International Standard

ISO/IEC 5087-2

First edition
2024-04

Information technology — City data model —

Part 2: City level concepts

Reference number
ISO/IEC 5087-2:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

A list of all parts in the ISO/IEC 5087 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The intended audience for this document includes municipal information systems departments, municipal software designers and developers, and organizations that design and develop software for municipalities.

Cities today face a challenge of how to integrate data from multiple, unrelated sources where the semantics of the data are imprecise, ambiguous and overlapping. This is especially true in a world where more and more data of interest are being openly published by various organizations. A morass of data is increasingly becoming available to support city planning and operations activities. In order to be used effectively, it is necessary for the data to be unambiguously understood so that it can be correctly combined, avoiding data silos. Early successes in data "mash-ups" relied upon an independence assumption, where unrelated data sources were linked based solely on geospatial location, or a unique identifier for a person or organization. More sophisticated analytics projects that require the combination of datasets with overlapping semantics entail a significantly greater effort to transform data into something useable. It has become increasingly clear that integrating separate datasets for this sort of analysis requires an attention to the semantics of the underlying attributes and their values.

A common data model enables city software applications to share information, plan, coordinate and execute city tasks, and support decision making within and across city services, by providing a precise, unambiguous representation of information and knowledge commonly shared across city services. This requires a clear understanding of the terms used in defining the data, as well as how they relate to one another. This requirement goes beyond syntactic integration (e.g. common data types and protocols), it requires semantic integration: a consistent, shared understanding of the meaning of information.

To motivate the need for a standard city data model, consider the evolution of cities. Cities deliver physical and social services that have traditionally operated as silos. If during the process of becoming smarter, transportation, social services, utilities, etc. were to develop their own data models, the result would be smarter silos. To create truly smart cities, data need to be shared across these silos. This can only be accomplished through the use of a common data model. For example, "Household" is a category of data that is commonly used by city services. Members of Households are the source of transportation, housing, education and recreation demand. This category represents who occupies a home, their age, their occupations, where they work, their abilities, etc. Though each city service can potentially gather and/or use different aspects of a Household, much of the data needs to be shared.

Supporting this interoperability among city datasets is particularly challenging due to the diversity of the domain and the heterogeneity of its data sources. The purpose of this document is to support the precise and unambiguous specification of city data using the technology of ontologies[1],[2] as implemented in the Semantic Web.[3] By doing so it will:

— enable the computer representation of precise definitions, thereby reducing the ambiguity of interpretation;

— remove the independence assumption, thereby allowing the world of Big Data, open-source software, mobile apps, etc. to be applied for more sophisticated analysis;

— achieve semantic interoperability, namely the ability to access, understand, merge and use data available from datasets spread across the Semantic Web;

— enable the publishing of city data using Semantic Web and ontology standards; and

— enable the automated detection of city data inconsistency, and the root causes of variations.

With a clear semantics for the terminology, it is possible to perform consistency analysis, and thereby validate the correct use of the document.

Figure 1 identifies the three levels of the ISO/IEC 5087 series. The lowest level, defined in ISO/IEC 5087-1, provides the classes, properties and logical computational definitions for representing the concepts that are foundational to representing any data. The middle level, defined in ISO/IEC 5087-2 (this document), provides the classes, properties and logical, computational definitions for representing urban-specific concepts common to all city services but not specific to any service. The top level provides the classes, properties and

logical, computational definitions for representing service specific concepts that are used by other services across the city. For example, ISO/IEC TS 5087-3:—[1)] is intended to define the transportation concepts. In the future, additional parts will be added to the ISO/IEC 5087 series covering services such as education, water, sanitation, energy, etc.

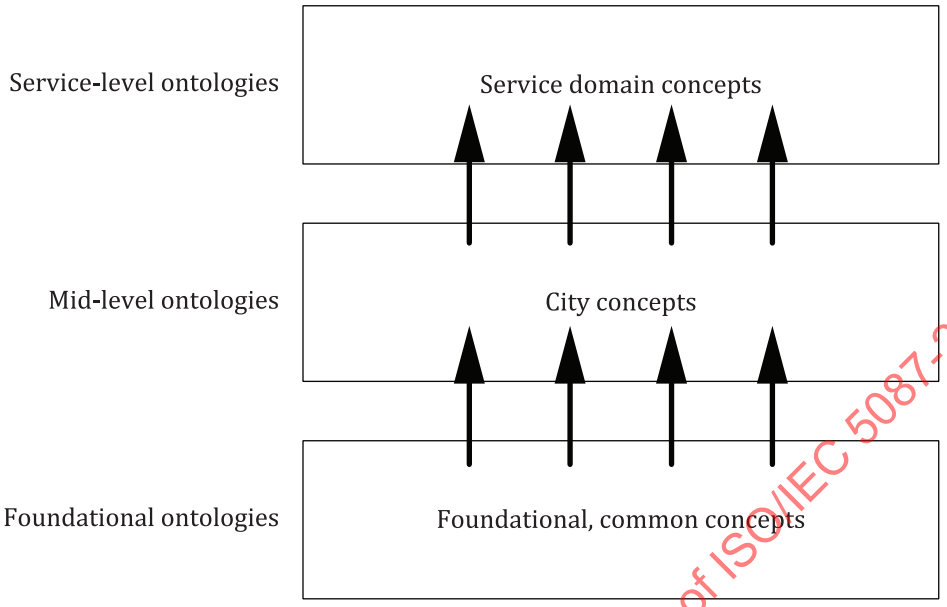

Figure 1 — Stratification of city data model

Figure 2 depicts example concepts for the three levels.

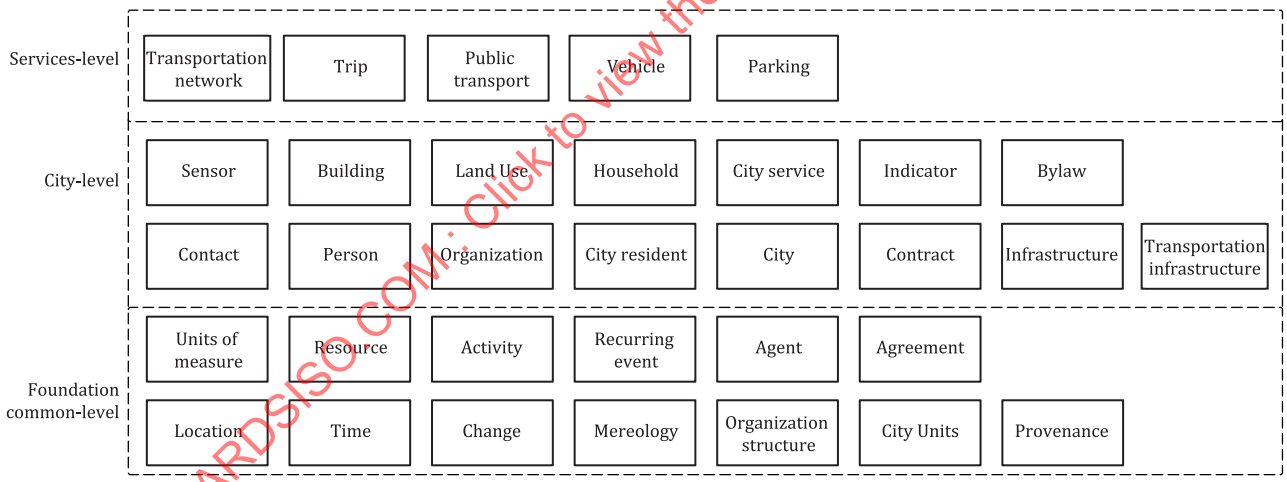

Figure 2 — Example concepts for each level

There are other existing standards that overlap conceptually with some of the terms presented in this document. The relationship between this document and existing standards that address similar or related concepts is identified in Annex A.

1) Under preparation. Stage at the time of publication: ISO/IEC AWI TS 5087-3:2024.

Information technology — City data model —

Part 2:
City level concepts

1 Scope

This document defines an ontology for city-level concepts defined using terms specified in ISO/IEC 5087-1. City-level concepts are used to represent data that is shared across multiple services and stakeholders in the city. City-level concepts are distinguished by their data being read and updated by multiple city services and stakeholders.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

SEMANTIC SENSOR NETWORK ONTOLOGY, W3C Recommendation 19 October 2017, https://www.w3.org/TR/vocab-ssn/

ISO/IEC 5087-1, *Information technology — City data model — Part 1: Foundation level concepts*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
cardinality
number of elements in a set

[SOURCE: ISO/TS 21526:2019, 3.11]

3.2
description logic
DL
family of formal knowledge representation languages that are more expressive than propositional logic but less expressive than first-order logic

[SOURCE: ISO/IEC 21972:2020, 3.2]

3.3
manchester syntax
compact, human readable syntax for expressing Description Logic descriptions

[SOURCE: https://www.w3.org/TR/owl2-manchester-syntax/ (Copyright © 2012. World Wide Web Consortium. https://www.w3.org/Consortium/Legal/2023/doc-license).]

3.4
measure
value of the measurement (via the numerical_value property) which is linked to both Quantity and Unit_of_measure

[SOURCE: ISO/IEC 21972:2020, 3.4]

3.5
namespace
collection of names, identified by a URI reference, that are used in XML documents as element names and attribute names

Note 1 to entry: Names may also be identified by an IRI reference.

[SOURCE: ISO/IEC 21972:2020, 3.5, modified — Note 1 to entry has been added.]

3.6
ontology
formal representation of phenomena of a universe of discourse with an underlying vocabulary including definitions and axioms that make the intended meaning explicit and describe phenomena and their interrelationships

[SOURCE: ISO 19101-1:2014, 4.1.26]

3.7
ontology web language
ontology language for the Semantic Web with formally defined meaning

Note 1 to entry: OWL 2 ontologies provide classes, properties, individuals and data values and are stored as Semantic Web documents.

[SOURCE: ISO/IEC 21972:2020, 3.7, modified — The preferred term "OWL 2 Web Ontology Language" has been replaced by the preferred term "ontology web language".]

3.8
quantity
property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference

Note 1 to entry: Quantities can appear as base quantities or derived quantities.

EXAMPLE 1 Length, mass, electric current (ISQ base quantities).

EXAMPLE 2 Plane angle, force, power (derived quantities).

[SOURCE: ISO/IEC Guide 99:2007, 1.1, modified — NOTEs 1 to 6 have been removed; new Note 1 to entry and two EXAMPLEs have been added.]

3.9
Semantic Web
W3C's vision of the Web of linked data

Note 1 to entry: Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data.

[SOURCE: https://www.w3.org/standards/semanticweb/ (Copyright © 2015. World Wide Web Consortium. https://www.w3.org/Consortium/Legal/2023/doc-license).]

