Chapter 5

A Logical Design Pattern for Change Over Time: Applications in Transportation Planning

Megan Katsumi, University of Toronto, Canada
Mark Fox, University of Toronto, Canada

5.1. Introduction

Ontologies may be identified as one of two types: those designed to capture static domains and those designed to capture dynamic domains. Whether a domain is considered to be static or dynamic amounts to a question of the intended use of the ontology and (consequently) the required scope. In some cases the need to represent the dynamic aspects of domain is obvious, such as in an industrial setting where manufacturing activities and their effects are concerned. In other cases the requirement may be more subtle, such as a need to capture past and present information about some object and thus some representation of its change over time. Such requirements are commonplace, so the question of how to represent changing objects is an important one.

A review of existing solutions for representing change over time in OWL revealed several approaches, but no guidance for the application of these solutions in practice. To facilitate the correct and consistent adoption of such a solution, a logical design pattern for representing change in OWL was developed and presented in [1]. In this chapter, an extended version of this work is presented. It includes and builds on the original material, describing how the logical design pattern played an important role in the design of a set of ontologies for transportation planning.

It is important to distinguish between capturing changes in a domain's objects, and capturing changes in the domain (or the understanding of the domain) itself. This work focuses on changes in domain *objects*, in other words: changes to the individuals that are represented with the ontology. Changes in the domain itself, corrections or revisions to how the domain is represented, are the subject of areas such as ontology evolution and versioning [2] and are not in the scope of this work.

The remainder of this chapter is structured as follows: first, the need for a design pattern for change is motivated; an overview of existing solutions for representing changing objects in OWL is provided, and the choice of approach for this pattern is explained. Next, the logical design pattern is presented followed by detailed examples of how the pattern was applied in practice. The chapter concludes with a brief overview of several insights that have arisen since the pattern's initial publication .

5.2. Motivation: Change in Transportation Planning

The ability to represent how objects change over time is often critical to capture accurate definitions and to support the desired functionality of the ontology (e.g. queries) in a given domain. Consequently, the representation of change is a requirement in many ontology applications, one such case is in the area of transportation planning.

Transportation planning encompasses a range of research and analysis activities aimed to address problems in transportation policy and decision-making. A key objective is to assist municipalities in preparing for future transportation demands and goals in the urban system. This task extends beyond a basic assessment of travel supply and demand to include factors such as economic and environmental sustainability and city liveability. As a result, transportation planning activities consume and produce a range of diverse data sets related to the urban system, its characteristics, and its behaviour. These data are obtained from a variety of sources, such as government open data portals, transit agencies, university-led studies, and industry partners. The challenge of data silos throughout these data sets is well known, and it results in a considerable data processing overhead for any transportation planning activities.

To address the challenges of knowledge management in transportation planning an OWL ontology, the iCity Transportation Planning Suite of Ontologies (TPSO) [3] ,was developed as part of an inter-disciplinary and cross-institution project on urban informatics, iCity-ORF [4]. The iCity TPSO would later evolve into an ISO standards effort to address a need for an ontology for transportation planning and city data beyond the iCity-ORF project.

Early on in the development of the iCity TPSO, it was clear that change over time would play a major role in the representation of the urban system. Many analysis tasks involve examining the state of the urban system at different points in time (e.g. peak and off-peak traffic hours). This requires the capacity to represent the history of various aspects of the infrastructure, such as how the flow of traffic changes on some part of the road network. Implicit in this is the understanding that some of the objects in the domain will demonstrate changing properties over time. The fact is that change must be captured for *many* objects involved in transportation planning activities. What was needed was a design pattern to ensure consistent, correct representation of change throughout the domain.

The following use cases are drawn from research activities in the iCity-ORF project. They serve to illustrate the need for an ontology capable of formalizing changes in the domain's objects. Subsequent sections will outline how the proposed logical design pattern adopted by the iCity TPSO is implemented to address the representation requirements for each of the use cases.

5.2.1. Traveller Information System

The tremendous amount and diversity of data generated by Intelligent Transportation Systems (ITS) has become an important resource for transportation services and applications. Travelling from one place to another often involves different information from different ITS services. Unfortunately, the multiplicity of ITS and their complexity has produced a body of heterogeneous data that cannot easily be integrated. Data from different sources must be analysed, classified and re-organized into a homogeneous format in order to be usable.

Many institutions and companies have developed Information Communication Technology (ICT) solutions to manage data integration and representation by using well-known industrial protocols and data specifications like the General Transit Feed Specification (GTFS) [5]. Nevertheless, these solutions lack a formal semantics; there is no common standard across systems to manage and exchange data and information.

ITS tools require integration of many heterogeneous data sources. Adaptability is challenging for traditional frameworks due to the overhead to integrate new and changing data sources. To address this challenge, an architecture has been designed to support scalable and extensible ITS applications using a semantic representation and integration. The ITSoS architecture, originally proposed by [6], is intended to leverage ontologies to support data integration. In general, the range of queries required to support an implementation of the ITSoS architecture will vary greatly as a function of the ITS application(s) to be supported. In the iCity-ORF project, the ITSoS architecture was utilized to develop the Advanced Traveler Information System (ATIS). To support this application, the iCity TPSO was required to capture data regarding the traffic status data on various road segments in the transportation network. The following are some example competency questions that were identified in the iCity project, based on this use case:

CQ1-1: What are the TTI_{Max} values that have been observed over some period of time?

CQ1-2: What are the TTI_{Max} values that have been observed at some location?

CQ1-3: What are the TTI_{Max} values that have been observed at some location, over some period of time?

