt_2013, 15.2_nyc_2013	Feature code linked by the cities are P.PPLA and P.PPL

Table 13 List of transversal inconsistencies in example

The CICC takes instances of indicators published by two cities, i.e., 15.2_trt_2013 and 15.2_nyc_2013. Correspondence are identified using methods defined in chapter 4.

Step 1. Primary instance: 15.2_trt_2013

Correspondence Identification

Corresponding primary Instance: 15.2_nyc_2013

Transversal Type Inconsistency Evaluation

Result: Trans_TC. Inconsistent

Description: the following inconsistencies exist:

- Trans_TC(trt_homeless_shelter_2013, nyc_homeless_shelter_2013)
- Trans_TC(trt_homeless_person_2013, nyc_homeless_person_2013)

Transversal Temporal Inconsistency Evaluation

Secondary instance: y2013_trt and y2013_nyc

Result: T2. Interval Equality Inconsistent, T3. Subinterval Inconsistent

Description: y2013_trt and y2013_nyc are evaluated to be different intervals as shown in Figure 48 below. Specifically, Toronto published an indicator for January to June of 2013. While New York City publishes an indicator for the entire year. Both intervals uses ot:unitMonth as temporal unit. Thus T2(15.2_trt_2013, 15.2_nyc_2013) is true. In addition, since y2013_trt is also during (ot:intervalDuring) the y2013_nyc. T3(15.2_trt_2013, 15.2_nyc_2013) is also true since y2013_trt is a subinterval of y2013_nyc.

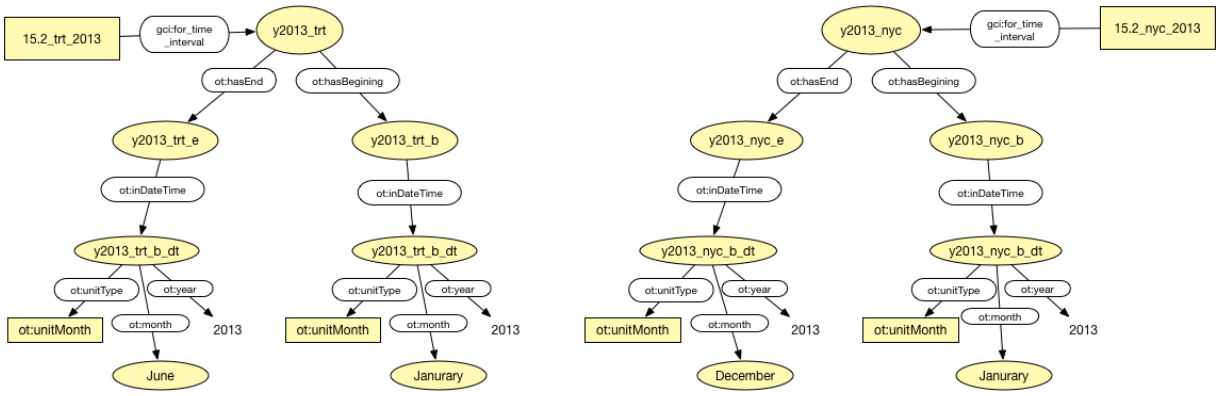


Figure 48 Representation of 2013 from Toronto and NYC with ot:unitMonth

Since both indicators are definitional consistent which means intervals referred by the supporting data are temporally consistent with y2013_trt and y2013_nyc. Therefore additional evaluations on time intervals are not necessary. CICC now evaluates place inconsistencies between 15.2_trt_2013 and 15.2_nyc_2013.

Transversal Place Inconsistency Evaluation

Secondary instance: geo:Toronto and geo:NYC

Result: Trans_G1. Feature Code Inconsistency

Description: the instances of City linked by the indicators, i.e., geo:Toronto and geo:NYC, have different feature codes of 'P.PPLA' (in this case, a provincial capital) and 'P.PPL' (a populated place) respectively.

CICC continues to evaluate the values of 15.2_trt_2013 and 15.2_nyc_2013 shown in Table 14 below. We omit these processes as they follow the same procedure described in previous section. The CICC continues to evaluate the populations and their definitions. Recall that the definition of homelessness from the two cities are defined differently.

Property	**City 1**	**City 2**
gci:'for_city'	Toronto	New York City
gci:'for_time_interval'	y2013_trt	y2013_nyc
om:'unit_of_measure'	population_ratio_unit	population_ratio_unit
om:value	15.2_trt_2013_value	15.2_nyc_2013_value
om:numerator	trt_homeless_pop_size_2013	nyc_homeless_pop_size_2013
om:denominator	trt_city_pop_size_2013	nyc_city_pop_size_2013

Table 14 Values of properties of 15.2_trt_2013 and 15.2_nyc_2013

Step 2. Primary Instance: trt_homeless_person_2013

Correspondence Identification

Corresponding Instance: nyc_homeless_person_2013

Transversal Type Inconsistency Evaluation

Result: Trans_TC. Inconsistent

Description: City specific definition differs for a homeless person. City specific definition of homeless person for the two cities are represented with the class Toronto_homeless_person and NYC_homeless_person respectively. For restriction of property gcis:livesIn, both cities are

restricted to have exactly one instance of the class Toronto_homeless_shelter and NYC_homeless_shelter. Recall that Toronto and NYC homeless shelter class are defined as shown below in Figure 49.

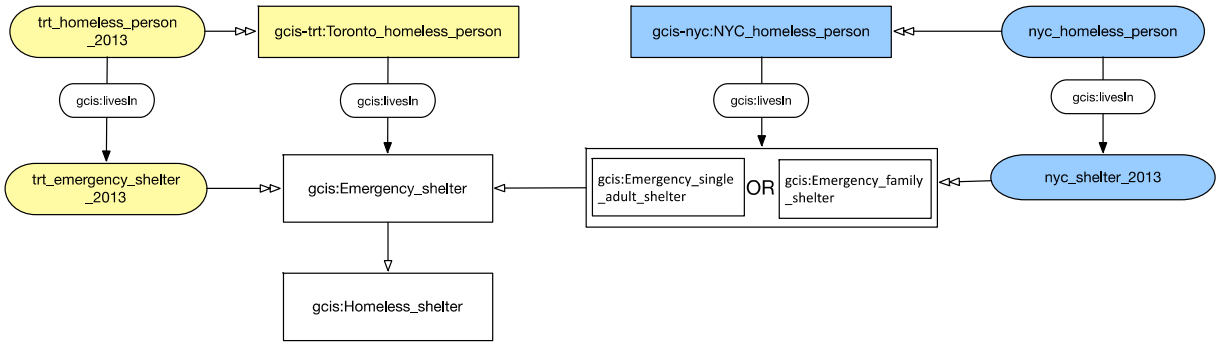


Figure 49 Toronto and NYC Homeless Person

The classes gcis:Emergency_single_adult_shelter and gcis:Emergency_family_shelter are both subclass of gcis:Emergency_shelter. In this case Toronto_homeless_person is inconsistent with NYC_homeless_person since the types of emergency shelters of NYC are single adult emergency shelter and family shelter which are subsets of shelters defined by Toronto.

Since the homeless shelter used in the definition of homeless person are different between Toronto and New York City, the indicators are transversally inconsistent in terms of type inconsistency Trans_TC. The same inconsistency applies to the instance trt_homeless_shelter_2013.

Step 3. Primary Instance: trt_homeless_shelter_2013

Correspondence Identification

Corresponding Instance: nyc_homeless_shelter_2013

Transversal Type Inconsistency Evaluation

Result: Trans_TC. Inconsistent

Description: Since both instances are instances of classes that are not equal, Trans_TC(trt_homeless_shelter_2013, nyc_homeless_shelter_2013) is true.

Since the homeless shelter used in the definition of homeless person are different between Toronto and New York City, the indicators are transversally inconsistent in terms of type inconsistency (Trans_TC), transversal temporal inconsistency (Trans_TI) and feature code inconsistency (Trans_G1).

5.4 Longitudinal Inconsistency Example

For longitudinal consistency analysis we introduce indicator data published by Toronto for 2015. Similar to transversal consistency analysis, the class representation of city specific knowledge will be evaluated between the two versions of the indicator. Suppose Toronto has changed its definition of homeless shelters by replacing treatment facilities with family shelters which was not part of the definition in 2013. In this case the 15.2 indicator of Toronto published in 2015 is inconsistent with its previous version published in 2013.

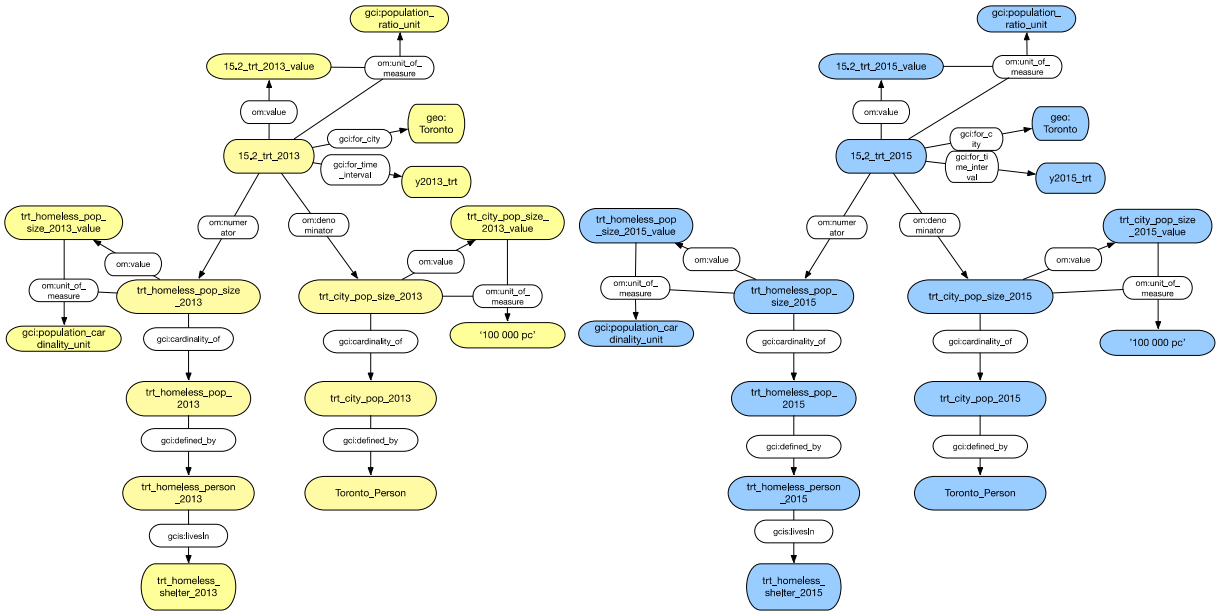


Figure 50 15.2 indicator value and supporting for Toronto in 2013 and 2015

The following inconsistencies have been embedded into the indicator data for purpose of demonstration.