3.10
unit_of_measure
definite magnitude of a quantity, defined and adopted by convention and/or by law

[SOURCE: ISO/IEC 21972:2020, 3.9]

4 Abbreviated terms and namespace prefixes

AAFC	Agriculture and Agri-Foods Canada
CGRM	Canadian Government Reference Mode
CLUMP	Canada Land Use Monitoring Program
DL	description logic
IPCC	International Panel on Climate Change
IRI	international resource identifier
LBCS	Land Based Classification Standards
OWL	ontology web language
RDF	resource description framework
RDFS	resource description framework schema
UML	Unified Modelling Language

The following namespace prefixes are used in this document:

— activity: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/Activity/

— agent: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/Agent/

— agreement: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/Agreement/

— building: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Building/

— bylaw: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Bylaw/

— city: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/City/

— cityresident: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/CityResident/

— cityunits: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/CityUnits/

— code: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Code/

— contact: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Contact/

— contract: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Contract/

— genprop: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/GenericProperties/

— geo: http://www.opengis.net/ont/geosparql#

— i72: http://ontology.eil.utoronto.ca/ISO21972/iso21972/

— infras: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Infrastructure/

— landuse: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Landuse/

— org: http://www.w3c.org/ns/org#

— org_s: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/OrganizationStructure/

— org_city: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Organization/

— owl: https://www.w3.org/2002/07/owl#

— partwhole: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/Mereology/

— person: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Person/

— rdf: https://www.w3.org/1999/02/22-rdf-syntax-ns#

— rdfs: https://www.w3.org/2000/01/rdf-schema#

— recurringevent: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/RecurringEvent/

— resource: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/Resource/

— loc: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/SpatialLoc/

— service: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/CityService/

— transinfras: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/TransportationInfrastructure/

— time: https://www.w3.org/2006/time#

— xsd: https://www.w3.org/2001/XMLSchema#

The formalization of the classes in this document is specified using the following table format, which is a simplification of description logic (DL) where the first column identifies the class name, the second column its properties and the third column each property's range restriction It shall be read as: The <Class> is a subClassOf the conjunction of the associated <property>s with their <value>s. Range restrictions are specified using the Manchester syntax. For example, Table 1 specifies that Agent is an rdfs:subClassOf the intersection of (Person or Organization) and org:memberOf only Organization.

Table 1 — Example formalization of the Agent class

Class	Property	Value Restriction
Agent	rdfs:subClassOf	Person or org_s:Organization
	org_s:memberOf	only Organization
	individual	{joe, frank}

CamelCase is used for specifying classes, properties and instances. For example, "legalName" instead of "legal_name". The first letter of a class name is capitalized. The first letter of a property and instance name are not capitalized. An instance of a class shall satisfy the class's definition. The instance's properties and values shall satisfy the value restrictions of the class it is an instance of.

The formalization of the properties in this document is done similarly, using the following table format that allows for the identification of properties and their sub-properties, inverse properties, or other characteristics. It is to be read as: The <property> is <characteristic> of <value>, or simply the <property> is <characteristic> if no value is applicable. For example, in Table 2 hasPrivilege is a sub-property of the agentInvolvedIn property. Characteristics are specified using the Manchester syntax.

Table 2 — Example property formalization

Property	Characteristic	Value (if applicable)
hasPrivilege	rdfs:subPropertyOf	agentInvolvedIn
	Irreflexive	

In the case of DL definitions of classes where the simplified table representation is insufficient, the DL specification will be supplied as an addition to the content in the table.

The patterns defined in this document have also been implemented in OWL and made available online. The location of these encodings is identified in Annex C.

5 Unique identifiers

All classes, properties and instances of classes have a unique identifier that conforms to Linked Data/Semantic Web standards. The unique identifier is an IRI. When using ISO/IEC 5087-2 (this document) in an

application, a class is identified by the IRI for the pattern of which it is a member, followed by the class name. In the Agent example in Clause 4, the Agent class's unique identifier would be:

https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Agent/Agent

Breaking the IRI down:

— "5087" identifies the series number

— "-2" identifies the part number

— "ed-1" indicates that the class is defined in edition 1 of the document

— "en" indicates that the class is defined in a pattern implemented in English

— The first "Agent" identifies the Agent Pattern

— The second "Agent" identifies the Agent class within the Agent Pattern

The IRI can be shortened using the prefix's defined in Clause 4:

agent:Agent

where agent: is the prefix for the Agent Pattern.

Properties are identified in the same manner. The IRI's of individuals created by an application of ISO/IEC 5087-2 would have IRIs unique to the application.

6 City-level ontologies

6.1 General

Much as the foundational concepts defined in ISO/IEC 5087-1 are common across arbitrary domains, concepts also exist that are generic across all cities. These concepts define the domain upon which city services operate. They are common for all cities but not specific to any one service. The city data model defines sixteen city-level ontologies, or patterns, to capture these concepts. These are described in the following subclauses. The content of these ontologies defines terms needed to capture these concepts at a generic level. They are not intended to be exhaustive, nor to capture the variety of classification systems and taxonomies that are potentially of interest for different user groups. The common terms defined here provide the foundational, common language that may be extended by different users as required.

The patterns presented here shall conform to the foundational concepts defined in ISO/IEC 5087-1. Specific references to foundational content defined in ISO/IEC 5087-1 are identified in text descriptions of pattern imports, as well as through the explicit identification of terms from ISO/IEC 5087-1.

The cardinality restrictions specified in the pattern formalizations are weakened intentionally in many cases to support the pragmatic implementation of the ISO/IEC 5087 series. For example, while common sense dictates that all persons should have at exactly one legal name, in practice there are many possible applications of city data where persons may be represented with no name at all. Users of this document are encouraged to extend and strengthen these restrictions where possible or warranted by the application.

6.2 Code pattern

6.2.1 General

The Code pattern provides a structure to address the challenge of value enumeration with a general approach. In city data there are many classes of things that are intended to be instantiated using a set list of values (e.g. classification systems). However, these values can change based on application or context. In such cases it is not desirable for a standard to prescribe a restricted set of possible values which will potentially not satisfy the needs of all applications. On the other hand, leaving the values completely open-

ended provides no utility for interoperability. The Code Pattern provides an intermediate solution for this challenge by introducing a generic set of classes and properties that can be used to extend such classes to define various classification systems in an integrated way.

Instead of enumerating value sets for classes in this document, values can be defined with an associated Code that specifies additional metadata about the value and its origins. This allows these classes to be extended with various value-systems as required by a particular application, while providing the necessary information to support interpretation and integration as needed.

6.2.2 Key classes and properties

The key classes and properties are formalized in Table 3 and Table 4, respectively. A code is introduced to capture the possible value of an object, according to a predefined system of values. It has the following key properties:

— **definedBy**: identifies the Organization that defined the code.

— **specification**: specifies a URI where the definition of the code can be found.

— **hasIdentifier**: identifies a unique identifier for the code.

— **genprop:hasName**: specifies a name or title for the code.

— **genprop:hasDescription**: specifies a description of the code.

6.2.3 Formalization

Table 3 — Key classes in the Code pattern

Class	Property	Value restriction
Code	definedBy	max 1 org_s:Organization
	specification	only xsd:string
	genprop:hasIdentifier	max 1 xsd:string
	genprop:hasName	only xsd:string
	genprop:hasDescription	only xsd:string

Table 4 — Key properties in the Code pattern

Property	Characteristic	Value (if applicable)
hasCode	rdfs:range	Code

6.3 Infrastructure pattern

6.3.1 General

The Infrastructure pattern defines the concepts needed to capture various types of city infrastructure, such as buildings and roads. The Infrastructure pattern reuses the Spatial Location pattern (from ISO/IEC 5087-1) in order to capture the location of these Infrastructure Elements.

It is extended by the Building pattern, and the Transportation Infrastructure pattern. It can be extended with other types of infrastructure as required.

6.3.2 Key classes and properties

The key classes are formalized in Table 5. An Infrastructure Element is a generic representation of a city structure of interest. All Infrastructure Elements may have a defined, where locations are spatial geometries as defined in ISO/IEC 5087-1. The Mereology pattern (from ISO/IEC 5087-1) is also reused in order to support

the possible representation of infrastructure parts (e.g. road segments) and their associated wholes (e.g. the entire road). The following are its properties:

— **loc:hasLocation**: specifies the location of the element as a loc:Location.

— **partwhole:hasProperPart**: specifies any sub-parts of the element.

— **genprop:hasIdentifier**: specifies identifiers for the element provided by the city.

— **genprop:hasName**: specifies names of the element.

— **genprop:hasDescription**: specifies descriptions of the element.

— **contact:hasAddress**: specifies the address of the element as a contact:Address.

— **i72:hasValue**: identifies the value of a Building as a cityunits:MonetaryValue. Distinctions may be made between different types of value, such as the government-assessed value of a building or the purchase price. Subproperties of hasValue may be introduced to distinguish these types as required. Suggested possible extensions include: hasCost (purchase price), hasGovernmentAssessedValue (government assessed value for tax purposes), hasCollateralValue (value assessed in relation to a loan), hasInsuredValue (value determined by insurance policy).

6.3.3 Formalization

Table 5 — Key classes in the Infrastructure pattern

Class	Property	Value restriction
InfrastructureElement	loc:hasLocation	only loc:Location
	partwhole:hasProperPart	only InfrastructureElement
	genprop:hasIdentifier	only xsd:string
	genprop:hasName	only xsd:string
	genprop:hasDescription	only xsd:string
	contact:hasAddress	only contact:Address
	i72:hasValue	only i72:MonetaryValue

6.4 Transportation Infrastructure pattern

6.4.1 General

The Transportation Infrastructure pattern defines the concepts that are relevant in describing the physical transportation infrastructure and their characteristics. This includes the concepts of a Road, Bridge and Tunnel. The Infrastructure pattern is reused here, as these transportation structures are all defined as types of (subclasses) Infrastructure Elements.

6.4.2 Key classes and properties

The key classes are formalized in Table 6. They include the common structural elements that comprise the physical (land) transportation infrastructure: Roads, Rail Lines, Bridges and Tunnels.

A Travelled Way is a type of Infrastructure Element that enables travel. It is defined with the following property:

— **aggregationOf**: identifies a Travelled Way Link that is aggregated to form the Travelled Way. aggregationOf is distinct from parthood in that a Travelled Way Link does not depend on the Travelled Way in order to exist.

A Travelled Way Link represents a (continuous) length of a Travelled Way. The start and end of a Travelled Way is typically identified according to operational or managerial significance. It has the following property:

— **aggregateOf**: identifies a Travelled Way that aggregates the Travelled Way Link.

Travelled Way Links can be decomposed into Travelled Way Segments. Travelled Way Segments are typically defined based on some physical characteristics of the infrastructure (e.g. lane additions/removals, intersections). A Travelled Way Segment has the following property:

— **partwhole:properPartOf**: identifies the Travelled Way Link that it is part of.

A Road is a type of Travelled Way. It describes a part of the physical transportation infrastructure that has been improved to allow travel by motor vehicles, persons, bicycles, and similar methods of conveyance. It is identified as a Road as such by a given governing body. It is defined with the following property:

— **aggregationOf**: identifies the Road Link that a Road may be decomposed into in order to represent its lengths at a finer granularity.

A Road Link is a type of Travelled Way Link that is represents a length of a Road. It has the following property:

— **aggregateOf**: identifies the Road that aggregates the Road Link.

A Road Segment is a type of Travelled Way Segment that represents a part of a Road Link. It has the following property:

— **partwhole:properPartOf**: identifies the Road Link that the Road Segment is a part of.