In the above questions TTI_{Max} refers to the Maximum Transportation Travel Index. TTI_{Max} is a measurement used to indicate traffic conditions by way of a comparison of the observed rate of travel to the maximum throughput speed on a road segment. Central to this use case is the association of the index with some part of the transportation network, and the need to capture the changes in the index over time.

5.2.2. Land Use and Travel Simulation

The Integrated Land Use, Transportation, Environment (ILUTE) model [7] is an agent-based microsimulation that evolves land use, population economics and demographics over time. A travel model, such as the Travel/Activity Scheduler for Household Agents (TASHA/GTAModel) [8], can then be used to simulate travel behaviour for the future-state output by ILUTE. Using the future-state land use and population data (including factors such as occupation, age, and household composition) output from the ILUTE simulation, the travel model generates a set of activities and associated travel requirements for each member of the population. Based on this, these models are capable of simulating typical travel behaviour for an entire (future state) city. These types of simulations involve a vast amount of data that can be analysed in a variety of ways. Reviewing and analysing the results of large scale micro-simulations, such as those generated by the ILUTE and TASHA models, is a challenging task.

The ontology can be used to capture models and simulation output and support question-answering to explore the results. Maintaining the data that serves as input to these simulations also poses a challenge for researchers. The ontology may also be used to address this by capturing and relating historical data to improve access for researchers. The following are some example competency questions that were identified in the iCity-ORF project, based on this use case:

CQ2-1: What is the occupation breakdown of the travellers whose trips originated/ended in a given zone, during a given time period?
CQ2-2: What were the purposes of the trips that originated/ended in a given zone, during a given time period?
CQ2-3: What is the age and occupation of the traveller who performed a particular trip?
CQ2-4: What land use classification is associated with a particular parcel at a given point in time?

Again, a key aspect of this use case is that the individuals of interest (persons, parcels of land, and so on) are subject to change over time. Certain properties, such as a person's occupation or the land use designation assigned to some parcel of land, will likely evolve throughout a simulation as well as in real-world historical data.

5.3. Background

The task of representing change over time in OWL has been addressed by way of the so-called N-ary relations approach [9], and the 4-dimensional (4D) approach introduced by Welty, Fikes, and Makarios [10].

In a traditional 3D approach a single relation can describe the position of two blocks at various points in time: $\mathsf{on}(\mathsf{A},\mathsf{B},T_1)$, to describe that A is "on" B at time T_1. The N-ary

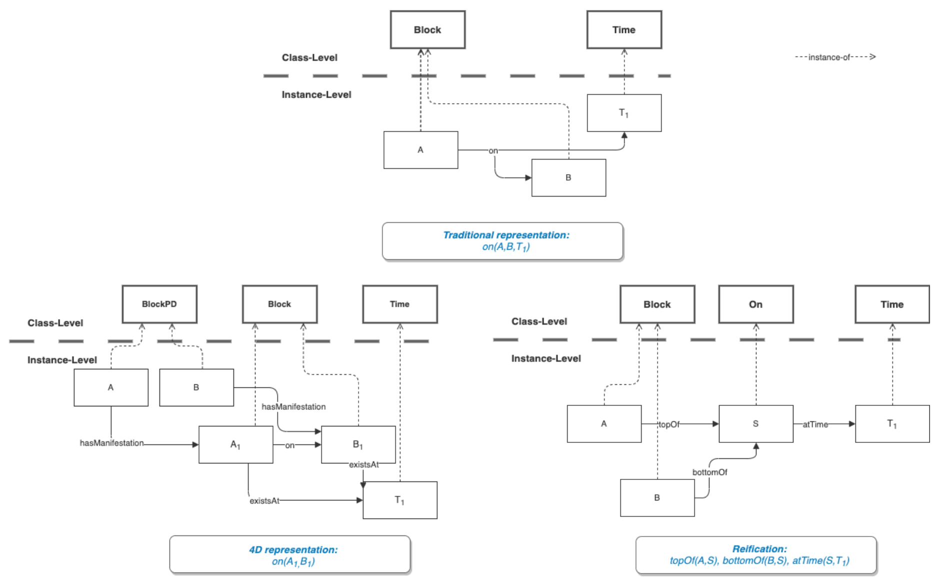


Figure 1. Example of 4D and reification (N-ary) approaches for representing the fact that Block A is on Block B at time T_1.

relations approach amounts to reification of the fluent property. The relation “on” becomes an object in the domain, and a class is introduced to capture instances of the relation: On(S). Three new relations are now required to capture the relationship between A,B,T_1, and On; for example: topOf(A,S), bottomOf(B,S), and holdsFor(S,T_1). A comparison of the approaches is illustrated in Fig. 1.

In an empirical study comparing representations of temporal information by Scheuermann and colleagues [11], the authors concluded that the N-ary relations representation was the more intuitive and most widely chosen representation approach to model a particular statement. Despite this, in practice the *representations* resulting from the N-ary approach may or may not be intuitive; this depends both on the fluent being captured as well as the skill of the designer. It is worth noting that Scheuermann’s survey also indicated that the 4D approach was preferred by participants with a higher level of knowledge representation expertise, likely due to its technical advantages. The advantages of the 4D pattern over the N-ary relations approach from a representation and reasoning perspective were also shown quantitatively in a comparison by Gangemi and Presutti [12] (in this work, the authors refer to the N-ary approach as the Situations pattern).

The 4D approach for OWL was first introduced with the 4D Fluents ontology presented by Welty, Fikes and Makarios [10]. The authors propose a compact, reusable ontology that enables a representation of fluents by adopting a 4D view. This avoids the complication of capturing N-ary relations, and leads to the rather nice, concise axiomatization. While the paper does include a brief example, the focus is on the approach itself, rather than its implementation.