Inconsistency Type	Instance or Class	Description
Long_TC. Type Inconsistency	Toronto_homeless_person_2013, Toronto_homeless_person_2015	The homeless person definitions used different types of shelters in 2013 and 2015
Long_T1	15.2_trt_2013, 15.2_nyc_2013	The interval 2015 has a different duration as the interval representing 2013.

Table 15 List of longitudinal inconsistencies in example

Step 1. Primary Instance: 15.2_trt_2013

Correspondence Identification

Corresponding Instance: 15.2_trt_2015

Longitudinal Type Inconsistency

Result: Long_TC. Inconsistent

Description: the following inconsistencies exist:

Long_TC(trt_homeless_shelter_2013, trt_homeless_shelter_2015)

Long_TC(trt_homeless_person_2013, trt_homeless_person_2015)

Longitudinal Temporal Inconsistency

Secondary instance: y2013_trt and y2015_trt

Result: Long_T1. Duration Inconsistency

Description: Toronto's 15.2 indicator is longitudinally inconsistent since the indicator was measured semi-annually in 2013 but annually in 2015. Figure 51 illustrates the case where both instances of y2013_trt and y2015_trt have the temporal unit ot:unitMonth, with y2013_trt covers the entire year of 2013 while y2015_trt covers only from January to June. Toronto's 15.2 indicator is therefore longitudinally inconsistent since the duration of time interval referred by the indicator in 2013 is different than the indicator published in 2015. Therefore Long_T1(15.2_trt_2013, 15.2_trt_2015) is true.

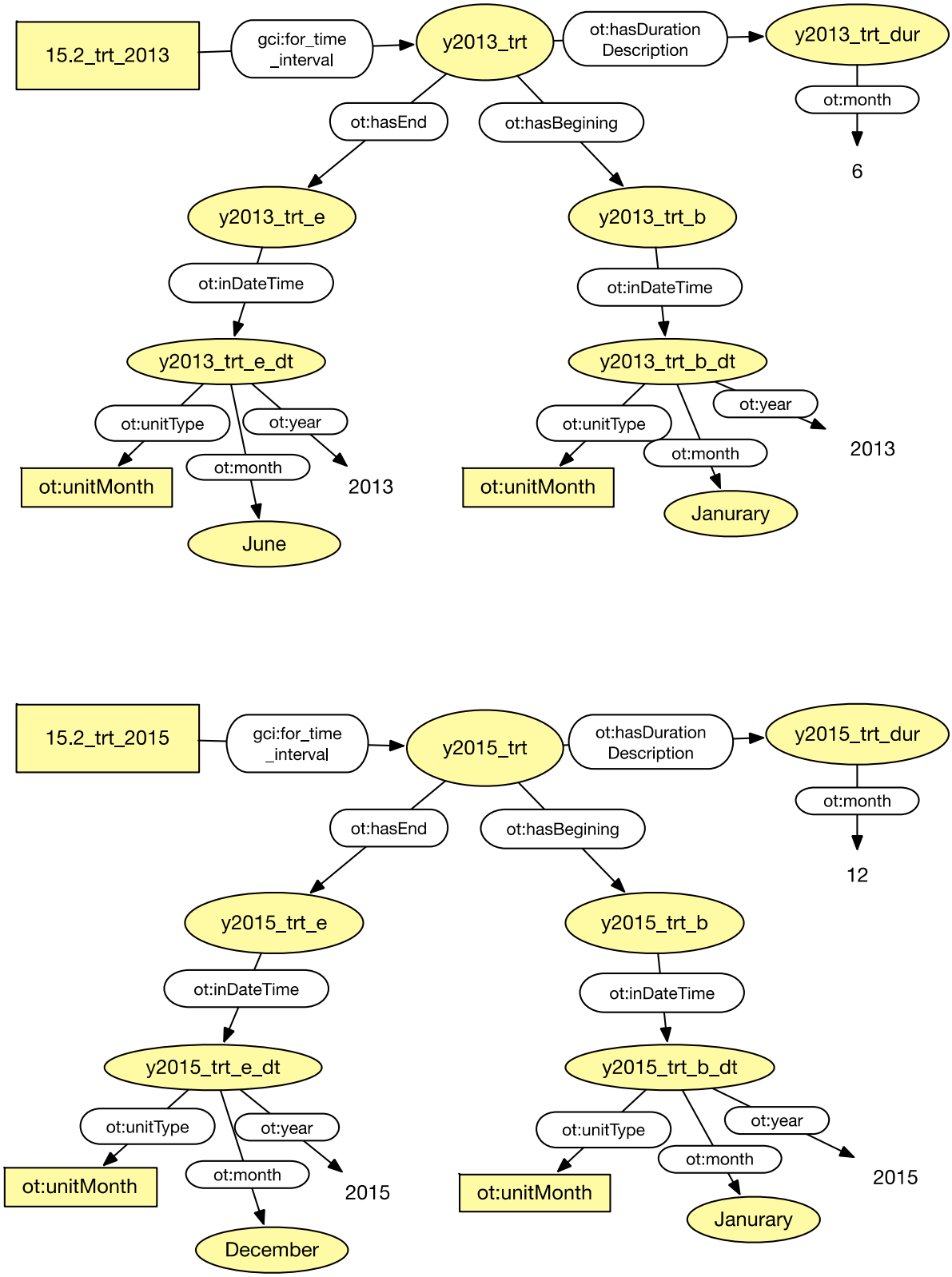


Figure 51 y2013 represents first half of 2013

Longitudinal Place Inconsistency

Secondary instance: geo:Toronto

Result: no inconsistency detected

Description: since both indicators are measuring the same instance geo:Toronto. The indicator is longitudinally consistent in terms of place.

CICC follows the same process for evaluation of supporting data instances as in transversal consistency analysis. We omit the steps taken by CICC until reaching the instances trt_homeless_person_2013 and trt_homeless_person_2015 where the following inconsistency occurs.

Step 2. Primary Instance: trt_homeless_person_2013

Correspondence Identification

Corresponding Instance: trt_homeless_person_2015

Longitudinal Type Inconsistency Evaluation

Result: Long_TC. Inconsistent

Description: the following inconsistencies exist:

Long_TC(trt_homeless_shelter_2013, trt_homeless_shelter_2015)

As shown in Figure 52, the original definition of homeless person in 2013 is shown on the left, the modified definition is on the right. The class Toronto_treatment_facility is inconsistent with the class Toronto_family_shelter. Therefore, the class Toronto_homeless_person published in 2013 is also inconsistent with the modified Toronto_homeless_person class published in 2015. Therefore Long_TC(trt_homeless_person_2013, trt_homeless_person_2015) is true and therefore 15.2 indicator of Toronto is longitudinally type inconsistent.

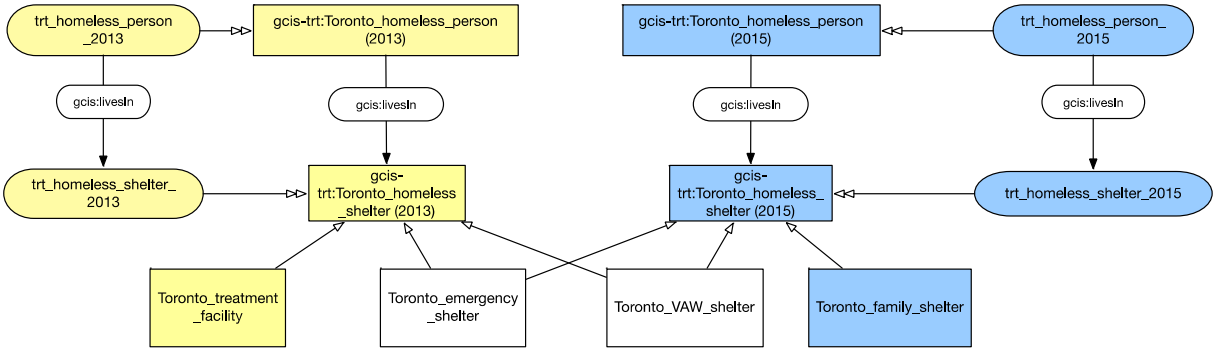


Figure 52 Toronto_homeless_person 2013 vs 2015

Step 3. Primary Instance: trt_homeless_shelter_2013

Correspondence Identification

Corresponding Instance: trt_homeless_shelter_2015

Longitudinal Type Inconsistency Evaluation

Result: Long_TC. Inconsistent

Description: Since both instances are instances of classes that are not equal, Long_TC(trt_homeless_shelter_2013, trt_homeless_shelter_2015) is true.