— **networkType**: identifies the RoadNetworkType of the Road Segment. The Road Network Type identifies the modes of travel permitted or otherwise supported on a particular Road Segment (e.g. bicycles, motorized vehicles). Its values may be defined more precisely with the use of the code:hasCode property. The network types for Road Links and Roads may be defined based on the Road Segments they contain.

A Rail Line is a type of Travelled Way. It describes a part of the physical transportation infrastructure that has been fitted with tracks to allow travel by trains and other sorts of rail vehicles. No distinction is made between Rail Line types at this level. A Rail Line is identified as such by a governing body. It is defined with the following property:

— **aggregationOf**: identifies the Rail Link that a Rail may be decomposed into in order to represent its lengths at a finer granularity.

A Rail Link is a type of Travelled Way Link that represents a length of a Rail Line. It has the following property:

— **aggregateOf**: identifies the Rail Line that the Rail Link is an aggregate of.

A Rail Segment is a type of Travelled Way Segment that represents part of a Rail Link. It has the following property:

— **partwhole:properPartOf**: identifies the Rail Link that the Rail Segment is a part of.

A Bridge is a type of Infrastructure Element. It describes a part of the physical transportation infrastructure that enables travel over a given area. It may contain some Road Segments or Rail Line Segments. A Bridge is identified as such by a governing body. It is defined with the following properties:

— **partwhole:hasProperPart:** identifies the Bridge Segments that a Bridge may be decomposed into to represent its lengths at a finer granularity.

— **supports:** identifies Travelled Way Segments that are supported by (i.e. travel over) the Bridge, if any.

A Bridge Segment is another type of Infrastructure Element. It is part of a Bridge and is defined with the following property:

— **supports:** identifies Travelled Way Segments that are supported by (i.e. travel over) the Bridge Segment, if any.

A Tunnel is a type of Infrastructure Element. It describes a part of the physical transportation infrastructure that enables travel underneath a given area. It may contain Road or Rail Line segments. A Tunnel is identified as such by a governing body. It is defined with the following properties:

— **partwhole:hasProperPart:** identifies the Tunnel Segments that a Tunnel may be decomposed into to represent its lengths at a finer granularity.

— **supports:** identifies Travelled Way Segments that are supported by (i.e. travel through) the Tunnel, if any.

A Tunnel Segment is another type of Infrastructure Element, it is part of a Tunnel and is defined with the following property:

— **supports:** identifies Travelled Way Segments that are supported by (i.e. travel over) the Tunnel Segment, if any.

6.4.3 Formalization

Table 6 — Key classes in the Transportation Infrastructure pattern

Class	Property	Value restriction
TravelledWay	rdfs:subClassOf	infras:InfrastructureElement
	aggregationOf	only TravelledWayLink
TravelledWayLink	rdfs:subClassOf	infras:InfrastructureElement
	aggregateOf	only TravelledWay
TravelledWaySegment	rdfs:subClassOf	infras:InfrastructureElement
	partwhole:properPartOf	some TravelledWayLink
Road	rdfs:subClassOf	TravelledWay
	aggregationOf	only RoadLink
RoadLink	rdfs:subClassOf	TravelledWayLink
	aggregateOf	only Road
RoadSegment	rdfs:subClassOf	TravelledWaySegment
	partwhole:properPartOf	some RoadLink
	networkType	only RoadNetworkType
RoadNetworkType	hasCode	code:Code
RailLine	rdfs:subClassOf	TravelledWay
	aggregationOf	only RailLink
RailLink	rdfs:subClassOf	TravelledWayLink
	aggregateOf	only RailLine
RailSegment	rdfs:subClassOf	TravelledWaySegment
	partwhole:properPartOf	some RailLink
Bridge	rdfs:subclassOf	InfrastructureElement
	partwhole:hasProperPart	only BridgeSegment
	supports	only TravelledWaySegment
BridgeSegment	rdfs:subClassOf	InfrastructureElement
	supports	only TravelledWaySegment
Tunnel	rdfs:subClassOf	InfrastructureElement
	partwhole:hasProperPart	TunnelSegment
	supports	only TravelledWaySegment
TunnelSegment	rdfs:subClassOf	InfrastructureElement
	supports	only TravelledWaySegment

6.5 Building pattern

6.5.1 General

The Building pattern defines the concepts to capture information about individual buildings, thus describing land use from a different perspective and at a finer level of granularity than typical land use classifications. The Infrastructure pattern is reused here, as Buildings and Building Units are defined as types of (subclasses) Infrastructure Elements. The Building pattern also reuses the Spatial Location pattern in order to capture the location of a building. Other attributes of a building are also captured, such as the type of building, units contained in a building, their monetary value, etc. The Mereology pattern from ISO/IEC 5087-1 is referenced to capture the disaggregate parts of a building, and the City Units pattern from ISO/IEC 5087-1 is referenced to capture the quantities and units of attributes required for land value considerations, such as sale prices and area.

6.5.2 Key classes and properties

The key classes and properties are formalized in Table 7 and Table 8, respectively. A Building is a structure with a roof and walls. It is defined as a type of Infrastructure Element and has the following key properties:

— **buildingFacility**: identifies a Facility(s), e.g. kitchen, bath, or air conditioning that is contained in the Building.

— **hasBuildingUnit**: identifies the Building Unit(s), if any, that are contained by the Building.

— **numBuildingUnits**: identifies the total number of Building Unit(s) in a Building as a (non-negative) integer value.

— **hasBuildingFootprintArea**: identifies the footprint occupied by the Building as a cityunits:Area quantity.

— **hasBuildingFloorArea**: identifies the floor area occupied by the Building as a cityunits:Area quantity. The floor area accounts for the area of each floor of the building. However, floor area excludes unoccupied areas such as basements.

— **floorAreaRatio**: identifies the building's floor area ratio as a cityunits:RatioIndicator quantity. This is a ratio of the gross floor area (the value of hasBuildingFloorArea) to its "buildable area" (i.e. lot size).

— **hasBuildingHeight**: identifies the Building's height as a cityunits:Length quantity.

— **numFloors**: identifies the number of floors in a Building as a (non-negative) integer value. This represents a total count of all floors in the building (including those below ground).

— **numAboveGroundFloors**: identifies the number of above ground floors in a Building a (non-negative) integer value.

— **windowToWallRatio**: identifies the percentage area of a Building's exterior envelope that is made up of glazing (i.e. glass installed in fixed openings such as windows and doors). The value is specified as a cityunits:RatioIndicator quantity.

— **builtAccordingToConstructionCode**: identifies the name of any construction code(s) applicable during the construction of the Building. Construction codes refer to a set of rules or instructions that specify the standards for constructed objects such as buildings and nonbuilding structures. Buildings are required to conform to the code throughout the construction of the building.

— **use**: identifies the use of the Building as a BuildingUse object(s), typically associated with a particular classification system.

— **hasConstructionStatus**: identifies the Construction Status of a Building.

— **yearOfConstruction**: identifies the year that construction on the Building was completed as a time: DateTimeDescription object.

— **propertyRegistrationID**: identifies the unique identifier of the real estate in the property register of authority where the Building is located.

Construction Status identifies the construction status of a building more precisely. For example, distinctions can be made between new construction and renovation, but also on-site vs. off-site (prefabrication) construction. This class is intended to be extended based on existing classification systems. Its values may be defined more precisely with the use of the code:hasCode property.

Building Use identifies the use of a building, e.g. based on occupancy (business, treatment, residential). This class is intended to be extended according to one or more existing classification systems. Its values may be defined more precisely with the use of the code:hasCode property.

A BuildingUnit refers to a part of a Building that is physically separate (i.e. has its own entrance). It should have its own identifier within the building and has a number of other characteristics that may be defined with the following properties:

— **unitInBuilding**: identifies the Building that the Building Unit belongs to (is contained in).

— **loc:hasLocation**: identifies the individual location of the Building Unit, as defined in the Location pattern.

— **contact:hasAddress**: identifies the address of the Building Unit, as defined in the Contact pattern.

— **hasRent**: identifies the rental fee for the Building Unit, if any.

— **hasUnitSize**: identifies the size of the Building Unit as an Area, defined in the City Units pattern.

— **numberOfRooms**: identifies the number of rooms in the Building Unit.

— **numberOfBedrooms**: identifies the number of bedrooms in the Building Unit.

— **floorToCeilingHeight**: identifies the floor to ceiling height in the Building Unit as a Length, defined in the City Units pattern.

— **buildingFacility**: identifies any Facility(s) contained in the Building Unit. This is distinct from any Facilities that the owners of the unit would have access to (e.g. provided by the building to its occupants) and includes only those that are part of the Building Unit.

Different types (subclasses) of Building such as House, Apartment Building, or Office Building may be defined as required. It is recommended to avoid confusing type of building structure with building use. For example, a “Detached House” is a type of building whereas an “Office” is not. A Building also requires some degree of permanence; a “Duplex” or a “Garage” may be defined as types of buildings, whereas a tent may not. Note also that a Building refers only to the structure, not the surrounding area; an airport terminal is a building, an aircraft hangar is a building, but the entire airport complex is not.

6.5.3 Formalization

Table 7 — Key classes in the Building pattern

Class	Property	Value restriction
Building	rdfs:subClassOf	infras:InfrastructureElement
	buildingFacility	only Facility
	hasBuildingUnit	only BuildingUnit
	hasBuildingFootprintArea	only cityunits:Area
	hasBuildingFloorArea	only cityunits:Area
	hasBuildingHeight	only cityunits:Length
	numFloors	max 1 xsd:nonNegativeInteger
	use	only BuildingUse
	hasConstructionStatus	max 1 ConstructionStatus
	yearOfConstruction	max 1 time:DateTimeDescription and time:day exactly 0 rdfs:Literal and time:month exactly 0 rdfs:Literal and time:year exactly 1 rdfs:Literal
	numBuildingUnits	max 1 xsd:nonNegativeInteger
	propertyRegistrationID	max 1 rdfs:Literal
	floorAreaRatio	max 1 i72:RatioIndicator and (i72:hasDenominator only i72:Area) and (i72:hasNumerator only i72:Area)
	numAboveGroundFloors	max 1 xsd:nonNegativeInteger
	windowToWallRatio	max 1 i72:RatioIndicator and (i72:denominator only i72:Area) and (i72:numerator only i72:Area)
	builtAccordingToConstructionCode	only xsd:string
ConstructionStatus	code:hasCode	only code:Code
Facility	loc:hasLocation	max 1 loc:Location
BuildingUnit	rdfs:subClassOf	infras:InfrastructureElement
	unitInBuilding	max 1 Building
	loc:hasLocation	Only loc:Location
	contact:hasAddress	max 1 contact:Address
	hasRent	only cityunits:MonetaryValue
	hasUnitSize	only cityunits:Area
	numberOfRooms	max 1 xsd:nonNegativeInteger
	numberOfBedrooms	max 1 xsd:nonNegativeInteger
	floorToCeilingHeight	only cityunits:Length
	buildingFacility	only Facility
BuildingUse	code:hasCode	only code:Code

Table 8 — Key properties in the Building pattern

Property	Characteristic	Value (if applicable)
hasBuildingFacility	subPropertyOf	partwhole:hasComponent
hasBuildingUnit	inverseOf	unitInBuilding
	subPropertyOf	partwhole:hasComponent
unitInBuilding	inverseOf	hasBuildingUnit
	subPropertyOf	partwhole:componentOf

6.6 Land Use pattern

6.6.1 General

The Land Use pattern provides the necessary concepts to describe a particular classification(s) applied to some area of land. It introduces the generic concept of a LandUseClassification which may be extended to capture specific classification systems as required. Annex B illustrates possible extensions with classifications from Land-Based Classification Standards (LBCS), Canada Land Use Monitoring Plan (CLUMP), and Agriculture and Agri-Food Canada (AAFC). Such extensions could be used to define multiple such systems such that relationships between classifications in different systems can be inferred. The Land Use Pattern imports the Spatial Location Pattern (defined in ISO/IEC 5087-1); in particular, Land Areas are defined as a type of Location, which may be described as a geometry. They may also be related to other Land Areas (or arbitrary Locations) by the spatial relations such as containment, contact, overlaps, etc.