The 4D approach was later re-interpreted by Krieger [13]. In this reinterpretation, all of the concepts that were originally interpreted as entities are re-interpreted as temporal parts (so-called "time slices") and a class of *perdurant*[1] individuals is introduced that has these entities as temporal parts. In other words, where the original 4D approach would have us refer to some special class of individuals, BlockAtT, that are temporal parts a particular entity of some class, Block, in Kreiger's approach we consider the Block class to be comprised of individuals that are temporal parts of some *Perdurant* – in this case, the "process" or lifespan of some Block. This rather simple switch results in several advantages that are discussed in greater detail by Krieger [13]. Of particular importance is the fact that this approach more easily supports the reuse of existing, atemporal ontologies. In the context of the iCity project, there were many relevant domain ontologies that were atemporal, but still desirable for reuse. As in the majority of ontology design projects, reuse of existing ontologies was an important consideration, both to simplify the development efforts and to create opportunities for linked data. Owing to these advantages, the logical design pattern presented here is based on Krieger's re-interpreted 4D approach.

The representation of change over time is addressed in several other places in more recent work in the form of design proposals [15], as well as new ontologies [16, 17]. The 4-dimensionalist approach, as well as the N-ary relations approach are captured in the two proposals presented by Zamborlini and Guizzardi [15]. In the same spirit as this work, authors aim to provide a framework for the different approaches in order to support decisions when modelling changing information in OWL. This presentation is very well grounded in formal ontology and is unquestionably a useful resource. Unfortunately, there is often a disconnect between formal and applied ontology, such that the application of this work may prove difficult for many Semantic Web practitioners. To make the representation of change over time more accessible, guidance that is more instructive and that may be more readily applied to existing ontologies is required.

The Global Legal Entity Identifier Ontology (GLEIO) [16] is a more recent effort, developed to provide semantics for the XML representations of legal entity identifier (LEI) data. The ontology does not extend the 4D Fluents ontology, but implements a similar approach that distinguishes between what they introduce as "variable" and "non-variable" entities in order to capture the changes that occur to LEI data over time. The semantics specified in GLEIO are instead drawn from other work by Moltmann [18]. Some documentation is also available in the GLEIO whitepaper [19], however its focus is (naturally) on GLEIO's application for its intended use in the legal identifier domain. No guidance or intention for the general application of the representation approach in other domains is provided.

Part 12 of the International Standard of Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities (ISO 15926) presents an upper ontology that adopts the 4D view [17]. This solution is embedded in an upper ontology which may or may not be a desirable design tool for certain tasks, thus its applicability is limited to those cases where the upper ontology is a suitable solution.

[1]The concept of a perdurant is one of two key concepts that correspond to distinct philosophical views of the world: perdurantism and endurantism, (or 4D and 3D, respectively). In the perdurantist view no entity is ever wholly present at some point in time, and so a perdurant represents the entire entity as it is extended through time. This terminological distinction is attributed to Lewis [14]. A detailed review of these concepts is out of the scope of this work.

Overall, there is a lack of pragmatic guidance to support adoption of the 4D approach in ontology design. Despite this, when defining temporal concepts, whether from scratch or by reusing existing ontologies, a repeatable solution can be recognized. This solution is distilled here in the form of a logical design pattern for the implementation of a 4D approach to representing change over time.

5.4. Representing Change 101

One of many temporal concepts required in the iCity ontology is that of a vehicle. There exist a variety of Semantic Web ontologies that define this concept but do not capture the possibility of changes that occur to a vehicle over time. Instead, an ontology will typically provide static definitions of the vehicle concept, describing properties such as the manufacturer, vehicle identification number (vin), colour, number of doors, type of engine, and so on. A simple example of a representation found in an existing ontology might be as follows:

$$\mathsf{Vehicle} \sqsubseteq {=}1\mathsf{hasVin.Vin}$$

$$\mathsf{Vehicle} \sqsubseteq \forall\mathsf{hasColour.Colour}$$

$$\mathsf{Vehicle} \sqsubseteq {=}1\mathsf{hasMake.Manufacturer}$$

Some of these properties may be subject to change over time (e.g. colour) while others may be static, however this is not identified in the definition. The same is true for many domain ontologies; concepts are defined but the possibility of their properties changing is not acknowledged. This is problematic when the concept of time plays a role in an intended application, as some aspects of a concept (e.g. a Vehicle) will change over time.

The following sections outline how an ontology with an atemporal definition such as the one above may be modified to capture change over time. First, an Ontology of Change that introduces the basic concepts of manifestations and perdurants is imported, then the logical design pattern is applied. Although the pattern does include some signature from the Ontology of Change, it is identified as a Logical Design Pattern [20] because it is independent of a particular domain; much of its signature is empty, containing place-holders for classes and properties and thus it provides a logical structure rather than specific content. This process may also be applied to define temporal concepts from scratch.

5.4.1. A Minimal Ontology of Change

The Foundational Ontology of Change[2] adopts the re-interpreted 4D view proposed by Krieger [13]. It is important to emphasize that the ontology itself is not the focus of

[2] http://ontology.eil.utoronto.ca/icity/Change

this contribution. It is a minimal set of axioms designed to provide a basis upon which temporal representations for the various concepts in the domain may be built. Should it be required, the guide prescribed here could easily be followed with some alternative, possibly stronger, foundational ontology of change.