5.5 Summary

In this chapter we have introduced the CICC implemented using SWI-Prolog. With indicator value and supporting data represented using city specific and theme ontologies such as GCIO, CICC is capable of trace through all instances of indicator value and supporting and evaluate definitional, transversal or longitudinal inconsistencies with respect to indicator's definition, indicator data across different cities, or for the same city at different time intervals. We have demonstrated the working process of CICC using examples created based on 15.2 indicator result and homeless population size data collected by Toronto and New York City. Instances used in the example, as well as city specific and theme general ontologies (e.g., GCIO, GCI-shelters) are listed in Appendix I. A full list of Prolog predicates and descriptions are provided in Appendix II.

Chapter 6
Conclusion and Future Work

6 Conclusion and Future Work

6.1 Summary and Contributions

It is now possible to perform comparative analysis of city performance with the introduction of definition and adoption of city indicators, such as ISO 37120. The introduction of city ontologies provides a standard for openly publish both indicator definitions and the data used to derive their values. But the validity is still unknown when comparing indicator data.

This research makes a critical contribution to enabling the comparative analysis of city performance. Assuming that cities adopt a standard set of indicators (e.g., ISO 37120), and they adopt a standard set of ontologies for publishing their data on the Semantic Web (e.g., Global City Indicator Ontology), the results of this research makes it possible to determine whether the published data is consistent. If the data is not consistent, then analysis will be invalid. We have defined three categories of consistency analysis:

1. **Definitional consistency** evaluates if data used to derive a city indicator is consistent with the indicator's definition (e.g., ISO 37120).
2. **Transversal consistency** evaluates if city indicators published by two different cities are consistent with each other.
3. **Longitudinal consistency** evaluates if an indicator published by a city is consistent over different time intervals.

Inconsistency types include:

- **Correspondence inconsistency**: where a node of indicator data does not have correspondence with respect to indicator's definition or another set of indicator data
- **Type inconsistency**: the type and restrictions of properties of a node of indicator do not comply with indicator's definition or another set of indicator data

- **Temporal inconsistency**: temporal data such as interval or time instants within a set of indicator data are not internally consistent, or inconsistent with corresponding temporal data of another set of indicator in the case of transversal and longitudinal consistency analysis

- **Place inconsistency**: geographical data such as cities within a set of indicator data are not internally consistent, or inconsistent with corresponding temporal data of another set of indicator in the case of transversal and longitudinal consistency analysis

- **Measurement inconsistency**: measurement data such as unit of measure within a set of indicator data are not internally consistent, or inconsistent with corresponding temporal data of another set of indicator in the case of transversal and longitudinal consistency analysis

Consistency analysis detects actual inconsistencies where corresponding indicator data do not agree under any circumstances, e.g., comparison were made between incorrect definitions such as homeless shelters and treatment facility, or potential inconsistencies, such as temporal differences of measurements.

6.2 Future Work

Possible future research directions include the definition of additional inconsistency types and identification of sources of inconsistencies. Enriching the representation of indicator's definition using theme and city specific ontologies enables the possibilities of identifying more inconsistency types or specify existing inconsistency types with more information. The next research objective is to further identify root-causes of difference in city indicator measurement both transversally and longitudinally.

Bibliography

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11), 832-843.

Baclawski, K., Kokar, M. M., Smith, J. E., & Letkowski, J. (2001, July). Consistency checking of RM-ODP specifications. In Proceedings of the 1st International Workshop on Open Distribute Processing: Enterprise, Computation, Knowledge, Engineering and Realisation: In conjunction with ICEIS 2001 (pp. 17-26). ICEIS Press.

Baclawski, K., Kokar, M. M., Waldinger, R., & Kogut, P. A., (2002). Consistency checking of semantic web ontologies, The Semantic Web—ISWC 2002 (pp. 454 459). Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/3-540-48005-6_40

Bohannon, P., Fan, W., Geerts, F., Jia, X., & Kementsietsidis, A. (2007, April). Conditional functional dependencies for data cleaning. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on (pp. 746-755). IEEE.

Belhajjame, K., Deus, H., Garijo, D., Klyne, G., Missier, P., Soiland-Reyes, S., & Zednik, S. (2012). PROV Model Primer. http://www.w3.org/TR/prov-primer

Brickley, D. (2006). W3C Semantic Web Interest Group Basic Geo (WGS84 lat/long) Vocabulary, World Wide Web Consortium (W3C). https://www. w3. org/2003/01/geo.

British Standards Institution, The (BSI) (2014), PAS 182 Smart city concept model – Guide to establishing a model for data interoperability. BSI Standards Publication. Available at: http://www.bsigroup.com/en-GB/smart-cities/Smart-Cities-Standards-and-Publication/PAS-182-smart-cities-data-concept-model/

Capadisli, S., Auer, S., Ngonga Ngomo, A-C., (2013), “Linked SDMX Data: Path to high fidelity Statistical Linked Data for OECD, BFS, FAO and ECB”, Submitted to Semantic Web, IOS Press.

City of Toronto, (2013), 2013 Street Needs Assessment Results, http://www.toronto.ca/legdocs/mmis/2013/cd/bgrd/backgroundfile-61365.pdf

City of Toronto. (2014), “Toronto's 2013 Results Under ISO 37120 Indicators of City Service Delivery and Quality Of Life”, available at: https://www1.toronto.ca/City%20Of%20Toronto/City%20Managers%20Office/Toronto%20Progress%20Portal/ISO%2037120/Final%20-%20Summary%20of%20Toronto's%20WCCD-%20ISO%2037120%20Results-6%20.pdf

City Protocol Agreement (CPA), (2015a), City Anatomy: A Framework to support City Governance, Evaluation and Transformation, CPA-I_001_Anatomy, http://www.cptf.cityprotocol.org/ancha/CPA-I_001_Anatomy.pdf

City Protocol Agreement (CPA), (2015b), City Anatomy Indicators, CPA-PR _002_Anatomy Indicators, http://www.cptf.cityprotocol.org/CPAPR/CPA-PR_002_Anatomy_Indicators.pdf

City Protocol Agreement (CPA), (2016), Foundation Ontology for the City Anatomy, CPA-PR_003_Anatomy Ontology, http://cityprotocol.org/wp-content/uploads/2015/09/Ontology_City-Anatomy.pdf

Coalition for the Homeless (2016), New York City Homeless Municipal Shelter Population, 1983-Present, Advocacy Department, available at: http://www.coalitionforthehomeless.org/basic-facts-about-homelessness-new-york-city/

Cong, G., Fan, W., Geerts, F., Jia, X., & Ma, S. (2007, September). Improving data quality: Consistency and accuracy. In Proceedings of the 33rd international conference on Very large data bases (pp. 315 326). VLDB Endowment.

Cooper, B. (1995). “Shadow people: the reality of homelessness in the 90s”, URL: gopher:// csf.colorado.edu: 70/ 00/ hac/ homeless/ Geographical- Archive/reality-australia on 14 June, 1999.

Costello, J., Canestraro, D. S., Gil-Garcia, J. R., & Werthmuller, D. (2007). Using XML for Web Site Management: Lessons Learned Report.

Cyganiak, R., Reynolds, D., (2014), The RDF data cube vocabulary. Candidate Recommendation, W3C.

Dirks, S., & Keeling, M. (2009). A vision of smarter cities. IBM Institute for Business Value.

Dirks, S., Keeling, M., & Dencik, J. (2009). How smart is your city?: Helping cities measure progress. IBM Institute for Business Value, IBM Global Business Services, New York.

Etzion, O., & Dahav, B. (1998). Patterns of self-stabilization in database consistency maintenance. Data & knowledge engineering, 28(3), 299-319.

Falodi, J., & Fox, M. S. (2015). A health ontology for global city indicators (ISO 37120). Working paper. Enterprise Integration Laboratory, University of Toronto.

Freitas, F., Candeias Jr, Z., & Stuckenschmidt, H. (2011). Towards checking laws' consistency through ontology design: the case of Brazilian vehicles' laws. Journal of theoretical and applied electronic commerce research, 6(1), 112-126. Chicago

Forde, A., & Fox, M. S. (2015). An innovation ontology for global city indicators. Working paper. Enterprise Integration Laboratory, University of Toronto, http://eil.mie.utoronto.ca/wp-content/uploads/2015/06/GCI-Innovation-Ontology-v15.pdf

Fox, M.S., (2013), “A Foundation Ontology for Global City Indicators”, Working Paper, Enterprise Integration Laboratory, University of Toronto, Revised: 16 May 2015.

Fox, M.S., (2015a), “Polisnosis Project: Ontologies For City Indicators”, Mechanical and Industrial Engineering Department, University Of Toronto, http://eil.utoronto.ca/smartcities/papers/PolisGnosis-11mar2015.pdf

Fox, M.S., (2015b), “An Education Ontology for Global City Indicators (ISO 37120)”, Mechanical and Industrial Engineering Department, University Of Toronto, http://eil.utoronto.ca/smartcities/papers/GCI-Education.pdf

Fox, M. S., (2015c), “The role of ontologies in publishing and analyzing city indicators”, Computers, Environment and Urban Systems, 54, 266-279.

Fox, M.S., and Grüninger, M., (1998), "Enterprise Modelling", AI Magazine, AAAI Press, Fall 1998, pp. 109-121.

Fox, M.S., Barbuceanu, M., Gruninger, M., and Lin, J., (1998), "An Organisation Ontology for Enterprise Modeling", In Simulating Organizations: Computational Models of Institutions and Groups, M. Prietula, K. Carley & L. Gasser (Eds), Menlo Park CA: AAAI/MIT Press, pp. 131-152.

Fox, M.S., and Huang, J., (2003), “Knowledge Provenance: An Approach to Modeling and Maintaining the Evolution and Validity of Knowledge”, EIL Technical Report, University of Toronto.

Fox, M.S., and Huang, J., (2005), "Knowledge Provenance in Enterprise Information", International Journal of Production Research, Vol. 43, No. 20.