6.6.2 Key classes and properties

The key classes are formalized in Table 9. A Land Area is a way of defining some area in an urban system. It has the following key properties:

— **loc:hasLocation**: identifies the geospatial representation of a Land Area's location.

— **landUse**: identifies a Land Use Classification that is identified for the Land Area.

— **hasArea**: identifies the size of a Land Area as cityunits:Area quantity.

— **hasPopulation**: identifies the population of a Land Area as an i72:Population quantity.

Land Use Classifications provide a means of describing the land use in a standard way. The Land Use Classification class is intended to serve as the root class that may be extended with various classification systems as required for a particular application. Its values may be defined more precisely with the use of the code:hasCode property.

6.6.3 Formalization

Table 9 — Key classes in the Land Use pattern

Class	Property	Value restriction
LandArea	loc:hasLocation	loc:Location
	landUse	min 1 LandUseClassification
	hasArea	only cityunits:Area
	hasPopulation	only cityunits:Population
LandUseClassification	code:hasCode	only code:Code

6.7 Person pattern

6.7.1 General

Persons are a key category in city models. They are the focus of many city services, and it is the combination of decisions of persons in the population that result in changed characteristics of the city, such as road congestion. For example, a person's decision to change places of employment will likely impact their daily travel behaviour. The Person pattern enables the representation of persons and their attributes of interest. Factors such as a person's age, income and place of residence are defined as properties of a person.

6.7.2 Key classes and properties

The key classes and properties are formalized in Table 10 and Table 11, respectively. The core class is Person, which is defined as a type (subclass) of agent:Agent and has the following properties:

— **hasPersonID**: identifies any instances of a Person's PersonId. Examples include a driver's licence or passport.

— **name**: identifies the instance of PersonName legally associated with a Person.

— **alias**: identifies any additional instances of PersonName associated with a Person. A Person may have one legal name, but multiple alias names.

— **contact:hasAddress**: identifies any instances of a contact:Address associated with the Person.

— **contact:hasPhoneNumber**: identifies any instances of PhoneNumbers associated with the Person.

— **hasEmail**: specifies zero or more email addresses as xsd:string values.

— **birthdate**: identifies the time:Instant when the Person was born.

— **birthplace**: identifies the contact:Address where the Person was born.

— **deathdate**: identifies the time:Instant when the Person died.

— **deathplace**: identifies the contact:Address where the Person died.

— **citizenOf**: specifies one or more Citizenships, each specifying the country (Country) and time interval (time:ProperInterval) the Person is a citizen. A Person can be a citizen of more than one country and for different time intervals. It is recommended that a Country be defined with using ISO 3166-2[4] alpha-2 2 letter country code, however different systems may be accommodated with the Code pattern.

— **parent**: identifies any parents of the Person. Note that we define the parent relation in a general sense (including either the legal or biological relation). This property may be specialized and restricted, for example hasBiologicalMother: exactly 1 Person.

— **spouse**: identifies any spouses of the Person.

— **children**: identifies any children of the Person.

— **sex**: identifies a Person's sex. A Person is associated with at most one sex. The definition of sex is distinct from that of a person's gender. As distinguished by Statistics Canada[5] "Sex refers to sex assigned at birth. Sex is typically assigned based on a person's reproductive system and other physical characteristics." Its values may be defined more precisely with the use of the code:hasCode property.

— **genderIdentity**: identifies a Person's associated Gender identity. The value of this may differ from a person's sex at birth, or it may be the same. Precisely how Gender is defined and instantiated varies based upon context and so may be defined by the user of this standard as appropriate. Its values may be defined more precisely with the use of the code:hasCode property.

— **income**: specifies the Person's annual income.

— **hasSkill**: identifies the Skills the Person has.

— **hasEducation**: identifies the Education the Person has.

Skill and Education classes are not defined and are left to the application to define. Their values may be defined more precisely with the use of the code:hasCode property.

PersonName represents the parts of a person's name. It has the following properties:

— **givenName**: identifies a string that is the given or first name of the Person.

— **additionalName**: identifies a string that is the middle or additional names of the Person.

— **familyName**: identifies as single string that is the family or last name of the Person.

PersonID represent a unique identifier for the Person. It has the following properties:

— **genprop:hasIdentifier**: identifies a string that encodes the unique identifier of the Person.

— **hasIDType**: specifies the type of identifier, its values may be defined more precisely with the use of the code:hasCode property.

— **photoID**: specifies whether the identifier contains a photo as a Boolean value.

— **validityPeriod**: a subproperty of time:hasTime, this identifies the time interval during which the identifier is valid.

— **issuedBy**: identifies the agent:Agent that issued the ID.

6.7.3 Formalization

Table 10 — Key classes in the Person pattern

Class	Property	Value restriction
Person	rdfs:subClassOf	agent:Agent
	hasPersonID	only PersonId
	name	max 1 PersonName
	alias	only PersonName
	contact:hasAddress	only contact:Address
	contact:hasTelephone	only contact:PhoneNumber
	email	only xsd:string
	birthDate	max 1 time:Instant
	birthplace	max 1 contact:Address
	deathDate	max 1 time:Instant
	deathplace	max 1 contact:Address
	citizenOf	only Citizenship
	parent	only Person
	spouse	only Person
	children	only Person
	sex	max 1 Sex
	hasGenderIdentity	only Gender
	income	only cityunits:MonetaryValue
	hasSkill	only Skill
	hasEducation	only Education
PersonName	givenName	only xsd:string
	additionalName	only xsd:string
	familyName	max 1 xsd:string
PersonID	genprop:hasIdentifier	exactly 1 xsd:string
	hasIDType	only IDType
	photoID	max 1 xsd:boolean
	validityPeriod	max 1 time:Interval
	issuedBy	max 1 agent:Agent
Citizenship	forCountry	max 1 addresss:Country
	validityPeriod	max 1 time:ProperInterval

Table 10 *(continued)*

Class	Property	Value restriction
Sex	code:hasCode	only code:Code
Gender	code:hasCode	only code:Code
IDType	code:hasCode	only code:Code
Skill	code:hasCode	only code:Code
Education	code:hasCode	only code:Code

Table 11 — Key properties in the Person pattern

Property	Characteristic	Value (if applicable)
validityPeriod	subPropertyOf	time:hasTime

6.8 City Resident pattern

6.8.1 General

NOTE Resident has been defined in a separate pattern because it is a specialization of the Person category defined in 6.7. Combining the two concepts into a single category would not be incorrect from a logical perspective. However, the categories are separated to define a core Person pattern, independent of any specializations, to facilitate the use of this document.

As different cities have different definitions of who is that city's Resident, the City Resident pattern contains the core properties required by each. For example, the city of Toronto's definition of a city resident includes the concept of owning property or owning or operating a business in the city. For Beijing, nationality is a unique aspect. Central to all the definitions is the concept of residing, variously referred to as a home or domicile in which the resident spends significant amounts of time; they can own it, rent it, or just stay in it. "Reside" has both a temporal and spatial dimension.

6.8.2 Key classes and properties

The key classes are formalized in Table 12. The CityResident class is a subclass of Person. The properties of the CityResident class are used to construct the definition of a resident for a particular city, which may then be defined as a subclass in an extension to this document. These properties are:

— **hasResidence**: specifies one or more individuals of Residence, where each individual specifies a residence distinguished by city, address and/or time interval. A resident can have more than one residence.

— **owns**: specifies an EntityOwnership where entities owned include include buildings, land areas, or organizations.

— **operates**: specifies an EntityOperation where the entity is an Organization that the resident operates.

All specializations (subclasses) of Resident shall have at least one hasResidence property that identifies where they reside. The remaining properties are optional and their specifications are intended to constrain their use in the context of specializations of Resident. For example, if an optional property is used in the definition of Toronto Resident, then its range is restricted to what is specified in Resident.

In the following definition of CityResident, the properties identified fall into two types: properties that are required in all specializations of Resident, e.g. Toronto Resident, Beijing Resident, and properties that are optional, but if used by a specialization of Resident, have their ranges restricted. A major part of determining whether a person is a resident of a city is the specification of where and when they have resided. The hasResidence property is required and links a CityResident to a Residence. The cardinality of the property is greater than one as over time a person may reside in more than one place/address, in the same city and/or different cities. The Residence class identifies the following key properties:

— **forCity**: identifies the city (City) of the residence.

— **time:hasTime** specifies the time interval during which the residence was held.

— **hasHomeType**: identifies the type of home, such house, apartment, or shelter. Its values may be defined more precisely with the use of the code:hasCode property.

— **contact:hasAddress**: identifies the address of the residence, if any.

— **hasResidentialRelationship**: specifies whether the residence is held by ownership, rental, or other arrangement. Its values may be defined more precisely with the use of the code:hasCode property.

The temporal intervals of individuals of Residences can used to determine a total or partial ordering of a person's residencies, as a person may reside in more than one place at the same time.

The ControlledEntity and Citizenship classes are necessary to capture the time interval during which an entity is owned or operated, or the person is a citizen of a country. The ControlledEntity class identifies the following key properties:

— **entity**: identifies a thing that is controlled.

— **time:hasTime**: identifies the time interval during which controlled relationship is held for the entity (i.e. during which the person controls it).

The ControlledEntity class is further specialized with the EntityOwnership and EntityOperation subclasses to indicate the entity control more precisely. EntityOwnership indicates an instance of ownership and is specialized with the following properties:

— **entity**: identifies a land area, organization, or building that is owned.

— **percentOwnership**: identifies the percent of the entity that is owned.

The EntityOperation class indicates an instance of entity control via operation. It is specialized with the following property:

— **entity**: indicates an Organization that is controlled. In some implementations it can be possible to infer entity operation on the basis of a person's role within an organization, but this is not formalized here.