The ontology introduces two key classes: TimeVaryingEntity and Manifestation[3]. A TimeVaryingEntity corresponds to the invariant part of a concept that is subject to change. As per Krieger, a TimeVaryingEntity is viewed as a perdurant. A TimeVaryingEntity has Manifestations that demonstrate its changing properties over time. The class TimeVaryingEntity is equivalent to the class of things that have some Manifestations - and *only* Manifestations - in the hasManifestation relation. A Manifestation is a snapshot of some TimeVaryingEntity, this relationship is formalized with the manifestationOf relation (defined as a functional property). A Manifestation exists at some Instant (possibly Interval) in time during which the TimeVaryingEntity exists.

In addition to recognizing the manifestationOf relationship, it is useful to recognize when two manifestations are of the same TimeVaryingEntity. This relationship is captured with the sameTimeVaryingEntity property, which is defined through object property chaining as follows:

$$\mathsf{manifestationOf} \circ \mathsf{inverse}(\mathsf{manifestationOf}) \rightarrow \mathsf{sameTimeVaryingEntity}$$

Naturally, some ontology of time is required for this representation. In the implementation described here, the W3C's Time Ontology [21] was selected owing mostly to its prevalence and comprehensiveness. However, it should be noted that this work does not rely on its use and so other theories of time might easily be substituted.

5.4.2. Implementation: A Logical Design Pattern for Change Over Time

To apply the Change of Time Varying Entities[4] logical design pattern requires that the designer import the Ontology of Change into the domain ontology being developed and perform the steps outlined in Fig. 2. The first steps define (or extends) the class to be a subclass of Manifestation. As a result, the class is now interpreted as having some temporal extent (i.e. it exists over some point or interval in time), and is a manifestation of some TimeVaryingEntity. From the previous vehicle example, we would have: Vehicle $\sqsubseteq$ Manifestation.

For each new subclass of Manifestation, the designer must now define its perdurant counterpart. This new class (introduced in Step 4) is a subclass of the TimeVaryingEntity class. It captures the invariant aspects of each concept, and is associated with a collection of instances of the Manifestations class that captures how it changes over time. For a vehicle, we might call the class defined in Step 4 "VehicleTVE" or "VehiclePD" for

[3]The ontology defines new set of terms to avoid confusion with other representations, as well as to improve the understandability for the related project on urban informatics. Further discussion of this design choice is outside of the scope of this work.

[4]http://ontologydesignpatterns.org/wiki/Submissions:Change_of_Time_Varying_Entities

For each new class (or existing class, if an ontology with an atemporal representation is being reused), $\underline{\mathsf{C}}$, that is subject to change:

1. If starting from scratch, define atemporal axiomatizations for all properties of $\underline{\mathsf{C}}$ as required.
2. Distinguish between the *variant* and *invariant* properties of $\underline{\mathsf{C}}$.
3. Define the concept $\underline{\mathsf{C}}$ as a subclass of Manifestation.

   Axiom Type 1. $\underline{C} \sqsubseteq Manifestation$
4. Define the perdurant (TimeVaryingEntity) counterpart class for the concept.

   Axiom Type 2. $\underline{PD_C} \sqsubseteq \texttt{TimeVaryingEntity}$
5. Include any invariant properties from the Manifestation subclass in the axioms for the TimeVaryingEntity subclass. Where $\underline{\mathsf{invariantProperty}}$ is the invariant property, $\underline{\mathsf{X}}$ is the restriction, and $\underline{\mathsf{CE}}$ is the class expression:

   Axiom Type 3. $\underline{PD_C} \sqsubseteq \underline{X}\ \underline{invariantProperty}.\underline{CE}$

   Exception: If the object in the class expression *also* subject to change (i.e., a subclass of Manifestation), then Axiom Type 3 is applied with the TimeVaryingEntity subclass in place of the Manifestation subclass class in the class expression ($\underline{\mathsf{CE}}$).
6. Restrict the hasManifestation relationship for this new pair of TimeVaryingEntity and Manifestation subclasses.

   Axiom Type 4. $\underline{PD_C} \equiv \exists\ hasManifestation.\underline{C}$
   $\sqcap\ \forall\ hasManifestation.\underline{C}$

   Axiom Type 5. $\underline{C} \equiv \exists\ manifestationOf.\underline{PD_C}$
   $\sqcap\ \forall\ manifestationOf.\underline{PD_C}$

Underlined terms indicate place-holders for domain-specific properties and classes to be specified when implementing the pattern: $\underline{\mathsf{PD_C}}$ denotes the class to define invariant part of the concept that is subject to change, and $\underline{\mathsf{C}}$ denotes class to define the variant part of the concept that is subject to change.

Figure 2. The process to apply the Logical OP for Change for a given concept.

readability to make its role as the time varying class (as opposed to the manifestation subclass) clear. The new class is defined with the following statement: $\mathsf{VehiclePD} \sqsubseteq \mathsf{TimeVaryingEntity}$.

Any property that cannot change over time (i.e. is *invariant*) should not only be a property of the Manifestation, but also a property of the *entire* TimeVaryingEntity (perdurant), whereas a property that may change with time should only be defined as a property of the Manifestation. Therefore, any properties that were originally defined in the atemporal class representation remain properties of the class (now, a Manifestation), and a *subset* of these properties will define the perdurant class. Precisely which properties these are is, in the end, an ontological decision for the designer. For example, the vin of a Vehicle should (in most cases) not change, and so it should be a property of both Vehicle and VehiclePD; this type of property is referred to as *invariant*. On the other hand, the colour of a car may change over time and so while it may be a property of Vehicle, it is not a property of VehiclePD. Step 5 includes these invariant properties in the axioms for the TimeVaryingEntity subclass. This is captured by applying the original axioms for each invariant property. For hasVin, the result is: $\mathsf{VehiclePD} \sqsubseteq = 1\ \mathsf{hasVin.Vin}$. The hasMake property is also invariant, however a Manufacturer may be subject to change. In this case, we apply the exception of Step 5 for the hasMake property: $\mathsf{VehiclePD} \sqsubseteq = 1$ $\mathsf{hasMake.ManufacturerPD}$.