Freitas, F., & Lins, F. (2012). The Limitations of Description Logic for Mathematical Ontologies: An Example on Neural Networks. In ONTOBRAS-MOST (pp. 84-95). Available at: http://ceur-ws.org/Vol-938/ontobras-most2012_paper7.pdf

Geppert, A., & Wimmers, E. L., (2000), Consistency Constraints in Database Middleware.

Global City Indicators Facility (2010a). “Overview Report of the Global City Indicators Facility”, GCIF, University of Toronto.

Global City Indicators Facility (2010b). “Global City Indicators©: Definitions and Methodologies” September 2010, GCIF, University of Toronto.

Grau, B. C., (2006), “OWL 1.1 Web Ontology Language Tractable Fragments”, The University of Manchester, http://www.w3.org/Submission/owl11-tractable/

Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., & Ayers, D. (2009). “Scovo: Using statistics on the web of data.” In The Semantic Web: Research and Applications (pp. 708- 722). Springer Berlin Heidelberg.

Hoornweg, D., Nunez, F., Freire, M., Palugyai, N., Herrera, E.W., and Villaveces, M., (2007), "City Indicators: Now to Nanjing", World Bank Policy Research Working Paper 4114.

Horridge, M., Parsia, B., & Sattler, U. (2009, September). Explaining inconsistencies in OWL ontologies. In International Conference on Scalable Uncertainty Management (pp. 124-137). Springer Berlin Heidelberg.

Horrocks, I. (1999). FaCT and iFaCT. Description logics, 22. Chicago

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. W3C Member submission, 21, 79.

Hunter, A., & Konieczny, S. (2005). Approaches to measuring inconsistent information. In Inconsistency tolerance (pp. 191-236). Springer Berlin Heidelberg.

Hunter, A., & Konieczny, S. (2006). Shapley Inconsistency Values. KR, 6, 249-259.

Hunter, A., & Konieczny, S. (2008). Measuring Inconsistency through Minimal Inconsistent Sets. KR, 8, 358-366.

ISO37120, (2014), "ISO 37120: Sustainable Development of Communities – Indicators for City Services and Quality of Life", International Organization for Standardization, First Edition, 2014-05-15, ISO37120:2014(E).

Mariuţa, Ş. (2014). Principles of Security and Integrity of Databases. Procedia Economics and Finance, 15, 401-405. Chicago

Martinez-Cruz, C., Blanco, I. J., & Vila, M. A. (2011). Ontologies versus relational databases: are they so different? A comparison.

McCarney, P. L., (2011) "Cities and climate change: The challenges for governance"

Coordinating Lead Author with H. Blanco, J. Carmin and M. Colley, Ch 9 in Climate Change and Cities: First Assessment Report of the Urban Climate Change Research Network, Cambridge University Press.

Mendel-Gleason, G. E., Brennan, R., & Feeney, K., (2015), “Ontology Consistency and Instance Checking”, available at: http://ceur-ws.org/Vol-1376/LDQ2015_paper_03.pdf

Motik, B., Sattler, U., & Studer, R. (2005). Query answering for OWL-DL with rules. Web Semantics: Science, Services and Agents on the World Wide Web, 3(1), 41-60.

Norton, B., Vilches, L. M., Len, A. D., Goodwin, J., Stadler, C., Anand, S., Harries, D., VillaznTerrazas, B., and Atemezing, G. A. (2012). NeoGeo Vocabulary Specification – Madrid Edition. Juan Martin Salas and Andreas Harth (editors), http://geovocab.org/doc/neogeo/

OECD, (2013). Main Economic Indicators (MEI), https://stats.oecd.org/glossary/detail.asp?ID=1577

Pan, F., & Hobbs, J. R. (2004, March). Time in owl-s. In Proceedings of the AAAI Spring Symposium on Semantic Web Services (pp. 29-36).

Parsiaa, B., Sirinb, E., Graua, B. C., Ruckhausa, E., & Hewlettb, D. (2005). Cautiously approaching SWRL. University of Maryland, Tech. Rep.

Patel-Schneider, P. F., Hayes, P., and Horrocks, I., (2004), OWL web ontology language semantics and abstract syntax. W3C Recommendation. Available at http://www.w3.org/TR/owl-semantics/

Rijgersberg, H., Wigham, M., and Top, J.L., (2011), “How Semantics can Improve Engineering Processes: A Case of Units of Measure and Quantities”, Advanced Engineering Informatics, Vol. 25, pp. 276-287.

Rijgersberg, H., van Assem, M., & Top, J. (2013). Ontology of units of measure and related concepts. Semantic Web, 4(1), 3-13.

Sattler U., Stevens R., Lord P., (2014), How does a reasoner work?. Ontogenesis. http://ontogenesis.knowledgeblog.org/1486

Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408.

2001. Available at: http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2001/oiled-dl.pdf

Shearer, R., Motik, B., & Horrocks, I., (2008), “HermiT: A Highly-Efficient OWL Reasoner”, In OWLED (Vol. 432, p. 91).

Smith, A. D. (2011). IBM Intelligent Operations Center key performance indicators (KPIs), Part 1: Defining a low-level KPI. Available at: http://www.ibm.com/developerworks/industry/library/ind-iockpi1/ind-iockpi1-pdf.pdf

Smith, A. D. (2013). IBM Intelligent Operations Center KPI Implementers Guide for Websphere Software. Document version, 1.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y., (2007), “Pellet: A practical owl-dl reasoner”, Web Semantics: science, services and agents on the World Wide Web, 5(2), 51-53.

Tsarkov, D., & Horrocks, I. (2006), “FaCT++ description logic reasoner: System description”, In Automated reasoning (pp. 292-297). Springer Berlin Heidelberg.

Uceda-Sosa, R., Srivastava, B., and Schloss, B., (2011), “Building a Highly Consumable Semantic Model for Smarter Cities”, In Proceedings of the workshop on AI for an Intelligent Planet, ACM.

Uceda-Sosa, R., Srivastava, B., and Schloss, B., (2012b), “Using Ontologies to make Smart Cities Smarter”, Available at: http://researcher.ibm.com/researcher/files/us-rschloss/SemTech_SCRIBE_2012Jun_V2.ppt

United Nations Centre for Human Settlements (UN-Habitat) (2000), “Strategies to combat homelessness”, Nairobi, Kenya, http://ww2.unhabitat.org/programmes/housingpolicy/documents/HS-599x.pdf

Wang, Y., Fox, M.S., (2015), “A Shelter Ontology for Global City Indicators (ISO 37120)”, Enterprise Integration Laboratory, University of Toronto, Working Paper.

Welty, C., McGuinness, D. L., & Smith, M. K. (2004). Owl web ontology language guide. W3C recommendation, W3C (February 2004) http://www. w3. org/TR/2004/REC-owl-guide-20040210.

Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). Swi-prolog. Theory and Practice of Logic Programming, 12(1-2), 67-96.

Appendix I – List of Files

Global Cities Indicator Ontologies

ISO37120 Indicator Identifier Ontology: http://ontology.eil.utoronto.ca/ISO37120.owl

Global City Indicator Foundation Ontology:
http://ontology.eil.utoronto.ca/GCI/Foundation/GCI-Foundation.owl

Global City Indicator Education Ontology: http://ontology.eil.utoronto.ca/GCI/Education/GCI-Education.owl

Global City Indicator Shelter Ontology: http://ontology.eil.utoronto.ca/GCI/Shelters/GCI-Shelters.owl

ISO 37120 Indicator Definitions

ISO 37120 Education Theme Indicators Definitions:
http://ontology.eil.utoronto.ca/GCI/ISO37120/Education.owl

ISO 37120 Shelter Theme Indicators Definitions:
http://ontology.eil.utoronto.ca/GCI/ISO37120/Shelters.owl

City of Toronto 2013 ISO 37120 Indicators

Education Theme Indicators:
http://ontology.eil.utoronto.ca/ISO37120/Toronto/2013/ISO37120_6_2013_TO.ttl

Shelter Theme Indicators:
http://ontology.eil.utoronto.ca/ISO37120/Toronto/2013/ISO37120_15_2013_TO.ttl

The following files contain fiction data and ontologies used for the purpose of testing the CICC

City Specific Ontologies

GCI Shelter Toronto: GCI-Shelters_Toronto.owl

GCI Shelter New York City: GCI-Shelters_NYC.owl

Toronto 15.2 Indicator

15.2_trt_2013_instance.ttl

15.2_trt_2015_instance.ttl

New York City 15.2 Indicator

15.2_nyc_2013_instance.ttl

City Indicator Consistency Checker

The following files contain the source code of the CICC. Axioms from prolog-utilities.pl, OWL-axioms.pl and OT-axioms.pl are adapated from professor Mark S. Fox at the University of Toronto.

- OWL-axioms.pl
- OT-axioms.pl
- prolog-utilities.pl
- consistency_utilities.pl
- consistency_internal.pl
- consistency_cardinality.pl
- consistency_definitional.pl
- consistency_transversal.pl
- consistency_longitudinal.pl
- consistency_trans_support.pl

Appendix II – CICC Refernce Manual

User Instruction

To upload files, run fileupload.pl. This will start file uploading portal.

To start CICC user interface, run startui.pl. This will start UI in Prolog where users can enter target instance/class names.

To start directly in SWI-Prolog, run [‘regPrefix’].

To output result to a file, use the following, for example,

```
?- tell('output.txt'),check_def_list_start('15.2_trt_2013'),told,!.
```

To display detailed trace and debug information, turn gci_trace on.

```
?- gci_trace(on).
```

Definitional consistency.

To check definitional consistency of an instance. Query the following predicate:

E.g. for ’15.2_trt_2013’ and corresponding definition class is37120:15.2

```
?- check_def_start(’15.2_trt_2013’, iso37120:’15.2’).
```

For each node, results will display as:

Instance type consistency: Pass or Fail

Cardinality check: Pass or Fail or ‘Does not have cardinality restriction’

Result for internal consistency for Placename, Temporal and Units can be found at the end of display.