6.8.3 Formalization

Table 12 — Key classes in the City Resident pattern

Class	Property	Value restriction
CityResident	rdfs:subClassOf	person:Person
	hasResidence	only Residence
	residentOf	only city:JurisdictionalArea
	owns	only EntityOwnership
	operates	only EntityOperation
Residence	forCity	max 1 city:City
	time:hasTime	max 1 time:ProperInterval
	hasHomeType	max 1 HomeType
	contact:hasAddress	max 1 contact:Address
	hasResidentialRelationship	max 1 ResidentialRelationship
ControlledEntity	time:hasTime	max 1 time:ProperInterval
	entity	some owl:Thing
EntityOwnership	rdfs:subClassOf	ControlledEntity
	entity	max 1 (landuse:LandArea or org:Organization or building:Building)
	percentOwnership	max 1 xsd:decimal

Table 12 *(continued)*

Class	Property	Value restriction
EntityOperation	rdfs:subClassOf	ControlledEntity
	entity	max 1 org:Organization
HomeType	code:hasCode	only code:Code
HousingTenure	code:hasCode	only code:Code

6.9 Household pattern

6.9.1 General

A household is an important concept in many areas of the city. Households are distinct, though often closely related to families and residences. This document does not provide an explicit representation of a Family, though one could be inferred from the properties of some Persons (as defined in the Person pattern). Residence is described in the City Resident pattern, and household is described in this pattern. The behaviour of a household can be represented by the collective activities of its members. Thus, membership is a key property of a household.

6.9.2 Key classes and properties

The key classes are formalized in Table 13. A Household refers to a collection of persons occupying a shared place of residence. Households may be (but are not necessarily) comprised of family members. Different, more precise definitions of Household may be adopted as required for different contexts and applications through extensions to this class. A Household has the following key properties:

— **householdOccupies**: identifies the cityresident:Residence that it occupies, i.e. the residence that is shared by its members. A Household is defined by this residence, such that if the members move (even collectively), the new residence constitutes a new Household.

— **org_s:hasMember**: identifies a person:Person who is a member of the Household.

— **time:hasTime**: specifies the time interval the Household exists in its current configuration.

6.9.3 Formalization

Table 13 — Key classes in the Household pattern

Class	Property	Value restriction
Household	householdOccupies	max 1 cityresident:Residence
	org_s:hasMember	only person:Person
	time:hasTime	only time:ProperInterval

6.10 City Organization pattern

6.10.1 General

An organization is defined broadly as a formal or semi-formal group for which structure and behaviour are defined.

Organizations such as schools and businesses are particularly important for cities as they determine regular travel patterns and other activities for much of the population. The Organization Pattern is drawn from the TOVE model of an Organization, originally presented in Reference [6]. It is an extension of the Organization Structure pattern defined in ISO/IEC 5087-1 which introduces city-specific concepts related to Organizations such as Students and Employees.

6.10.2 Key classes and properties

The key classes and properties are formalized in Table 14 and Table 15, respectively. An Organization is a company or other sort of formal or informal group of individuals in the urban system with some identified structure and behaviour. It is an extension of the org_s:Organization class defined in ISO/IEC 5087-1 to include properties relevant for city services. It includes the following key properties:

— **orgAddress**: a specialization to contact:hasAddress that is also defined according to the org_s:Site address specified in the Organization Structure Pattern defined in ISO/IEC 5087-1. Specifies the contact: Address associated with the Organization.

— **contact:hasTelep**hone: identifies the contact:PhoneNumber for the Organization.

— **operatingHours**: identifies the OperatingHours of the Organization.

— **hasGoal**: identifies the Goal(s) of the Organization. This allows for the representation of various groups' responsibilities and intents.

— **loc:hasLocation**: identifies the loc:associatedLocation(s) of the Organization. This is more general than the contact address, and allows for the representation of any number of spatial locations that the organization is associated with occupying (e.g. office spaces or other facilities).

An Organization may be further classified as a For-Profit Organization, Government Organization, or Non-Profit Organization. A For-Profit Organization has the following additional key properties:

— **hasIndustryType**: specifies an Industry Type assigned to the organization based on the kind of business conducted. There are different classifications of Industry Types, the inclusion of each is outside the scope of this document. Its values may be defined more precisely with the use of the code:hasCode property.

— **hasEstablishment**: specifies a Business Establishment where the organization conducts business.

A Government Organization has the following additional key properties:

— **hasProgram**: specifies a service:Program defined for the organization.

— **jurisdiction:** specifies the Jurisdictional Area that the organization is responsible for.

A Non-Profit Organization has the following additional key properties:

— **hasProgram**: specifies a service:Program defined for the organization.

A Role is a specialization of org_s:Role as defined in the Organization Structure Pattern in ISO/IEC 5087-1. It extends the class with the following properties:

— **hasGoal**: specifies a Goal defined for the Role. Any agent in the Role will have this as a Goal.

— **hasProcess**: specifies an activity:Activity that is performed as part of the Role.

— **hasResource**: specifies a resource:Resource that is (expected to be) allocated for the Role.

A Goal describes a desired state for an Organization, Organization Agent, or Role. It is defined as a kind of activity:State.

A Business Establishment is a physical location where a For-Profit Organization conducts business. It has the following properties:

— **loc:hasLocation**: specifies the loc:Location of the establishment.

— **contact:hasAddress**: specifies the contact:Address of the establishment.

An Organization Agent is a member of an organization. It has the following properties:

— **org_s:memberOf**: specifies the Organization that the Organization Agent is a member of.

— **playsRole**: specifies a Role that the Organization Agent is assigned in the context of the Organization.

— **hasGoal**: specifies the Goal of the Organization Agent in the context of the Organization. Goals may be inherited by the agent's Role, or defined directly for the Agent.

— **hasEmployment**: specifies the details of an agent's Employment, if applicable.

An Employment is a type of organizational membership where the agent receives monetary compensation for their membership in an Organization. It has the following properties:

— **employedAs**: identifies the occupation that the agent is employed as. An Occupation describes the type of work performed by some employee. Different classifications of occupations may be defined, such as: General Office/Clerical; Manufacturing/Construction/Trades; Professional/Management/Technical; Retail Sales and Service. Enumeration of this categorization is outside of the scope of this document in an extension as required for a given application. Its values may be defined more precisely with the use of the code:hasCode property.

— **hasCompensation**: identifies the monetary compensation received by the agent.

— **employedBy**: identifies the Organization that the agent is employed by.

— **hasEmploymentStatus**: identifies the Employment Status of the employee. An employment status may be categorized as one of: full-time regular, part-time regular, full-time-work-at-home, part-time-work-at-home. Enumeration of this categorization is outside of the scope of this document in an extension as required for a given application. Its values may be defined more precisely with the use of the code: hasCode property.

Compensation is a generalization of monetary compensation (received for employment). It is further defined as either a Wage or a Salary and has the following property:

— **hasPay**: identifies the cityunits:MonetaryValue of the Compensation.

Wage is a type of compensation that is defined on an hourly basis. It specializes Compensation with the following properties:

— **hourlyPay**: identifies the cityunits:MonetaryValue of compensation to be paid per hour.

— **overtimePay**: identifies an additional rate of pay as cityunits:MonetaryValue to be paid per hour in the case of overtime compensation.

Salary is a type of compensation that is defined on an annual basis. It specializes Compensation with the following property:

— a**nnualPay**: identifies the cityunits:MonetaryValue of compensation to be paid for a year.

Operation specifies the time during which an organization regularly conducts business (i.e. its hours of operation). It uses the Recurring Event pattern (defined in ISO/IEC 5087-1) to define its operation as a recurring event. Operation specializes a Recurring event with the following properties:

— **recurringevent:hasDayOfWeek**: specifies the day of the week for this recurring instance of Operation.

— **hasOpeningTime**: specifies the opening time for this instance of Operation.

— **hasClosingTime**: specifies the closing time for this instance of Operation.

6.10.3 Formalization

Table 14 — Key classes in the City Organization pattern

Class	Property	Value restriction
Organization	rdfs:subClassOf	org_s:Organization
	orgAddress	only contact:Address
	contact:hasTelephone	only contact:PhoneNumber
	operatingHours	only Operation
	hasGoal	only Goal
	loc:hasLocation	only loc:Location
ForProfitOrganization	rdfs:subClassOf	Organization
	hasIndustryType	only IndustryType
	hasEstablishment	only BusinessEstablishment
GovernmentOrganization	rdfs:subClassOf	Organization
	hasProgram	only service:Program
	jurisdiction	only city:JurisdictionalArea
NonProfitOrganization	rdfs:subClassOf	Organization
	hasProgram	only service:Program
Role	rdfs:subClassOf	org_s:Role
	hasGoal	only Goal
	hasProcess	only activity:Activity
	hasResource	only resource:Resource
Goal	rdfs:subClassOf	activity:State
BusinessEstablishment	loc:hasLocation	max 1 loc:Location
	contact:hasAddress	only contact:Address
OrganizationAgent	rdfs:subClassOf	agent:Agent
	org_s:memberOf	only Organization
	playsRole	only Role
	hasGoal	only Goal
	hasEmployment	only Employment
Employment	employedAs	some Occupation
	hasCompensation	some Compensation
	employedBy	some Organization
	hasEmploymentStatus	only EmploymentStatus
Compensation	hasPay	max 1 cityunits:MonetaryValue
Wage	rdfs:subClassOf	Compensation
	hourlyPay	max 1 cityunits:MonetaryValue
	overtimePay	only cityunits:MonetaryValue
Salary	rdfs:subClassOf	Compensation
	annualPay	max 1 cityunits:MonetaryValue
Operation	rdfs:subClassOf	recurringevent:RecurringEvent
	recurringevent:hasDayofWeek	max 1 {friday, monday, saturday, sunday, thursday, tuesday, wednesday}
	hasOpeningTime	max 1 xsd:time
	hasClosingTime	max 1 xsd:time

Table 14 *(continued)*

Class	Property	Value restriction
IndustryType	code:hasCode	only code:Code
EmploymentStatus	code:hasCode	only code:Code
Occupdation	code:hasCode	only code:Code

Table 15 — Key properties in the City Organization pattern

Property	Characteristic	Value (if applicable)
org_s:hasSite o org_s: siteAddress	rdfs:subPropertyOf	orgAddress
hourlyPay	rdfs:subPropertyOf	hasPay
overtimePay	rdfs:subPropertyOf	hasPay
hasAnnualPay	rdfs:subPropertyOf	hasPay
hasOpeningTime	rdfs:subPropertyOf	recurringevent:startTime
hasClosingTime	rdfs:subPropertyOf	recurringevent:endTime

6.11 City pattern

6.11.1 General

The City pattern combines information captured in several patterns, specifically: the land areas occupied by cities, government organizations and administrative areas, and associated bylaws.

6.11.2 Key classes and properties

The key classes are formalized in Table 16. More general than a City is the concept of a JurisdictionalArea. A JurisdictionalArea is an abstract entity that is characterized not only by its location, but by the objects that occupy it (persons, buildings, etc.), the governing body(s) it is subject to, and the activities that occur within it. It has the power to implement administrative or policy decisions (made and enacted on its behalf by its governing body). A JurisdictionalArea has the following properties:

— **genprop:hasName**: The name assigned to the Jurisdictional Area.

— **hasLandArea**: identifies the spatial area occupied by the JurisdictionalArea.

— **residentPopulation**: number of residents of the area. Note in many cases that this number may be distinct from the population associated with the Land Area that an Administrative Area occupies. The resident population is determined based on the definition of a resident which is often times more specific than an occupant of an area.

— **hasGovernment**: identifies the GovernmentOrganization that manages the JurisdictionalArea.

— **hasBylaw**: identifies the bylaws in existence in the JurisdictionalArea.