The Change Ontology recognizes a constraint that any TimeVaryingEntity (and only a TimeVaryingEntity) should have manifestations (and only manifestations) in the hasManifestation relation, and vice versa for any Manifestation belonging to a TimeVaryingEntity. Further, any individual Manifestation must have some (unique) individual TimeVaryingEntity that it is a manifestation of, and conversely any individual TimeVaryingEntity must have some individual Manifestations. Similar constraints should be specialized for all corresponding pairs of Manifestation and TimeVaryingEntity subclasses. This is enforced by Step 6. For the classes Vehicle and VehiclePD, the result is: $\mathsf{VehiclePD} \equiv \exists \mathsf{hasManifestation.Vehicle} \sqcap \forall \mathsf{hasManifestation.Vehicle}$ and
$\mathsf{Vehicle} \equiv \exists \mathsf{manifestationOf.VehiclePD} \sqcap \forall \mathsf{manifestationOf.VehiclePD}$.

By applying the pattern proposed here, the atemporal representation of the Vehicle class considered initially is easily transformed to a temporal representation that captures changes that may occur in the domain. It should now be clear why adopting the approach by Krieger is advantageous for the reuse of domain ontologies. Any existing, atemporal ontologies might be reused with this approach, requiring only that we add to rather than rename or manipulate the existing axioms. While this example only considers the concept of a Vehicle, in practice any number of concepts can be adapted from existing ontologies or defined from scratch in the same way. Observe that only 6 additional axioms were required in this example to apply the pattern to modify the original representation. In general, there will necessarily be 4 axioms (Axiom Types 1,2,4,5), with additional axioms for each invariant property.

Note that this pattern does not include specialized notions of change such as those required to capture activities and resource consumption. Certainly, other useful types of properties may be identified and captured in ODPs, however these would be best pursued as separate efforts in order to maintain the simplicity of the representation of change presented here.

5.5. Additional Semantics for Change

Applying the pattern in the Sec. 5.4.2 achieves a straightforward representation of change in a given domain. In some cases, an extension to this representation may be desired. Capturing additional semantics provides a better understanding of the domain and may support various interesting reasoning tasks such as entity recognition, property inheritance, and consistency checking, described below:

Entity recognition: recognizing what entity some individual is a manifestation of. As an example in the context of urban informatics: given a dataset of vehicle locations where each observation corresponds to a Vehicle, and implicitly a Vehicle PD. Can we identify which VehiclePD an observation corresponds to, or recognize when two observations refer to the same VehiclePD?

Property inheritance: inferring the inheritance of properties between an entity and its manifestations. This can be useful in cases of incomplete information.

Consistency checking: determining whether the assertion that a manifestation corresponds to some entity is correct (or possible). Perhaps the record of some Vehicle has been incorrectly annotated as a manifestation of a different (incorrect) VehiclePD; can this be detected?

The semantics required for these tasks cannot be enforced in OWL directly, due to limitations in its expressive capabilities. Instead, the design pattern is extended in SWRL [22] to indicate of how the ontology may be extended (by rules or other means) should the intended application require it.

A deeper semantics of change may be specified by more closely considering the properties' behaviour *relative* to a particular concept. In this context, there are two distinct types of properties: invariant and variant. It is straightforward to see that a property's type is defined relative to a particular concept. For example, a property such as height may be invariant for a table, but variant for a person. Identifying these property types is based on the semantics of the concept; in practice, this is an ontological decision for the designer. Instructive questions to ask when making this assessment are: *Can the value of this property change? If the value of the property changes, is it still the same thing (e.g. is it still the same vehicle)?*

These property types are reminiscent of the meta-properties introduced by Guarino and Welty's OntoClean [23]. It is important to first make the distinction that the "properties" described in OntoClean are in fact not the same as the properties discussed here. The notions of rigidity and identity are ascribed to *meanings* of expressions, and thus they are more general than the property types described here that apply to properties of classes in OWL. Invariant properties are analogous to being essential for a particular Class ("property"), whereas the variant properties might be thought of as non-rigid, as they are not essential for all of the instances of things which they are properties of. However, there is no direct mapping between these property types as the notions of invariant and variant used here are defined relative to a particular class, while essence and rigidity are not.

5.5.1. Invariant Property

The pattern presented previously recognizes a type of property that is not subject to change over time for an object, termed *invariant*. An invariant property holds for an object, independent of time. This means that an invariant property is a property of the TimeVaryingEntity class, and must be inherited by all of its Manifestations. Capturing this semantics supports the reasoning task of property inheritance described earlier. It is formalized as follows, where again, the underlined terms are place-holders for domain-specific properties and classes to be specified when implementing the pattern. The following axiom types are restricted to invariant properties that do not have a TimeVaryingEntity or Manifestation as the object of the invariant property. An axiom type to address that type of relationship will follow.