Transversal Consistency

```
?- check_trans_list_start(’15.2_trt_2013’,’15.2_nyc_2013’).
```

Note: The query will fail if the ‘Year’ values (gci:’for_time_interval’) for two instances are different.

Longitudinal Consistency

```
?- check_long_list_start(’15.2_trt_2013’,’15.2_trt_2015’).
```

Note: The query will fail if the ‘Year’ values (gci:’for_time_interval’) for two instances are the same and values for ‘City’ are different.

Testing

To automate error generation, call

```
?- loop_test(N)
```

where N is the number of runs. Result will be stored in output file output’N’.txt under current directory.

Note: currently this only runs test for definitional consistency. Transversal and longitudinal test can be run manually with the following predicates.

```
?- tell('output_trans.txt'), auto_error,
check_trans_list_start('15.2_trt_2013','15.2_nyc_2013'),told,!.

?- tell('output_long.txt'), auto_error,
check_long_list_start('15.2_trt_2013','15.2_trt_2015'),told,!.
```

Note: there is a slight chance that loop_test/1 will run into an infinite loop. This occurs when auto_error/0 creates a loop in values in one of the instances. E.g. (X, p1, Y), (Y, p2, X).

Predicate Description

regPrefix.pl

initialization file

consistency_utilities.pl

owl_triples(?Class, ?Property, ?Range)

Retrieve owl triples in the form of (Class, Property, Range) since rdf (Sub, P, Obj) works only on individuals. Triples are defined as Restrictions by assigning an abstract class in OWL.E.g. rdf(gci:'Population_size',gci:'cardinality_of',X) will return false. Cardinality information is omitted. It is based on owl_range_restriction/3 but it covers all restriction type instead of owl:'onClass' only and will also retrieve information about inherited properties and range restriction.

owl_triples_union(?Class, ?Property, ?UnionList, ?List)

class_union(?Class, ?UnionList)

Extract each classes in a union into a list

owl_disjoint_class(?Class1, ?Class2)

returns true if ?Class1 and ?Class2 are disjoint classes or are subclasses of two disjoint classes.

combine_two_list(?List1, ?List2, ?ListCombined)

Combine two separate lists into a single list.

units_compatable(Xunit,Yunit)

Xunit and Yunit are units. They are compatible if Xunit is a subclass of Yunit, or the singular unit of Xunit is a subclass of Yunit.

units_ins_match(X,P,Y)

The units of X and Y are related with P given that X P Y. X and Y are instances (of Quantity), P is a property relating X and Y. The units of X and Y must also have a relationship P between them. For example, pc is numerator of Population Ratio Units, homeless pop size is

numerator of 15.2 indicator. The pairs (15.2, homeless pop size) and (Population ratio units, pc) are both related with om:numerator.

consistency_definitional.pl

class_prop(?Class, ?PList)

return all properties of ?Class and store in ?PList.

class_range(?Class, ?Property ?NList)

return all range restrictions of ?Property associated with ?Class and store in ?NList.

leaf_node(?Class)

True if ?Class does not have any properties.

class_individual_prop(?Instance ?Class ?List)

Return all properties of ?Instance as defined by ?Class and store the properties in a ?List. setof/3 was used with template being the intersection of ?Prop from owl_triplets/3 and rdf/3. Only properties that exist in the triples of ?Instance are returned since required properties can be checked through cardinality checks.

class_individual_value (?Instance ?Property ?List)

For a ?Property of ?Instance, return all values and store in a ?List. Calls setof/3. For testing purpose only. Not used anymore.

rdf_iso_list(?Class, ?Def, ?PList, ?NestedList, ?NDefList)

Return all ranges of all properties from ?PList of ?Class and store into ?NestedList. Return all ranges of all properties from ?PList of ?Def and store into ?NDefList This is a recursive call of bagof/3 until end of list is reached. This is used to retrieve triples with properties of interest only (exists in both ?Class and ?Def).

?NestedList is a nested list but it will catch all values of the same property. E.g. Homeless person livesIn Shelter1, Shelter2, etc will be stored in a list as [[Shelter1, Shelter2,..],[…]]. This list will be flatten in generate_list_iso/2

This predicate will fail out when, for example, owl_triples(X, for_city_service, Y) where Y doesn't exist. But this is always called after class_individual_prop/3 where each property in [H|T] was ensured to have a value.

generate_list_iso(?Class, ?Def, ?PList, ?NList, ?DefList)

Return all range restriction that is linked to ?Class via properties from ?PList as ?NList, and all range restrictions from ?Def as ?DefList. This predicate is a calls rdf_out_list/3. ?NestedList passed from rdf_iso_list/3 is flattened in this predicate.

check_iso_list(?List, ?List2)

?List is the list of elements to evaluate with respect to definition class in ?List2. Evaluate definitional consistency of all element in the list. Evaluation has two parts:

- Instance type check: check if an individual is an instances of the correct class as specified in definition. Calls **check_def/2** predicate with respect to each instance in ?List.
- Cardinality check: check the cardinality of each property and compare with the cardinality restriction from the definition. Calls **check_def_card/1** for each instance in ?List.

It then generates a list (NList) and a list of classes from the definition DefList of values of current instance by calling generate_list_iso/5. NList is appended to (calls append/3) the tail of ?List as NewList. DefList is appended to (calls append/3) the tail of ?List2 as NewDefList. Recursively call itself with respect to NewList and NewDefList. The base case is check_def_list([]).

Check_def (?Class, ?Def)

Check if ?Class is an individual of, subclass of, or equal to ?Def. Both ?Class and ?Def are classes.

Check_def (X, ?Def)

?Def is the definition class such as iso37120:15.2. For each X, X can be a class or an instance, checks according to follow:

- if X is an instance then evaluate its type by check_def_class/2
- if X is a class then evaluate itself with check_def_class/2

check_def_class_prop (?Class, ?Def)

This predicate evaluates if ?Class has all properties and range restriction specified by ?Def. checks if ?Class and ?Def are disjoint classes or a leaf, if true then fail, else continue. Get properties of ?Def with class_prop/2 and store as PList. Evaluate all properties in PList with check_all_prop/3.

check_all_prop(?Class, ?Def, PList)

Get range restriction of property of the definition ?Def with owl_triples(?Def, P, V). Get all range restriction of ?Class with class_range/2 and store all restrictions as NList. Evaluate all elements in NList with respect to V using check_all_value/4. Runs recursively until PList is empty.

check _all_value(?Class, ?Property, ?Val, NList)

Evaluate each element in NList with respect to ?Val using check_def_class/2 or check_def_class_prop/2. Making sure all range restriction of ?Class is consistent with ?Val. Runs recursively until NList is empty.

check_def_card(?Instance)

Check the cardinality of each property and compare with the cardinality restriction from the definition**.** Check the 'Type' of ?Instance with rdf/3. Get list of properties of ?Instance with cardinality restrictions. Calls class_pcard_prop/3. If false, then return a message indicating that there is no cardinality restriction on the property**.** Check cardinality restrictions with check_def_all_pcard/3

Check_def_list_start(?Instance, ?Class)

First predicate to call. Check the first node with check_def/2 and check_def_card/1 and generate a list of values and pass into check_iso_list/1

check_time_city_def(?Instance)

check if placename and temporal values of ?Instance are internally consistent. Calls time_city_consistent/3

class_pcard_prop(?Instance,?Class, ?List)

Similar to class_individual_prop/3 but return only properties with cardinality restrictions and store in ?List.

check_def_all_pcard(?Instance, ?Class, ?List)

?List is a list of properties. For an instance, check the cardinality of values of all of its properties with cardinality restrictions. Return the value of cardinality restriction of ?Class for head of ?List which is a property. Calls owl_prop_cardinality/5 Count the number of triples that satisfy rdf(?Instance, property, _) and match the number with cardinality restrictions. Calls inst_pcard_check/6 Recursively call check_def_all_pcard/3 with ?List replaced by tail of ?List.

Consistency_Internal.pl

find_all_related_start(?instance, ?List)

Calls find_all_related/2.

find_all_related(?List, ?ResultList)

For an instance, return all values that are related to it via any property. Results are stored in a list. Find all values of ?Instance. Calls generate_list_def/2. Append generated list Recursively generating list of values

find_all_city(?Instance,?List)

For an instance, return all values that are related to it via gci:'for_city' and gci:'located_in' property. Results are stored in a list.

find_all_time(?Instance,?List)

For an instance, return all values that are related to it via gci:'for_time_interval' and pr:'effective' property. Results are stored in a list. This will be modified to cover the case when an area is within a city or vice versa.

city_consistent_start(?Instance, ?List)

Calls finds_all_related/2, finds_all_city/2, city_consistent/2, and city_consistent_sub in order.

city_consistent(?List, City)

To be used after find_all_city/2. Evaluate all instances of City from list resulted are the same. Detects G1 inconsistency

city_consistent_sub(?List, City)

Evaluate all instances of City from ?List to verify if they are locatedIn City. This predicate checks rdf triples rdf(X, gci:located_in, City) and rdf(X, geo:locatedIn, City) where X are elements in ?List. Detects inconsistency G2.

city_consistent_dynamic(?List, City)

Evaluate all instances of City from ?List to verify if they are effective in the same year as City. This predicate checks rdf triples rdf(X, kp:effective, Year), rdf(City, kp:effective, Year) where X are elements in ?List. Detects inconsistency G3.

time_consistent_start(?Instance, ?List)

Calls finds_all_related/2, finds_all_time/2, time_consistent/2.

time_consistent(?List, Time)