— **administrativeArea**: identifies administrative areas within the JurisdictionalArea. They do not have to be distinct and can be hierarchical.

— **administrativeAreaOf:** identifies an administrative area that the JurisdictionalArea is part of (e.g. a ward is part of a city).

A City is a specialization of a Jurisdictional Area that is formally identified as such. It has the following additional property:

— **legalName:** identifies the legal name of the City.

A CityAdministrativeArea is specialization of a Jurisdictional Area that has been identified for use by a City to reflect its unique areas such as districts, wards, neighbourhoods, or prefectures. It is specialized with the following property:

— **administrativeAreaOf**: identifies a Jurisdictional Areas that this administrative area is part of. A City Administrative Area is identified as an administrative area of a City.

6.11.3 Formalization

Table 16 — Key classes in the City pattern

Class	Property	Value restriction
JurisdictionalArea	genprop:hasName	max 1 xsd:string
	landuse:hasLandArea	only landuse:LandArea
	residentPopulation	max 1 i72:Population
	hasGovernment	only org:GovernmentOrganization
	hasBylaw	only bylaw:Bylaw
	administrativeArea	only JurisdictionalArea
	administrativeAreaOf	only JurisdictionalArea
City	rdfs:subClassOf	JurisdictionalArea
	legalName	max 1 xsd:string
CityAdministrativeArea	rdfs:subClassOf	JurisdictionalArea
	administrativeAreaOf	some City

6.12 City Service pattern

6.12.1 General

NOTE In some contexts (namely, computing) the inclusion of an Application category can also be expected, given that concept of an application is closely related to that of a service. However, the technology-oriented concepts of Application and Service have not been identified as appropriate for this document. The concept of service defined by the City Service Pattern is more generic, as described below.

Cities provide a variety of services to residents and businesses, including health and social services. The City Service pattern is based on the Canadian Government Reference Model (CGRM).[7] It identifies the following concepts as a basis for understanding the services that governments provide:

— Programs: these are major city initiatives that address the needs of their constituents (citizens, clients). They are a mandate to achieve outcomes by delivering services. For example, ending homelessness.

— Services: these deliver outputs to clients that contribute to program outcomes. For example, providing shelters for the homeless.

— Processes: these are activities that deliver services. For example, homeless person registration, bed allocation, etc.

— Resources: these are used in carrying out processes. For example, shelter space, beds and personnel.

6.12.2 Key classes and properties

The key classes are formalized in Table 17. A Program defines a set of services that focus on a shared set of outcomes. For example, a “poverty reduction program” can be made up of a set of services such as mobiles services that provide food and clothing to those that live on the street, and a training service that provides basic skills for those living on the street. A program has a set of stakeholders that can contribute or benefit.

A program is defined as a type of activity: it is a high-level activity, made up of all of the more granular activities (e.g. services) involved in it. A program has the following properties:

— **genprop:hasName**: identifies the name of the program. All programs require a name.

— **genprop:hasDescription**: identifies a description of the program.

— **hasService**: identifies the services that make up the program.

— **hasOutcome**: identifies the outcomes that the program is trying to achieve.

— **hasContributingStakeholder**: identifies the stakeholders that contribute to the program.

— **hasBeneficialStakeholder:** identifies the stakeholders that benefit from the program.

— **hasInput**: identifies the inputs to the program.

— **hasOutput**: identifies the outputs of the program.

A program is composed of one or more services. In turn, each services may be comprised of different activities, inputs, outputs and outcomes. The following are the service properties:

— **activity:hasSubActivity**: identifies more specific activities that comprise the service.

— **hasInput**: identifies the inputs to the service.

— **hasOutput**: identifies the outputs of the service.

— **hasOutcome**: identifies the outcomes that are specific to the service.

— **hasContributingStakeholder**: identifies the stakeholders that contribute to the service.

— **hasBeneficialStakeholder**: identifies the stakeholders that benefit from the service.

Outcomes are what stakeholders experience as a result of a program or service. Outcomes capture positive and negative, intended and unintended results. Outcome contains the following properties:

— **hasIndicator**: identifies an i72:Indicator to measure to the outcome.

— **genprop:hasDescription**: identifies a general description of the outcome as a string.

— **hasBeneficialStakeholder**: identifies the stakeholder affected.

— **fromPerspectiveOf**: identifies the stakeholder who is determining the importance of the impact.

— **hasImportance**: specifies how important the impact is (e.g. high, medium, low). Its values may be more precisely defined using the code:hasCode property.

— **intendedImpact**: identifies the intended direction of the change (e.g. positive, negative). Its values may be more precisely defined using the code:hasCode property.

A Stakeholder is a person or organization that either contributes to or benefits from a program or service or both. It has the following properties:

— **genprop:hasName**: identifies a title for the stakeholder as a string.

— **genprop:hasDescription**: identifies a general description of the stakeholder as a string.

— **hasCatchmentArea**: identifies the regional span of the stakeholders as a loc:Location

— **hasCatchmentAreaType**: identifies the type of regional span of the stakeholders (e.g. local, provincial). Its values may be defined more precisely with the use of the code:hasCode property.

— **performs**: identifies activities performed by the stakeholder.

Input is a type of resource:TerminalResourceState and defines the resources and the stakeholders that contribute them that are input to an activity:

— **hasContributingStakeholder**: the stakeholders that contribute the resources as input.

— **i72:for_time_interval**: specifies the time interval over which the input is used.

— **genprop:hasName**: name for the input.

— **genprop:hasDescription**: description for the input.

Output is a type of resource:TerminalResourceState that provides a quantitative summary of an activity. For example, if the activity is "we provide training" the output could be "we trained 50 people to NVQ level 3". Or a production output could produce 100 meals for the homeless. Basic elements of these outputs are “what” has been produced and the quantity.

— **usedByIndicator**: identifies the indicators that use this output in determining the value of the Indicator.

— **genprop:hasName**: identifies a name for the output.

— **genprop:hasDescription**: identifies a description for the output

In addition, the City Services pattern introduces the hasProgram property to support the reference to a Program by classes in other patterns.

6.12.2.1 Formalization

Table 17 — Key classes in the City Services pattern

Class	Property	Value restriction
Program	rdfs:subClassOf	activity:Activity
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	only xsd:string
	hasService	only Service
	hasOutcome	only Outcome
	hasContributingStakeholder	only Stakeholder
	hasBeneficialStakeholder	only Stakeholder
	hasInput	only Input
	hasOutput	only Output
Service	rdfs:subClassOf	activity:Activity
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string
	hasInput	only Input
	hasOutput	only Output
	hasOutcome	only Outcome
	hasContributingStakeholder	only Stakeholder
	hasBeneficialStakeholder	only Stakeholder
Outcome	hasIndicator	only i72:Indicator
	genprop:hasDescription	only 1 xsd:string
	hasBeneficialStakeholder	max 1 Stakeholder
	fromPerspectiveOf	max 1 Stakeholder
	hasImportance	max 1 Importance
	intendedImpact	max 1 ImpactDirection
ImpactDirection	code:hasCode	only code:Code

Table 17 *(continued)*

Class	Property	Value restriction
Importance	code:hasCode	only code:Code
Stakeholder	rdfs:subClassOf	(org:Organization or person:Person)
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	only xsd:string
	hasCatchmentArea	only loc:Location
	hasCatchmentAreaType	only CatchmentAreaType
	performs	some activity:Activity
CatchmentAreaType	code:hasCode	only Code
Input	rdfs:subClassOf	resource:TerminalResourceState
	hasContributingStakeholder	only Stakeholder
	i72:for_time_interval	only time:DateTimeInterval
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string
Output	rdfs:subClassOf	resource:TerminalResourceState
	usedByIndicator	only i72:Indicator
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string

6.13 Contract pattern

6.13.1 General

A contract is a legal document that specifies an agreement(s) between two or more parties. The aim of the Contract pattern is not to formalize the semantics of all possible involved legal concepts, but rather to enable to representation of the general structure and contents of a particular contract. A Contract is defined as a type of document and is distinct from the agreement it specifies.

6.13.2 Key classes and properties

The key classes and properties are formalized in Table 18 and Table 19, respectively. A Contract is a document with the additional following properties:

— **specifiesAgreement**: identifies the agreement(s) that are specified by some Contract.

— **hasParty**: identifies the person(s) and/or organization(s) that are involved in the Contract.

— **hasSignatory**: identifies the person(s) responsible for signing the Contract.

— **hasContractualElement**: identifies the decomposition of a Contract into more precise parts.

— **isValidFor**: identifies the interval in time over which the Contract is valid for, if applicable.

— **isExecutedOn**: identifies the interval or instant in time at which the terms in the Contract are executed, if applicable.

A ContractualElement represents a part of a contract. Therefore, it can only exist in the context of a given Contract.

— **hasContractText**: identifies the excerpt of text from the Contract that corresponds to the ContractualElement.

A ContractualElement may be more precisely identified as:

— **ConditionPrecedent**: a part of the Contract that identifies conditions that is required to be met in order for the contract to take effect.

— **ContractualCommitment**: a part of the Contract that identifies some Agreement between the parties. It has the following property:

  — **specifiesAgreement**: indicates an Agreement specified by this part of the Contract.

— **ContractualDefinition**: a part of the Contract that defines key terms that are referred to in the Contract.

— **NonBindingTerm**: a part of the Contract that is not legally binding.

— **Representation**: a part of the Contract that specifies some assertions that are taken to be true at the time of the contract and serve to influence a party's decision to enter into the Contract.

— **Warranty**: a part of the Contract that promises some indemnification if an assertion made in the Contract is false.

A general hasContract property is also defined to associate other objects (e.g. services, employment) with a Contract object.

6.13.3 Formalization

Table 18 — Key classes in the Contract pattern

Class	Property	Value restriction
Contract	specifiesAgreement	some agreement:Agreement
	hasParty	min 2 (person:Person or org:Organization)
	hasSignatory	min 2 person:Person
	hasContractualElement	some ContractualElement
	isValidFor	only time:Interval
	isExecutedOn	only time:TemporalEntity
ContractualElement	inverse (hasContractualElement)	some Contract
	hasContractText	some xsd:string
ConditionPrecedent	rdfs:subClassOf	ContractualElement
ContractualCommitment	rdfs:subClassOf	ContractualElement
	specifiesAgreement	some agreement:Agreement
ContractualDefinition	rdfs:subClassOf	ContractualElement
NonBindingTerm	rdfs:subClassOf	ContractualElement
Representation	rdfs:subClassOf	ContractualElement
Warranty	rdfs:subClassOf	ContractualElement

Table 19 — Key properties of the Contract pattern

Property	Characteristic	Value (if applicable)
hasContract	Range	Contract

6.14 Bylaw pattern

6.14.1 General

Municipal bylaws are types of laws that are distinguished by their origin (passed by a municipal government), jurisdiction (apply to the municipality, or some area within the municipality), and authority (municipal laws are often limited by higher levels of government in terms of what they can regulate and how).

The intent of the Bylaw pattern is to capture the major components of a municipal bylaw, such as dates, geographic areas of application, penalties, etc. It is not intended to provide a legal semantics with which to codify a particular bylaw. The following three types of bylaws are represented in the Bylaw pattern:

a) Main bylaws;

b) Amending bylaws, which reflect material changes, in principle or substance, to an existing bylaw; and

c) Revision bylaws, which reflect limited changes to an existing bylaw.