Axiom Type 6. $\underline{\mathit{PD_C(?x)}}$, $\mathit{manifestationOf(?x_t,?x)}$, $\underline{\mathit{invariantProperty_C(?x,?y)}}$, $\mathit{(not(TimeVaryingEntity))(?y)} \rightarrow \underline{\mathit{invariantProperty_C(?x_t,?y)}}$

Axiom Type 7. $\underline{\mathit{PD_C(?x)}}$, $\mathit{manifestationOf(?x_t,?x)}$, $\underline{\mathit{invariantProperty_C(?x_t,?y)}}$, $\mathit{(not(Manifestation))(?y)} \rightarrow \underline{\mathit{invariantProperty_C(?x,?y)}}$

The vin is an example of an invariant property for a Vehicle. Axiom Types 6 and 7 may be applied to capture this as follows:

- VehiclePD(?x), manifestationOf(?x_t,?x), hasVin(?x,?y), (not(TimeVaryingEntity))(?y) $\rightarrow$ hasVin(?x_t,?y)
- VehiclePD(?x), manifestationOf(?x_t,?x), hasVin(?x_t,?y), (not(TimeVaryingEntity))(?y) $\rightarrow$ hasVin(?x,?y)

Assuming that the design pattern is applied appropriately it is not necessary to include the clause regarding the nature of the object of the property (in this case, whether the vin is a time varying concept or not), however it is good practice to do so. Note that by omitting this clause the pattern can be expressed directly in OWL with the use of object property chaining, as follows: manifestationOf o $\underline{\text{invariantProperty}_C} \rightarrow \underline{\text{invariantProperty}_C}$

An alternative form to the above rules must be taken into account in the case that the object of the invariant property is *also* a time varying concept. In this case, the property holds with a perdurant individual (members of the TimeVaryingEntity class), and thus also between the corresponding manifestations. This results in an additional consideration with regard to the particular manifestations that are to be related: they must exist at the same time. The alternate rules are specified below.

Axiom Type 8. $\underline{\mathit{PD_C(?x)}}$, $\mathit{manifestationOf(?x_t,?x)}$, $\underline{\mathit{invariantProperty_C(?x,?y)}}$, $\mathit{manifestationOf(?y_t,?y)}$, $\mathit{existsAt(?x_t,?t)}$, $\mathit{existsAt(?y_t,?t)} \rightarrow \underline{\mathit{invariantProperty_C(?x_t,?y_t)}}$

Axiom Type 9. $\underline{\mathit{PD_C(?x)}}$, $\mathit{manifestationOf(?x_t,?x)}$, $\mathit{manifestationOf(?y_t,?y)}$, $\underline{\mathit{invariantProperty_C(?x_t,?y_t)}} \rightarrow \underline{\mathit{invariantProperty_C(?x,?y)}}$

Returning to the Vehicle example, a manufacturer is something that may be subject to change over time (consider: its employees, countries of operation, net worth, an so on). The hasMake property may be defined in more detail by applying the Axiom Types 8 and 9 to obtain the following rules:

- VehiclePD(?x), manifestationOf(?x_t,?x), hasMake(?x,?y), manifestationOf(?y_t,?y), existsAt(?x_t,?t), existsAt(?y_t,?t) $\rightarrow$ hasMake(?x_t,?y_t)
- VehiclePD(?x), manifestationOf(?x_t,?x), manifestationOf(?y_t,?y), hasMake(?x_t,?y_t) $\rightarrow$ hasMake(?x,?y)

An important characteristic of the 4D representation that is highlighted by the identification of these rules is that object properties that are inverses of one another in an atemporal representation will not necessarily be the inverse of one another here. Specifically, for an invariant property of a time varying entity, the “inverse” must be likewise invariant for the object of the relation in order for the two relations to be *defined* as the inverse of one another. Consider the example of the hasMake and manufacturerOf relations between a Vehicle and a Manufacturer. In an atemporal domain, these relations would likely be defined as inverse properties of one another. However, in a 4D view, this relationship is no longer desirable. While the hasMake relation is invariant for a Vehicle, the manufacturerOf relation is variant for a Manufacturer; Company X was not always the manufacturer of a particular Vehicle, so only some of a company’s manifestations should have this property. While it is still possible to refer to the inverse of the hasMake relation, the important point is that in the 4D view this inverse property is *not* equivalent to the manufacturerOf relation.

5.5.2. Invariant Property Key

All invariant properties can in some cases differentiate between distinct entities. For example, observations of a Vehicle at two points in time with distinct invariant properties must be different Vehicles. However, certain invariant properties also serve to support entity *recognition*; they identify not only when two entities are distinct, but also when they are the same. We refer to such properties as invariant property keys. OWL2 provides the HasKey construct to define the semantics of a key for a particular class. For all invariant property keys, we can specify axioms as follows, where $\underline{\text{invariantPropertyKey}_C}$ is any invariant property key for some concept C:

Axiom Type 10. $\underline{\textit{PD}_C}$ *HasKey* $\underline{\textit{invariantPropertyKey}_C}$

An invariant property key can indicate that an individual is a manifestation of a particular perdurant, and similarly when two individuals are manifestations of the same perdurant. Capturing this semantics supports the reasoning task of entity recognition described earlier. It can be specified with the following axiom types:

Axiom Type 11. $\underline{\textit{C(?x}_t\textit{)}}$, $\underline{\textit{invariantPropertyKey}_C\textit{(?x}_t\textit{,?n)}}$,
$\underline{\textit{PD}_C\textit{(?y)}}$, $\underline{\textit{invariantPropertyKey}_C\textit{(?y,?n)}}$, *not(TimeVaryingEntity(?n))*
$\rightarrow$ *manifestationOf(?x*$_t$*,?y)*

Axiom Type 12. $\underline{\textit{C(?x}_{t1}\textit{)}}$, $\underline{\textit{invariantPropertyKey}_C\textit{(?x}_{t1}\textit{,?n)}}$,
$\underline{\textit{C(?x}_{t2}\textit{)}}$, $\underline{\textit{invariantPropertyKey}_C\textit{(?x}_{t2}\textit{,?n)}}$,
not(Manifestation(?n)) $\rightarrow$ *sameTimeVaryingEntity(?x*$_{t1}$*,?x*$_{t2}$*)*