To be used after find_all_time/2. Evaluate if all instances of Time Intervals (ot:Interval) from list resulted are consistent. Temporal consistency of intervals associated to instances related to indicator Two intervals are consistent if they are the same instance (rdf_equal/2) or satisfy time_interval_equal/2 It is noted if an interval is during another (time_during/2). Detects inconsistency T2.

time_during(?Instance, Time)

?Instance is an instance of ot:Interval. Evaluates if ?Instance is during Time. Checks for ot:intervalDuring property or calls ot_interval_during/2. Detects inconsistency T3. Other interval relations in T1 and T3 can be added here.

time_overlap/2, time_meets/2, time_starts/2, and time_end/2 are defined in similar manner.

time_result_pass(?List, ?Instance)

output message. List all instances related to indicator that is associated with temporal ?instance. Same for time_result_fail/2 and time_result_potential/2.

time_city_consistent(?Instance, Time, City)

combination of time_consistent_start/2 and city_consistent_start/2 so both can be checked with a single predicate against an instance.

unit_consistent_start(?Instance, Unit)

Calls finds_all_related/2, finds_all_unit/2, unit_consistent/2.

find_all_unit(?Instance,?List)

return all instances that has a value for property om:unit_of_measure.

unit_qi_consistent_start(?Instance)

Calls find_all_related/2, then finds all instances of om:Quantity. Store the instances to a ?List then calls unit_ins_consistent/2.

unit_ins_consistent(?List, Unit)

?List is all instances of om:Quantity related to the indicator. Unit is the unit of measure of the indicator. Calls units_ins_match/5. Detects M2 inconsistency.

units_ins_match(?InstanceX, ?Property, ?InstanceY)

?InstanceX and ?InstanceY are related by ?Property. Evaluates if the units of ?InstanceX and ?IntanceY are also related by ?Property, or if the singular units of ?InstanceX and ?InstanceY are related by ?Property.

units_ins_match(?InstanceX,?Property,?InstanceY,XUnit,YUnit,YSingUnit)

same as units_ins_match/3 but also returns the units.

unit_qm_consistent_start(?Instance)

unit_qm_consistent(?List)

check if the unit of measure for each Quantity and its Measure match. The units must be exactly the same so do not need to specify a Unit. Detects inconsistency M1.

unit_multiple_start(?List, Unit)

unit_multiple(?List, Unit)

Find the singular unit of units linked to elements in ?List, and compare if it is the same instance as Unit, or it is linked to Unit via some property P. Detects M3.

find_all_quantity(?List1, ?List2)

find all instance of om:'Quantity' from listed generated in find_all_related/2.

find_all_measure(?List1, ?List2)

find all instance of om:'Measure' from listed generated in find_all_related/2.

find_all_unit_multiple(?List1, ?List2)

find all instance of om:' Unit_multiple_or_submultiple' from listed generated in find_all_related/2.

consistency_cardinality.pl

owl_prop_cardinality(?Class, ?Property, ?Range, ?Restriction, ?Number)

Return the range, restriction and cardinality of a property of a class. PCard is cardinality type which is either owl:'qualifiedCardinality', owl:'minQualifiedCardinality', or owl:'maxQualifiedCardinality'. The value of cardinality restriction is a literal thus we need to convert it to an atom of number with atom_number/1.

convertNum(?Literal, ?Number)

Convert ?Literal to an atom.

owl_prop_card_exactly(?Instance, ?Property, ?Range, ?Number, Count)

Count number of triples with ?Instance and ?Property, that is, rdf(?Instance, ?Property, _) with aggregate_all/3. Match this number (Count) with the cardinality restriction (?Number) with =:=/2.

Same for owl_prop_card_min, owl_prop_card_max but check for <=/2 and >=/2 respectively.

inst_pcard_check(?Instance, ?Property, ?Range, ?Restriction, ?Number ,Count)

Based on the type of ?Restriction then call one of the three predicates above. A cut operator (!) was implemented at the end to eliminate choice points.

consistency_transversal.pl

gci_same_type(?instanceX, ?InstanceY)

simply check if X and Y are the same type.

same_year(?InstanceX, ?InstanceY, ?Year)

If X and Y has the same value for ‘for_time_interval’ property.

card_trans(?InstanceX, ?InstanceY, ?Property ?Number)

Check if X and Y has the same number of values for ?Property by counting with aggregate_all/3.

trans_subclass(?InstanceX, ?InstanceY)

check if X and Y are instances subclasses of each other. In this case they are potentially inconsistent. Calls rdf_reachable/3.

Check_trans_class(?ClassX, ?ClassY)

Checks if ?ClassX and ?ClassY are disjoint classes. Then return a list of properties. Then call check_trans_all_prop/3.

check_trans_all_prop (?ClassX, ?ClassY, ?PList)

Return range restriction of ?ClassX and ?ClassY as NXList and NYList for each property in ?PList. Then calls check_trans_all_value/5 to evaluate range restrictions of ?ClassX and ?ClassY.

check_trans_all_value (?ClassX, ?ClassY, ?Property, ?NXList, ?NYList)

Calls check_def_class/2 on elements of ?NXList and ?NYList, calls check_trans_class/2 if fails. Classes are inconsistent if both check_def_class/2 and check_trans_class/2 fails.

check_trans_list_start(?InstanceX, ?InstanceY)

Calls same_year/2 to check if both instances have the same time interval. Then evaluate ?InstanceX and ?InstanceY by calling (check_def_class/2 ; check_trans_class/2). Then calls generate_list/3.

check_trans_list(?List)

similar to check_def_list/1 but ?List is a nested list. Each element in ?List is a pair of instance in the form [X,Y]. e.g. [[X1,Y1,],[X2,Y2],…]. Calls (check_def_class/2 ; check_trans_class/2) for each pair of [X,Y] generate new list and append to the tail of ?List.

class_prop_trans(?ClassX, ?ClassY, ?Class, ?List)

return a list of properties from both classes.

class_individual_prop_trans(?InstanceX, ?InstanceY, ?Class, ?List)

return a list of properties of a class that an individual has value of.

rdf_out_trans_list(?InstanceX, ?InstanceY, ?PList, ?XList, ?YList)

For each property in ?PList, return all values of X and Y and store them into XList and YList respectively. card_trans is called within this function.

rdf_out_list_n(?InstanceX, ?InstanceY, ?PList, ?NList)

calls class_individual_value/3 on each property in ?PList on ?InstanceX and ?InstanceY to return two lists XList and YList which are list of range restrictions of X and Y respectively. Calls card_trans/4 to check cardinality of property for both ?InstanceX and ?InstanceY. Calls combine_list_groupby/3 and flatten/3 to generate ?NList.

combine_list_groupby(?List1,?List2,?FinalList)

combine ?List1 and ?List2 into a nested list ?FinalList. Each element is in the form [X,Y] and is grouped by the class. E.g. X has values C1 and C2 for property p while Y has C3 and C1 as values of p. The list returned will be [[C1, C1], [C2, C3]]. Incorrect comparison such as [C1, C3], [C2, C1] shall be avoided. Calls same_type_list/3 and diff_type_list/3.

same_type_list(?List1, ?List2, ?FinalList)

used in combine_list_groupby/3. Return a list of values with each pair being instances of the same class

diff_type_list (?List1, ?List2, ?SameList, ?FinalList)

complement of same_type_list/3. ?SameList is returned from same_type_list/3. Generate a list of all remaining elements.

generate_list(?InstanceX, ?InstanceY, ?List)

Generate a list for check_trans_list/1. All values of X and Y are stored in pairs of [X,Y] that are grouped by their types. Calls class_individual_prop_trans/4, rdf_out_trans_list/5, combine_list_groupby/3 in order.

Consistency_trans_support.pl

find_all_time_trans(?List, ?TimeList)

To be called after find_all_related_start/2. Returns a list of instances of ot:Interval in a list related to the head of ?List.

generate_time_trans_list(?InstanceX, ?InstanceY, ?List)

Generate list of instances of ot:Intervals of ?InstanceX and ?InstanceY and combine into a single nested list ?List via the predicate ?combine_list_groupby/3. ?List is in the form of [[X1,Y1],[X2,Y2]…]. Calls, find_all_related_start/2 find_all_time_trans/2, and combine_list_groupby/3.

trans_time_consistent(?List)

?List is in the form of [[Xt,Yt]|T]. Xt and Yt are instances of Interval. Checks if Xt and Yt are temporal consistent. First check if they are interval consistent, return true if true. If false, then check if one interval is during another. Calls time_interval_equal/2, if true then return true, if false then calls time_during_trans/2.

time_interval_equal(?InstanceX, ?InstanceY)

Checks if time intervals are equal by evaluating the beginning and end (instants) of the intervals. The instants will be evaluated upon temporal unit and datetime parameters (values of year, month, etc.). Calls ot_instant_equal/2 and ot_inst_unit_equal/2

ot_instant_equal(?InstanceX, ?InstanceY)

Modified based on Mark's OT-axiom file where instants are compared based on temporal units. Compares data values of intervals based on datatype properties such as ot:year, ot:month, etc.

ot_inst_unit_equal(?InstanceX, ?InstanceY)

Evaluates if the temporal units of the intervals ?InstanceX and ?InstanceY are the same.

time_during_trans(?instanceX, ?InstanceY)

Checks if time interval X is during Y or vice versa. calls time_during/2

ot_inst_unit (?instance, Unit)

return the temporal unit of an instant

time_interval_unit(?instance, Unit)

return the temporal unit of an interval

generate_city_trans_list(?InstanceX, ?InstanceY, ?List)

generate city instances related to X and Y in a paired list in the form of [[X1,Y1],[X2,Y2]…]. Calls find_all_city_trans/2 and combine_list_groupby/3.

find_all_city_trans(?List, CityList)

Return a list with instances that is related to a City via for_city and located_in property

trans_feature_code(?instanceX, ?InstanceY)

Evaluates if the feature codes of instances are the same. Feature codes are instances of geonames. Two placenames are consistent in terms of admin division if both province or state, and country have the same feature code.

trans_nearby(?instanceX, ?InstanceY)

Checks if instance X is a nearby feature of Y. Evaluates the property geo:nearbyFeature

trans_dynamic(?instanceX, ?InstanceY)

Checks if the effective time interval is the same or time interval consist between the instances.

find_all_unit_trans(?List, UnitList)

Extract values of om:unit_of_measure of all elements in the ?List and store in UnitList. Generate_unit_trans_list (?InstanceX, ?InstanceY, ?List). Generate unit of measure instances related to X and Y in a paired list. Calls find_all_unit_trans/2

unit_multiple_trans(?List)

?List is in the form of [[X,Y],[X2,Y2]…]checks if unit of X is a multiple or submultiple of units of Y and vice versa

unit_consistent_trans(?InstanceX, ?InstanceY)

Check the unit of X and Y. If it's the same instance (rdf_equal/2) then return true. If not, check if the unit of X and Y are related via property P.