6.14.2 Use cases

— A commercial organization operating within the city is considering storing or manufacturing hazardous materials. They wish to know what restrictions existing on the handling, storage and manufacturing of hazardous materials. The various search mechanisms are able to precisely find the information as the bylaw explicitly specifies the resources or activities or both that are the focus of the bylaw, as the pattern provides the appropriate tagging of the information.

— A homeowner in the city wishes to remove a tree for their property. They wish to know whether there are any restrictions regarding tree removal. They use their favourite search engine to find the relevant bylaw as the bylaw was tagged using the pattern, making it easy for the search engine to find the relevant law amongst the myriad of information found.

6.14.3 Key classes and properties

The key classes are formalized in Table 20. The core concept of the Bylaw pattern is Law, of which a Bylaw is a specific type. Its properties decompose a Law into conceptually relevant components that support connection with other city concepts. The following are a Law's properties:

— **genprop:name**: identifies a short name for the Bylaw, to be used in other descriptions to refer to the Law.

— **legislationJurisdiction**: identifies the jurisdiction that enacted the Law.

— **legislationIdentifier**: specifies a unique identifier for the bylawLaw. The range is a xsd:string.

— **abstract**: specifies a brief statement of the Law purpose. The range is a xsd:string.

— **keywords**: identifies keywords used to categorize the Law for search purposes. The range is zero of more xsd:string.

— **legislationLegalForce**: specifies whether the Law is currently in force, not in force or partially in force.

— **hasDefinition**: words or phrases that are defined to have a specific meaning within the context of the Law. The range is restricted to Definition.

— **impacts**: identifies what is impacted by the Law. Not restricted to any class type.

— **legislationDate**: date which the Law is officially adopted/signed by the legislationJurisdiction city: JurisdictionalArea.

— **datePublished**: date the Law is officially published.

— **dateInEffect**: date after which the Law is in effect.

— **expires**: date on which the Law expires.

— **hasClause**: clauses that make up the body of the Law. Restricted to Clause.

— **hasPenaltyClause**: clauses that specify penalties for not adhering to the Law.

— **hasSeveranceClause**: clauses that specify that the bylaw remains valid if any portion of the Law is found to be invalid by a higher jurisdiction.

— **hasTransitionClause**: clauses that cover the period during which entities affected by the Law can do things to conform to the new conditions.

— **hasRepealClause**: clauses that specify all previous Laws that deal with subjects that are addressed in the new bylaw are required to be either repealed or amended.

— **hasSchedule**: schedules that are attached to the Law and are referenced by the Law. A Schedule is part of the Bylaw.

A Bylaw is a type of Law put in place by city. It adds and specializes with the following properties:

— **legislationJurisdiction**: identifies the city:City that enacted the Bylaw.

— **legislationType**: type of bylaw chosen from bylawMain, bylawAmending or bylawRevision.

— **impacts**: who and/or what is impacted by the Bylaw. Restricted to Person, Organization, LandArea, city: JurisdictionalArea, and/or Activities.

A MainBylaw is a subclass of Bylaw and has the following additional properties:

— **legislationType**: value bylawMain

An AmendingBylaw is a subclass of Bylaw and has the following additional properties:

— l**egislationChanges**: the Bylaw that is being amended.

— **legislationType**: value bylawAmending.

A RevisionBylaw is a subclass of Bylaw and has the following additional properties:

— **legislationChanges**: the Bylaw that is being amended.

— **legislationType**: value bylawRevision.

A Definition concept that specifies the defined terms used in the Bylaw. It has the following properties:

— **genprop:hasName**: the formal term being defined. It is an xsd:string.

— **genprop:hasDescription**: the definition of the genprop:hasName.

— **partwhole:properPartOf**: the law that the definition is a part of.

A Clause is a statement of a rule, provision, requirement, etc. that is part of the body of the Bylaw, or its schedules, penalties, etc. It has the following properties:

— **genprop:hasIdentifier**: unique identifier/number for the clause. It is an xsd:string.

— **genprop:hasName**: title or name of the clause, if any. It is an xsd:string.

— **genprop:hasDescription**: the content of the clause.

— **clauseType**: one of (severance or penalty or transition or repeal or schedule or bylaw).

— **hasClause**: links to any subclauses of this clause.

— **partwhole:properPartOf**: The part of the law or clause this clause is contained in.

A Schedule is attached to the Bylaw and is part of the Bylaw. It has the following properties:

— **genprop:hasIdentifie**r: unique identifier/number for the schedule. It is an xsd:string.

— **genprop:hasName**: title or name of the schedule, if any. It is an xsd:string.

— **genprop:hasDescription**: an introductory description of the Schedule. It is an xsd:string.

— **hasClause**: links to all clauses contained in the Schedule.

— **partwhole:properPartOf**: the law this Schedule is part of.

6.14.4 Formalization

Table 20 — Key classes in the Bylaw pattern

Class	Property	Value restriction
Law	genprop:hasName	only xsd:string
	legislationJurisdiction	max 1 city:JurisdictionalArea
	legislationIdentifier	max 1 xsd:string
	abstract	max 1 xsd:string
	keywords	only xsd:string
	legislationLegalForce	only {InForce, NotInForce, PartiallyInForce}
	hasDefinition	only Definition
	legislationDate	max 1 xsd:DateTime
	datePublished	max 1 xsd:DateTime
	dateInEffect	max 1 xsd:DateTime
	expires	max 1 xsd:DateTime
	hasClause	only Clause
	hasPenaltyClause	only Clause
	hasSeveranceClause	only Clause
	hasTransitionClause	only Clause
	hasRepealClause	only Clause
	hasSchedule	only Schedule
Bylaw	rdfs:subClassOf	Law
	legislationJurisdiction	max 1 city:City
	legislationType	only {mainBylaw, amendingBylaw, revisionBylaw}
	impacts	only (person:Person or org:Organization or landuse:LandArea or city:JurisdictionalArea or activity:Activity)
MainBylaw	rdfs:subClassOf	Bylaw
	legislationType	value mainBylaw
AmendingBylaw	rdfs:subClassOf	Bylaw
	legislationType	value amendingBylaw
	legislationChanges	exactly 1 Bylaw
RevisionBylaw	rdfs:subClassOf	Bylaw
	legislationType	value revisionBylaw
	legislationChanges	exactly 1 Bylaw

Table 20 *(continued)*

Class	Property	Value restriction
Definition	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string
	partwhole:properPartOf	exactly 1 Law
Clause	genprop:hasIdentifier	max 1 xsd:string
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string
	clauseType	some {severance,penalty,transition, repeal, schedule, bylaw}
	hasClause	only Clause
	partwhole:properPartOf	exactly 1 (Law or Clause)
Schedule	genprop:hasIdentifier	max 1 xsd:string
	genprop:hasName	max 1 xsd:string
	genprop:hasDescription	max 1 xsd:string
	hasClause	only Clause
	partwhole:properPartOf	exactly 1 Law

6.15 Contact pattern

6.15.1 General

Contact information is relevant for a range of concepts in the city domain. For example, a building can have an associated address. Similarly, a person or an organization can have a contact address (or phone number, email address, etc.). A person's contact address can differ from their place of residence.

Rather than define these attributes separately for persons, organizations, etc., it makes sense to capture the general concepts associated with contact information in a separate pattern.

6.15.2 Key classes and properties

The key classes are formalized in Table 21. Both Address and PhoneNumber classes are designed to accommodate international versions. In addition to drawing from the iContact ontology, the pattern reuses concepts from the Spatial Location pattern (defined in ISO/IEC 5087-1) to associate an address with a location.

The Address Class has the following properties:

— **hasAddressType**: specifies the type of address, e.g. home, work. The values for AddressType may be defined more precisely with the use of the code:hasCode property.

— **hasStreetNumber**: specifies the number on the street for the address.

— **minStreetNumber**: a subproperty of hasStreetNumber, specifies the minimum street number of an address in the case that it is defined a street number range.

— **maxStreetNumber**: a subproperty of hasStreetNumber, specifies the maximum street number of an address in the case that it is defined with a street number range.

— **hasStreet**: specifies the name of the street for the address.

— **hasStreetType**: specifies the type of the street for the address, e.g. road, drive. The values for StreetType may be defined more precisely with the use of the code:hasCode property.

— **hasStreetDirection**: specifies the direction of the street for the address, e.g. east, west. The values for StreetDirection may be defined more precisely with the use of the code:hasCode property.

— **hasUnitNumber**: specifies the unit or suite number of the address.

— **hasPostalBox**: specifies the box number for the address.

— **hasBuilding**: specifies the name of the building for the address.

— **hasCitySection**: specifies the section of the city for the address.

— **hasCity**: specifies the name of the city for the address.

— **hasProvince**: specifies the state or province for the address. Its values may be defined more precisely with the use of the code:hasCode property.

— **hasPostalCode**: specifies the zip or postalcode for the address.

— **hasCountry**: specifies the country for the address. The values of Country may be defined more precisely with the use of the code:hasCode property. Use of the ISO 3166-2 alpha-2 2 letter country code is recommended.

— **loc:hasLocation**: specifies the geospatial representation of the location of the address.

The PhoneNumber class has the following properties:

— **hasCountryCode**: specifies the country code for the number.

— **hasAreaCode**: specifies the area code for the number.

— **hasPhoneNumber**: specifies the remaining digits of the number after the area code.

— **hasPhoneType**: specifies the type of phone number, e.g. cell phone, home phone, etc. PhoneType values may be defined more precisely with the use of the code:hasCode property.

6.15.3 Formalization

Table 21 — Key classes in the Contact pattern

Class	Property	Value restriction
Address	hasAddressType	only AddressType
	hasStreetNumber	max 1 xsd:string
	minStreetNumber	max 1 xsd:string
	maxStreetNumber	max 1 xsd:string
	hasStreet	max 1 xsd:string
	hasStreetType	max 1 StreetType
	hasStreetDirection	max 1 StreetDirection
	hasUnitNumber	max 1 xsd:string
	hasPostalBox	max 1 xsd:string
	buildingName	max 1 xsd:string
	hasCitySection	max 1 xsd:string
	hasCity	max 1 city:City
	hasProvince	max 1 State
	hasPostalCode	max 1 xsd:string
	hasCountry	max 1Country
	loc:hasLocation	max 1 loc:Location
AddressType	code:hasCode	only code:Code
StreetDirection	code:hasCode	only code:Code
StreetType	code:hasCode	only code:Code

Table 21 *(continued)*

Class	Property	Value restriction
State	code:hasCode	only code:Code
Country	code:hasCode	only code:Code
PhoneNumber	hasCountryCode	max 1 xsd:nonNegativeInteger
	hasAreaCode	max 1 xsd:nonNegativeInteger
	hasPhoneNumber	max 1 xsd:nonNegativeInteger
	hasPhoneType	max 1 PhoneType
PhoneType	code:hasCode	only code:Code

6.16 Sensors pattern

6.16.1 General

NOTE Sensors are included in the city-level of the ISO/IEC 5087 series (this document) due to their broad relevance across city services. Actuators and other categories of knowledge specific to control systems are not currently identified as relevant for multiple services and so are not included in this document.