The vin of a Vehicle is an example of such a property. Axiom Types 10–12 can be applied to capture the following rules:

- VehiclePD HasKey hasVin
- Vehicle(?x), hasVin(?x,?n), VehiclePD(?y), hasVin(?y,?n), (not(TimeVaryingEntity))(?n) $\rightarrow$ manifestationOf(?x,?y)
- Vehicle(?x$_{t1}$), hasVin(?x$_{t1}$,?n), Vehicle(?x$_{t2}$), hasVin(?x$_{t2}$,?n), (not(Manifestation))(?n) $\rightarrow$ sameTimeVaryingEntity(?x$_{t1}$,?x$_{t2}$)

Axiom types 11 and 12 are restricted to invariant property keys where the object of the property is not subject to change. As with general invariant properties, a slightly different rule is required in the case that the property relates two time varying concepts:

Axiom Type 13. $C(?x_t)$, $invariantPropertyKey_C(?x_t,?n_t)$, $PD_C(?y)$, $invariantPropertyKey_C(?y,?n)$, $manifestationOf(?n_t,n)$
$\rightarrow manifestationOf(?x_t,?y)$

Axiom Type 14. $C(?x_{t1})$, $invariantPropertyKey_C(?x_{t1},?n_{t1})$, $C(?x_{t2})$, $invariantPropertyKey_C(?x_{t2},?n_{t2})$, $sameTimeVaryingEntity(?n_{t1},?n_{t2})$
$\rightarrow sameTimeVaryingEntity(?x_{t1},?x_{t2})$

5.6. Applications for Transportation Planning

As discussed in Sec. 5.2, the requirement to represent change over time arose repeatedly in the transportation planning domain. This section revisits the motivating scenarios, providing a summary of real world examples where the design pattern was applied successfully to achieve an ontology design that would satisfy the requirements.

5.6.1. *Traveller Information System*

The logical design pattern was adopted to create a representation to capture the changing metrics associated with arcs in a model of the transportation system. An abridged description of an arc, as required to satisfy the traveller information system use case, is as follows:

> An Arc is a directed connection in the Network that enables transportation via a particular Mode(s) from one Node to another. An Arc begins and ends at some Node. ... An Arc may have some posted and/or free flow speed, and may have an associated Maximum Travel Time Index (TTI_{Max}) at a given point in time.

In the context of the iCity project, the start and end nodes of an arc were recognized as invariant properties: a change in either start or end node would be interpreted as a different arc altogether. On the other hand, the associated property describing the TTI_{Max} (among others) was recognized as a variant property. The following outlines the process followed from Sec. 5.4.2 along with the resulting axioms generated, an excerpt from the Transportation System Ontology[5]:

1. If starting from scratch, define atemporal axiomatizations for all properties of C as required.
   The atemporal axiomatization for the class Arc is defined (in part) as follows:

[5] http://ontology.eil.utoronto.ca/icity/TransportationSystem/

$$\mathsf{Arc} \sqsubseteq = 1\mathsf{startNode}.\mathsf{Node}$$

$$\mathsf{Arc} \sqsubseteq = 1\mathsf{endNode}.\mathsf{Node}$$

$$\mathsf{Arc} \sqsubseteq \forall \mathsf{hasTTI}.\mathsf{TTI}_{\mathsf{Max}}$$

2. Distinguish between the *variant* and *invariant* properties of C.
   Variant properties:
   - hasTTI

   Invariant properties:
   - startNode
   - endNode
3. Define the concept Arc as a subclass of Manifestation.

$$\mathsf{Arc} \sqsubseteq \mathsf{Manifestation}$$

4. Define the perdurant (TimeVaryingEntity) counterpart class for the concept.

$$\mathsf{ArcPD} \sqsubseteq \mathsf{TimeVaryingEntity}$$

5. Include any invariant properties from the Manifestation subclass in the axioms for the TimeVaryingEntity subclass.
   Since Node is also identified as a changing object in the domain, the exception applies for both invariant properties:

$$\mathsf{ArcPD} \sqsubseteq = 1\mathsf{hasStart}.\mathsf{NodePD}$$

$$\mathsf{ArcPD} \sqsubseteq = 1\mathsf{hasEnd}.\mathsf{NodePD}$$

6. Restrict the hasManifestation relationship for this new pair of TimeVaryingEntity and Manifestation subclasses:

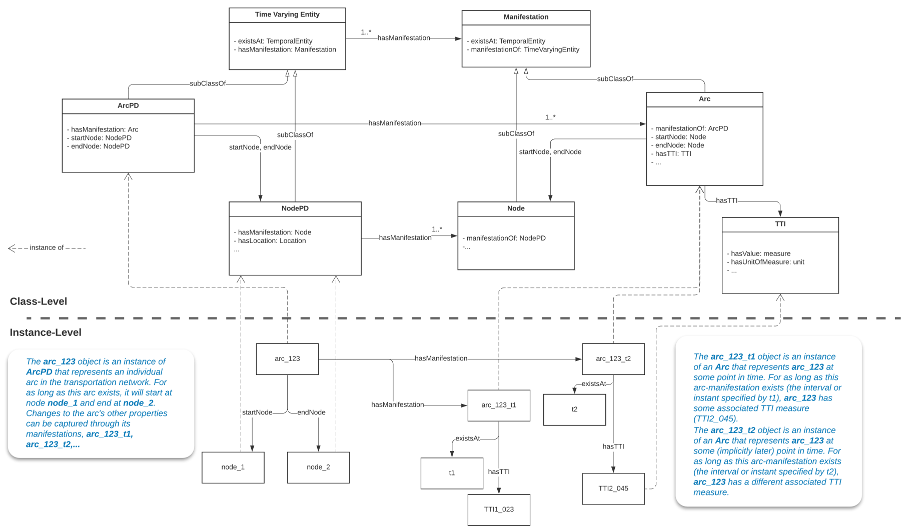
$$\mathsf{ArcPD} \equiv \exists \mathsf{hasManifestation}.\mathsf{Arc} \sqcap \forall \mathsf{hasManifestation}.\mathsf{Arc}$$