Consistency_longitudinal.pl

check_time_city(?InstanceX, ?InstanceY)

check_time_city(?InstanceX, ?InstanceY, City, YearX, YearY)

check time and city. Two instances should have same value for City while different value for Year

check_long_list_start(?InstanceX, ?InstanceY)

Call check_time_city/2 first. Then generate a list and call check_long_list/1

check_long_list(?List)

same as check_trans_list/1 with different output message.

long_dur(?InstanceX,?InstanceY)

Both ?InstanceX and ?InstanceY are instances of ot:Interval. Evaluate if their duration (Dur), which is an instance of ot:DurationDescription, are equal.

Example.pl

Example assertions. We use four sets of instances of 15.2 indicator to test the CICC during development phase.

[15.2_trt_2013, 15.2_trt_2015, 15.2_nyc_2013, 15.2_nyc_2015]

Each instance on its own can be used for definitional consistency evaluation.

The pairs (15.2_trt_2013, 15.2_nyc_2013) and (15.2_trt_2015, 15.2_nyc_2015) can be used for transversal consistency evaluation.

The pairs (15.2_trt_2013, 15.2_trt_2015) and (15.2_nyc_2013, 15.2_nyc_2015) can be used for longitudinal consistency evaluation.

testCase.pl

An error generator that generates the following errors.

- Replace the object of a triple (Sub, Prop, Obj) with a random instance.
- An alternate to this type of error is to change a triple but only with objects from the same class of the original value
- Add or remove an assertion for a subject's property.

Automated test can be done with loop_test/1 as stated in the beginning. Errors can be introduced with manual_test/0.

change_instance(?Instance, ?Property, ?Old, ?New)

change the object of a triple from ?Old to ?New by calling rdf_retractall/3 and rdf_assert/3.

add_property(?Instance1, ?Instance2)

assert a triple (?Instance1, Prop, ?Instance2) where Prop is a property of ?Instance1 generated randomly. ?Instance2 is a randomized instance.

change_property(?Instance1, ?Instance2)

Add or remove a triple wrt a property by calling add_property/2 or rdf_retractall/3 randomly.

remove_property(?instance, ?Property)

remove all values of ?Property from ?instance by calling rdf_retractall/3.

error_generate(?instance1, ?Instance2)

Call change_instance/4 upon a random instance. ?Instance2 is a randomized instance.

error_same_type(?instance1, ?Instance2)

Call change_instance/4 upon a random instance. ?Instance2 is a randomized instance that has the same class as the original object.

class_individual_prop_test(?Instance, ?Class, ?List)

same as class_individual_prop/3.

random_prop_test(?Instance, ?Property)

Randomize a property from ?Instance generated by class_individual_prop_test/3.

class_pcard_prop_test(?Instance, ?Class, ?List)

same as class_pcard_prop /3.

random_prop_card_test(?Instance, ?Property, ?Class)

Randomize a property from ?Instance generated by class_pcard_prop_test /3.

Auto_error()

Return an instance from the KB randomly. Then randomly runs the following predicates: error_generate/2, error_same_type/2, change_property/2. Then repeat.

Manual_error()

A predicate for debugging purpose. It's auto_error/0 without repeat.

create_filename(?Integer, ?String)

generate filenames based on number of runs. E.g. 100 test runs will generate files named as output1.txt, output2.txt….output100.txt

loop_test(?Integer)

Run error generator and CICC for ?Integer times. All outputs are stored in text file.

- Generate filename based on ?Integer.
- Check definitional consistency for all instance, i.e., ['15.2_trt_2013','15.2_nyc_2013','15.2_trt_2015','15.2_nyc_2015']
- Add 1 to ?Integer.
- Reset the prolog database. Then reload regPrefix.pl
- Recursively run with ?Interger +=1.

Appendix III – Prolog Implementation

TC1. Class Type Inconsistency

```
%check if a class C is consistent with definition class Def
check_def_class(C,Def):-
        rdfs_individual_of(C,Def);rdf_equal(C,Def);rdfs_subclass_of(C,Def).
```

TC2. Instance Type Inconsistency

```
%check if instance X is consistent with class Def
check_def (X,Def):-
        %dif/2 predicate omits the type owl:NamedIndividual so only classes will be considered
        rdf(X,rdf:'type',Type), dif(Type,'http://www.w3.org/2002/07/owl#NamedIndividual'),
        %check if classes are equal, evaluate properties if not
        (check_def_class(Type,Def)->lprint(['Class and Definition are equal classes
']);check_def_class_prop(Type,Def)).
```

TC3. Property Inconsistency

```
%Type is the class of instance X
rdf(X,rdf:'type',Type),

%evaluate properties and restrictions between Type and Def
check_def_class_prop(Type, Def):-
        %return a list of properties PList from Def. Where Def is the corresponding definition
class
        class_prop(Def,PList),
        %Evaluate the values of all properties in PList
        check_all_prop(Type,Def,PList).
check_all_prop(Type,Def,[Prop|T]):-
        %return value restriction Val of Def for property Prop
        owl_triples(Def,Prop,Val),
        %return all values of Prop as list NList.
        class_range(Type,Prop,NList),
        %Evaluate the values in NList with respect to Val
```

```
        check_all_value(Type,Prop,Val,NList) ->(tprint([]),tprint([‘Property ’, Prop, ‘is
consistent’]));
        lprint(['Class is T3.PROPERTY INCONSISTENT with Definition']),
        lprint([' - Property restrictions do not satisfy with the definition']),
        !,fail)),
        %call itself recursively until all properties in PList are checked for Type and Def.
        check_all_prop(Type,Def,T),!.
%Base case
check_all_prop(Type,Def,[]).
check_all_value(Type,Prop,Val,[XNext|T]):-
        %return XNext which is the value restriction of Type for Prop
        owl_triples(Type,Prop,XNext),
        %evaluate classes XNext with respect Val
        %evaluate properties if not equal
        %fail if XNext and Val are type inconsistent
        (check_def_class(XNext,Val)->(lprint([]),lprint(['Class and Definition are equal classes
']));
                (tprint([]),tprint(['Class and Definition are NOT equal, Comparing properties ']),
         (check_def_class_prop(XNext,Val)->( tprint([XNext, ' is type consistent with ',Val]));
        lprint(['Class is T1.TYPE INCONSISTENT with Definition']), !,fail)))
        ),
        %call itself recursively until all values of Prop for Type has been checked.
        check_all_value(Type,Prop,Val,T).
%base case
check_all_value(Type,Prop,Val,[]).
```

T1. Non-Overlap Interval Inconsistency

```
%[Xt|T] is a list where all elements are instances temporal entities
%AllList is a list of all instances in the indicator data
time_consistent_noverlap([Xt|T],AllList,Time):-
        findall(X,(member(X,AllList),rdf(X,Prop,Xt)),XList),((
```

```
        (rdf(Xt,ot:'before',Time);ot_interval_before(Xt,Time));
        (rdf(Xt,ot:'after',Time);ot_interval_after(Xt,Time)))->
                time_result_fail(XList,Xt);lprint(['Intervals are T1 consistent ']),
        time_consistent_noverlap(T,AllList,Time).%base case
time_consistent_noverlap ([],AllList,Time).
```

T2. Interval Equality Inconsistency

```
%Evaluates temporal consistency of intervals associated to instances related to indicator
%Two intervals are consistent if they are the same instance or satisfy time_interval_equal
%It is noted if an interval is a subinterval or using different temporal units
time_consistent([Xt|T],AllList,Time):-
        findall(X,(member(X,AllList),rdf(X,Prop,Xt)),XList),((rdf_equal(Xt,Time);time_interval
_equal(Xt,Time))->
                time_result_pass(XList,Xt);
                ((time_subinterval(Xt,Time);time_unit_match(Xt,Time)) -
>time_result_potential(XList,Xt));time_result_fail(XList,Xt)),
        time_consistent(T,AllList,Time).
%base case
time_consistent([],AllList,Time).
```

T3. Subinterval Inconsistency

```
time_subinterval(Xt,Time):-
(time_during(Xt,Time);time_overlap(Xt,Time);time_meets(Xt,Time);time_starts(Xt,Time);time_
ends(Xt,Time)).
time_during(Xt,Time):- rdf(Xt,ot:intervalDuring,Time);ot_interval_during(Xt,Time).
time_overlap(Xt,Time):-rdf(Xt,ot:intervalOverlap,Time).
time_meets(Xt,Time):- rdf(Xt,ot:intervalMeets,Time).
time_starts(Xt,Time):- rdf(Xt,ot:intervalStarts,Time).
time_ends(Xt,Time):- rdf(Xt,ot:intervalEnds,Time).
```