The Sensors pattern is included in this document due to the importance of data collection for all manner of city services. Data collection efforts take various form, whether through manual canvassing or the use of sensors. With a growing access to the Internet of Things, data from available sensors will continue to expand, likely to include observations about persons, vehicles, and so on. It is important to not only capture the collected data, but also the source of the observations.

The representation of sensors shall conform to the ontology specified in The W3C Recommendation "The SSN (Semantic Sensor Network) Ontology".[6] It is included in its entirety with the prefix 'ssn'. The W3C standard ontology for sensors and their observations, the SSN (Semantic Sensor Network) Ontology, accessed from https://www.w3.org/ns/ssn/ shall be used in the context of this document.

Annex A
(informative)

Relationship to existing standards

A.1 CityGML

A.1.1 Scope

CityGML[8] is an XML-based standard for representing 3-dimensional city models. Target application areas identified include: “urban and landscape planning; architectural design; tourist and leisure activities; 3-dimensional cadastres; environmental simulations; mobile telecommunications; disaster management; homeland security; vehicle and pedestrian navigation; training simulators and mobile robotics.” It is intended to capture the data necessary to generate 3-dimensional portrayals in appropriate tools, providing not only geometry but data regarding surface characteristics and objects of interest (e.g. buildings, water bodies). Data mappings between related terms are identified and summarized in Table A.1.

A.1.2 Relevance

Although the purpose of CityGML is to support the capture and exchange of 3-dimensional city models, the thematic extension modules of CityGML capture information on city objects and so are relevant to this document.

A.1.3 Data mappings

Table A.1 — Relationship between terms in this document and CityGML

ISO/IEC 5087-2 term	Corresponding CityGML term	Relationship
contact:Address	Address	In CityGML, an Address is associated with an XALAddress, defined according to the eXtensible Address Language (OASIS),[9] an XML vocabulary for addresses. Many of the properties identified for a contact:Address have correspondences with xAL elements (though there are no formal definitions, so precise mappings are not possible). However, the xAL Address element itself defines an address as a free format in contrast with the properties specified for contact:Address. CityGML also associates a MultiPoint geometry with an Address object via the multiPoint property. Similarly, contact:Address is also associated to a geometry through the loc:hasLocation property. However, it is not restricted to multipoint geometries.
building:Building	Building	The CityGML representation captures different parts of a building such as structural elements and logical subdivisions, whereas building:Building's focus is on units (as these are central to representing occupancy). Both describe a number of physical characteristics, such as the number of floors (storeys). CityGML introduces a property for a building's function and usage, whereas building:Building only identifies building:BuildingUse. building:Building also identifies construction-related information such as relevant contruction codes and construction status. The intended meaning of the terms is generally aligned though neither one is a specialization (or generalization) of the other.
landuse:LandArea	LandUse	LandUse objects are areas that (are identified to) have specific physical and biological uses/characteristics. ISO/IEC 5087-2 defines LandArea in a similar way. The LandUse class distinguishes between associated functions (intended purposes) and uses (actual uses) of the land area, whereas landuse:LandArea specifies a single property for land use that can capture both use and function (depending on the classification system used).
landuse:LandUse	LandUseFunctionValue, LandUseUsageValue	The land use classification of some area is represented with the LandUse class in ISO/IEC 5087-2. A given classification system may identify intended use, actual use, or both. It is identified with code lists for LandUseFunctionValue and LandUseUsageValue in CityGML. The main distinction between these approaches is that the LandUse class supports a more detailed representation with possible extensions to define the classification systems, and more detailed information on the codes themselves using the Code Pattern. In contrast, the land uses and land functions in CityGML are defined simply as code lists (enumerations of possible values).

Table A.1 *(continued)*

ISO/IEC 5087-2 term	Corresponding CityGML term	Relationship
transinfras:Bridge	Bridge	Overall, the Bridge term in CityGML provides a more detailed definition than that of transinfras:Bridge. It specifies more specific properties (e.g. relating to the bridge's usage, construction materials) that are not captured in ISO/IEC 5087-2. Despite this, transinfras:Bridge identifies a relationship between a bridge and a transinfras:TravelledWaySegment that it supports; nothing comparable is captured in CityGML. Both terms specify a decomposition – CityGML defines BridgeParts, while transinfras:Bridge defines BridgeSegments.
transinfras:Tunnel	Tunnel	Overall, the Tunnel term in CityGML provides a more detailed definition than that of transinfras:Tunnel. It specifies more specific properties (e.g. relating to the tunnel's usage, construction materials) that are not captured in ISO/IEC 5087-2. Despite this, transinfras:Tunnel identifies a relationship between a tunnel and a transinfras:TravelledWaySegment that it supports; nothing comparable is captured in CityGML. Both terms specify a decomposition – CityGML defines TunnelParts, while transinfras:Tunnel defines TunnelSegments.
transinfras:RailLine	Railway	In CityGML, Railways are associated with Intersections and subdivided into Sections. In ISO/IEC 5087-2, there is no specification of intersections, but the decompositon is more detailed, distinguishing between transinfras:RailLinks and transinfras:RailSegments. CityGML specifies detailed attributes such as rail function and use that are not included in ISO/IEC 5087-2. The terms are closely related but neither one is a specialization (or generalization) of the other.
transinfras:Road	Road	In CityGML, roads are associated with Intersections and subdivided into Sections. In ISO/IEC 5087-2, there is no specification of intersections, but the decompositon is more detailed, distinguishing between transinfras:RoadLinks and transinfras:RoadSegments. CityGML specifies detailed attributes such as road function and use that are not included in ISO/IEC 5087-2. The terms are closely related but neither one is a specialization (or generalization) of the other.

A.2 INSPIRE

A.2.1 Scope

The INSPIRE directive[10] is aimed at supporting the sharing of and access to spatial data throughout the EU, particularly data that can have an impact on the environment. INSPIRE aims to create an infrastructure to achieve this, part of which includes the specification of data models in the Unified Modelling Language (UML). These specifications are defined according to 34 data themes, ranging from Addresses, to Geology, to Human Health and Safety. Data mappings between related terms are identified and summarized in Table A.2.

A.2.2 Relevance

Out of the 34 data themes identified, the Cadastral Parcels,[11] Administrative Units,[12] Land Cover,[13] Land Use,[14] and Buildings[15] specifications are of particular relevance to the ontologies defined in this document.

A.2.3 Data mappings

The data models are defined in UML. However, much of the intended semantics is captured in the accompanying description rather than the model itself. The mappings that follow have been identified based on these descriptions. A key distinction between INSPIRE and this document is that the classes identified in INSPIRE are all interpreted as spatial objects, whereas in this document most of the classes are related to spatial objects through a "hasLocation" property.

Table A.2 — Relationship between terms in this document and INSPIRE

ISO/IEC 5087-2 term	Corresponding INSPIRE term	Relationship
landuse:LandArea	CadastralParcel	The CadastralParcel term in INSPIRE is a specialization of LandArea as defined in ISO/IEC 5087-2. INSPIRE describes the CadastralParcel as "…a single area of Earth surface (land and/or water), under homogeneous real property rights and unique ownership, real property rights and ownership being defined by national law."[11] In the context of ISO/IEC 5087-2, a LandArea is a generic representation of an identified region of land (possibly but not necessarily with unique ownership). CadastralParcel has an associated cadastral reference and optional portrayal attributes, whereas a LandArea does not. Both terms have some associated geometry and area.
landuse:LandArea	CadastralZoning	The CadastralZoning term in INSPIRE is a specialization of LandArea as defined in ISO/IEC 5087-2. INSPIRE describes the CadastralZoning as "the intermediary areas (such as municipalities, sections, blocks, …) used in order to divide national territory into cadastral parcels."[11] In the context of ISO/IEC 5087-2, a LandArea is a generic representation of an identified region of land. Different levels of LandAreas may be defined, but no distinction is made between them. CadastralZoning has an associated cadastral zoning reference and optional portrayal attributes whereas a LandArea does not. Both terms have some associated geometry and area.
city:JurisdictionalArea	AdministrativeUnit	The AdministrativeUnit in INSPIRE represents the spatial area itself, whereas the city:JurisdictionalArea class represents the abstract entity that has jurisdiction over the spatial area. In other words, the INSPIRE AdministrativeUnit corresponds to the landuse:LandArea defined for a city:JurisdictionalArea.

Table A.2 *(continued)*

ISO/IEC 5087-2 term	Corresponding INSPIRE term	Relationship
Landuse:LandArea, landuse:hasLandUse	LandCoverVector, LandCover-Unit	Both specifications adopt the same position for the representation of land use as that of INSPIRE: not to prescribe a particular land cover classification system but to provide a mechanism of capturing land cover data. ISO/IEC 5087-2 associates land with various land use and classification systems with a single, hasLandUse property. Whereas this module of INSPIRE focuses solely on land cover, ISO/IEC 5087-2 includes both types of classification systems. INSPIRE provides a generic structure (LandCoverNomenclature) for defining classification codes whereas ISO/IEC 5087-2 does not. INSPIRE distinguishes between vector and raster land cover representations whereas ISO/IEC 5087-2 does not. In ISO/IEC 5087-2 land cover classes are associated with LandAreas in the same way as land use classes are assigned to areas. INSPIRE captures land over as observations and associates an observation date. It allows for the specification of a "covered percent" where in ISO/IEC 5087-2 it is assumed that the land cover is defined to completely cover the identified area of land.
landuse:LandArea, landuse:hasLandUse	ExistingLandUseObject, ExistingLandUseSample, ExistingLandUseGrid, ZoningElement	ISO/IEC 5087-2 provides a generic approach to representing land use. Two land use classification systems are currently included, but it is intended to be easily extended with others. INSPIRE requires the use of the HILUCS (Hierarchical INSPIRE Land Use Classification System), but also allows for the specification of other classification systems. INSPIRE clearly distinguishes between Land Use and Land Cover, whereas ISO/IEC 5087-2 does not, allowing for classification systems that combine the two notions. ISO/IEC 5087-2 describes land use with the property hasLandUse that is associated to a LandArea. INSPIRE introduces different classes (geometries) to represent Existing Land Use, Sampled Land Use, Gridded Land Use, and Planned Land Use (zoning). Existing, Sampled and Gridded Land Use are not identified explicitly in ISO/IEC 5087-2 but can be captured with different spatial representations of the area being described. Unlike INSPIRE, the spatial plans from which some zoning originates are not captured in ISO/IEC 5087-2.
building:Building	AbstractBuilding, BuildingAndBuildingUnitInfo,	In INSPIRE, AbstractBuilding (subclass of AbstractConstruction): allows for the representation of buildings and building parts. Like the Building class in ISO/IEC 5087-2, it captures information such as building use and dwellings, but represents only a count of these features, whereas building:Building represents the actual relationships (e.g. between a person and a dwelling). INSPIRE supports extended representations of an abstract building as 2-dimensional or 3-dimensional geometries. It also provides an extended representation (BuildingAndBuildingUnitInfo) that includes address representation and detailed physical characteristics such as materials and other building features (e.g. chimneys). This is not included in the ISO/IEC 5087-2 definition of Building.