T4. Temporal Granularity Inconsistency

```
%retreive temporal unit of interval X. fails if unit is not equal within X
time_unit(Xt,Unit):- rdf(X, ot:hasEnd, Xend), rdf(X, ot:hasBeginning, Xbeg),
```

```
        rdf(Xbeg, ot:inDateTime, Xbegdt), rdf(Xend, ot:inDateTime, Xenddt),
        rdf(Xbegdt, ot:unitType, Unit), rdf(Xenddt, ot:unitType, Unit).
time_unit_match(Xt,Time):- time_unit(Time,Unit),time_unit(Xt,Unit).
```

G1. Place Equality Inconsistency

```
%Given a list, check all instances in the list has the same value as City
%elements of list are instances of values of for_city or located_in
city_consistent([X|T],City):-
        ((rdf(X,gci:'for_city',City);rdf(X,gci:'located_in',City))->
                lprint([]),lprint([X, '  is consistent with the indicator']);lprint([]),lprint([X, ' is G1
INCONSISTENT with indicator '])),
        city_consistent(T,City).
%base case
city_consistent([],City).
```

G2. SubPlace Inconsistency

```
%Given a list, check all instances are located in the City
city_consistent_sub([X|T],City):-
        (rdf(X,gci:'located_in',PartCity), rdf(PartCity,geo:'locatedIn',City)->
                lprint([]),lprint([X, '  located in area : ',PartCity,' which is within city ',
City]);lprint([]),lprint([X, ' does not refer to a subplace of ', City])),
        city_consistent_sub(T,City).
%base case
city_consistent_sub([],City).
```

G3. Dynamic Place inconsistency

See implementation of G4.

G4. Dynamic Place Temporal Inconsistency

```
%Given a list, check all instances are dynamically City
%instead of just checking if Year are the same instance. Also need to check if two intervals are
consistent with time_interval_equal
city_consistent_dynamic([X|T],City):-
```

```
        lprint(['G3: Dyanmic Place Inconsistency ']),
        (rdf(X,kp:'effective',Year), rdf(City,kp:'effective',Year)->
                lprint([]),lprint([X, ' and ', City, ' are effective in Year : ',Year]);lprint([]),lprint([' City effective year not equal ']));
        city_consistent_dynamic(T,City).

%base case
city_consistent_dynamic([],City).
```

M1. Quantity Measure Inconsistency

```
%check unit consistency between a Quantity and its Measure
unit_qm_consistent([X|T]):-
        rdf(X,om:'value',Measure),rdfs_individual_of(Measure,om:'Measure'),
        rdf(X,om:'unit_of_measure',Unit1),rdf(Measure,om:'unit_of_measure',Unit2),
        ((rdf_equal(Unit1,Unit2))->
                lprint([]),lprint([X, ' and ', Measure, ' Units Match']);lprint([]),lprint([X, ' and ', Measure, ' Units DONT Match ']));
        unit_qm_consistent(T).

%base case
unit_qm_consistent([]).
```

M2. Indicator Unit Component Inconsistency

```
%the list is generated with all instances of om:Quantity related to the indicator
unit_ins_consistent([H|T],Ind):-
(units_ins_match(Ind,P,H,XUnit,YUnit,YSingUnit)->
lprint([]),lprint([H, ' has Unit : ',YUnit,' or singular unit', YSingUnit,' indicator has ',XUnit, ' they are connected with property ',P]);lprint([]),lprint([H, ' Unit instance not equal ']));unit_ins_consistent(T,Ind).
%base case
unit_ins_consistent([],Ind).
```

M3. Singular Unit Inconsistency

```
%find the sigular unit of X and compare with Unit
%the list is generated with instances of om:Unit_multiple_or_submultiple
unit_multiple([X|T],Unit):-
        rdf(X,om:'unit_of_measure',ThisUnit),owl_triples(ThisUnit,om:'singular_unit',SingUnit),
lprint([]),lprint([X, 'has singular Unit ',SingUnit]),
        ((rdf_equal(SingUnit,Unit),lprint([]),lprint(['Singular unit is same as
indicator']));(rdf(Unit,P,SingUnit),
                lprint([]),lprint([X, ' has singular Unit : ',SingUnit,' indicator has ',Unit, ' they are
connected with property ',P]))->lprint([]),lprint(['Unit consistent']);lprint([]),lprint([X, ' Unit
instance not equal ']),
        unit_multiple(T,Unit).

%base case
unit_multiple([],Unit).
```

Trans_TC. Transversal Type Inconsistency

```
check_trans_class(TypeX,TypeY):-
        %class_individual_prop/2 returns properties of class Type as a list PList
        owl_disjoint_class(TypeX,TypeY)->(tprint([]),tprint([TypeX, ' and ', TypeY, ' are disjoint
so fail']),fail);(
        (leaf_node(TypeX);leaf_node(TypeY))->fail;
        class_prop_trans(TypeX,TypeY,PList),
        %check_def_all_prop/3 checks if instance X matches range restriction of Type for each
property in PList
        (check_trans_all_prop(TypeX,TypeY,PList)->(tprint([]),tprint([TypeX, ' all properties
are good with ',TypeY]));
        (lprint([]),
        lprint(['Classes are Transversal TYPE INCONSISTENT (Trans_TC) ']),
        lprint(['       Class 1: ',TypeX]),
        lprint(['       Class 2: ',TypeY]),!,fail))
        ).
```

```
check_trans_all_prop(TypeX,TypeY,[Prop|T]):-
        %owl_triplets can get all properties from both the class and parent classes
        owl_triples(TypeX,Prop,ValX),
        owl_triples(TypeY,Prop,ValY),
        %return all values of a certain property as list NList.
        class_range(TypeX,Prop,NXList),
        class_range(TypeY,Prop,NYList),
        %Check if the value matches the class restriction from the definition
        (check_trans_all_value(TypeX,TypeY,Prop,NXList,NYList)->(tprint([]),tprint([NXList, '
Values are good with ',NYList]));(tprint([]),tprint([NXList, ' one of these Values is TYPE
INCONSISTENT with ',NYList]),!,fail)),
        check_trans_all_prop(TypeX,TypeY,T),!.
%base case
check_trans_all_prop(TypeX,TypeY,[]).

check_trans_all_value(TypeX,TypeY,Prop,[XNext|TX],[YNext|TY]):-
        owl_triples(TypeX,Prop,XNext),
        owl_triples(TypeY,Prop,YNext),
        %check_def_class(XNext,Val),
        (check_def_class(XNext,YNext)->(tprint([]),tprint([XNext, ' and ',YNext, ' is SAME ']));
                (check_trans_class(XNext,YNext)->(lprint([]),lprint([XNext, ' is Transversally
type consistent with ',YNext]));
                (lprint([]),
                lprint(['Classes are Transversally TYPE INCONSISTENT (Trans_TC) ']),
                lprint([' Class 1:', XNext]),
                lprint([' Class 2:', YNext]),!,fail)))
        ),
        tprint([]),tprint(['Step 4: Class ', XNext, ' individual of ', YNext]),
        check_trans_all_value(TypeX,TypeY,Prop,TX,TY).
%base case
check_trans_all_value(TypeX,TypeY,Prop,TX,[]).
```

```
check_trans_all_value(TypeX,TypeY,Prop,[],TY).
```

Trans_G1. Feature Code Inconsistency

```
%-----Trans_G1 Feature Code Inconsistency-------
%evaluates if feature codes are the same between two indicators
city_g1(X,Y,CityX,CityY):-rdf(X,gci:'for_city',CityX),rdf(Y,gci:'for_city',CityY),
        rdf(CityX,'http://www.geonames.org/ontology#featureCode',Code),
        rdf(CityY,'http://www.geonames.org/ontology#featureCode',Code).
city_g1(X,Y,CityX,CityY,CodeX,CodeY):-
rdf(X,gci:'for_city',CityX),rdf(Y,gci:'for_city',CityY),
        rdf(CityX,'http://www.geonames.org/ontology#featureCode',CodeX),
        rdf(CityY,'http://www.geonames.org/ontology#featureCode',CodeY).
city_g1(X,Y):-city_g1(X,Y,_CityX,_CityY,CodeX,CodeY),rdf_equal(CodeX,CodeY).
%check if parent features are of the same feature code
city_g1_parent(X,Y,CityX,CityY):-rdf(X,gci:'for_city',CityX),rdf(Y,gci:'for_city',CityY),
        (rdf(CityX,'http://www.geonames.org/ontology#parentFeature',ParentX);
        rdf(CityX,'http://www.geonames.org/ontology#parentCountry',ParentX);
        rdf(CityX,'http://www.geonames.org/ontology#parentADM1',ParentX)),
        (rdf(CityY,'http://www.geonames.org/ontology#parentFeature',ParentY);
        rdf(CityY,'http://www.geonames.org/ontology#parentCountry',ParentY);
        rdf(CityY,'http://www.geonames.org/ontology#parentADM1',ParentY)),
        rdf(ParentX,'http://www.geonames.org/ontology#featureCode',Code),
        rdf(ParentY,'http://www.geonames.org/ontology#featureCode',Code).
city_g1_parent(X,Y):-city_g1_parent(X,Y,_CityX,_CityY).
```

Long_TC. Longitudinal Type Inconsistency

Same implementation as Trans_TC

Long_T1. Duration Inconsistency

```
%check time and city. Two instances should have same value for City while different value for
Year.
%Also checks duration of the years
check_time_city(X,Y, City, YearX, YearY):-
```

```
        rdf(X, gci:'for_city', City), rdf(Y, gci:'for_city', City),
        rdf(X, gci:'for_time_interval', YearX), rdf(Y, gci:'for_time_interval', YearY),
        YearX \= YearY,long_dur(YearX,YearY).
%check if two intervals have the same instance of ot:DurationDescription
long_dur(Time1,Time2):-
rdf(Time1,ot:hasDurationDescription,Dur),rdf(Time2,ot:hasDurationDescription,Dur).
```