$$\mathsf{Arc} \equiv \exists \mathsf{manifestationOf}.\mathsf{ArcPD} \sqcap \forall \mathsf{manifestationOf}.\mathsf{ArcPD}$$

The resulting representation is depicted in Fig. 3

5.6.2. Land Use and Travel Simulation

The microsimulations used in the iCity project model activities at the individual, agent-level (i.e. the individual residents of a municipality). Capturing changes in population and land use over a long time horizon for a microsimulation means not only capturing changes to features of the city, but changes to the *individual* persons that comprise the city's population. The ODP for change was adopted to achieve a representation capable of capturing these changes over time.



*The **arc_123** object is an instance of **ArcPD** that represents an individual arc in the transportation network. For as long as this arc exists, it will start at node **node_1** and end at **node_2**. Changes to the arc's other properties can be captured through its manifestations, **arc_123_t1, arc_123_t2,...***

*The **arc_123_t1** object is an instance of an **Arc** that represents **arc_123** at some point in time. For as long as this arc-manifestation exists (the interval or instant specified by t1), **arc_123** has some associated TTI measure (TTI2_045). The **arc_123_t2** object is an instance of an **Arc** that represents **arc_123** at some (implicitly later) point in time. For as long as this arc-manifestation exists (the interval or instant specified by t2), **arc_123** has a different associated TTI measure.*

Figure 3. Excerpt of the 4D representation adopted for arcs in a transportation network.

In the iCity Ontology, a Person was defined as having some date of birth, age, occupation, and so on. Household membership and places of work and residence were also important attributes as they would inform subsequent activity-based travel simulations. A Person's date of birth was identified as an invariant property, whereas their household membership and employer were identified as variant properties. The following outlines the process followed from Sec. 5.4.2 along with the resulting (excerpt of) axioms generated for the Person[6] and Urban System[7] Ontologies:

1. If starting from scratch, define atemporal axiomatizations for all properties of Person as required.
   The atemporal axiomatization for the class Arc is defined (in part) as follows:

$$\mathsf{Person} \sqsubseteq \forall \mathsf{worksFor}.(\mathsf{Person} \sqcup \mathsf{Organization})$$
$$\mathsf{Person} \sqsubseteq \forall \mathsf{memberOf}.\mathsf{Household}$$
$$\mathsf{Person} \sqsubseteq = 1 \mathsf{birthDate}.\mathsf{Instant}$$

2. Distinguish between the *variant* and *invariant* properties of Person.
   Variant properties:
   - worksFor
   - memberOf

   Invariant properties:
   - birthDate

[6] http://ontology.eil.utoronto.ca/icity/Person/
[7] http://ontology.eil.utoronto.ca/icity/UrbanSystem/

3. Define the concept Person as a subclass of Manifestation.

$$\mathsf{Person} \sqsubseteq \mathsf{Manifestation}$$

4. Define the perdurant (TimeVaryingEntity) counterpart class for the concept.

$$\mathsf{PersonPD} \sqsubseteq \mathsf{TimeVaryingEntity}$$

5. Include any invariant properties from the Manifestation subclass in the axioms for the TimeVaryingEntity subclass.

$$\mathsf{PersonPD} \sqsubseteq = 1 \mathsf{birthDate.Instant}$$

6. Restrict the hasManifestation relationship for this new pair of TimeVaryingEntity and Manifestation subclasses:

$$\mathsf{PersonPD} \equiv \exists \mathsf{hasManifestation.Person} \sqcap \forall \mathsf{hasManifestation.Person}$$

$$\mathsf{Person} \equiv \exists \mathsf{manifestationOf.PersonPD} \sqcap \forall \mathsf{manifestationOf.PersonPD}$$

5.6.3. Query Patterns

Owing to the consistent representation enabled by the ODP, a pattern for formalizing queries for retrieving temporal information was easily be recognized and employed for a number of scenarios. Three basic types of queries are identified here, however many other patterns are certainly possible. Example formulations of each query type are provided in SPARQL [24], and assume the following prefixes definitions:

- `PREFIX change:` ⟨http://ontology.eil.utoronto.ca/icity/Change/⟩
- `PREFIX time:` ⟨http://www.w3.org/2006/time/⟩

Query Pattern 1: 'get variant property values' This type of query is concerned with retrieving all known values of a given property. In certain contexts, this might be useful to provide a historical view of an object's change over time. The most basic form of this query is as follows, where `{object}` represents the object and `{variantProperty}` represents the variant property in question:

```
SELECT  ?val ?t
WHERE {
{object} change:hasManifestation ?x_t.
{object} change:existsAt ?t.
?x_t {variantProperty} ?val.
}
```

Query Pattern 2: 'get variant property values at a specific time' This type of query is concerned with retrieving the value of a variant property at a specific point (or interval) in time. The basic form of this query is as follows, where {`object`} represents the object, {`time`} represents the target time, and {`variantProperty`} represents the variant property in question:

```
SELECT  ?val
WHERE {
{object} change:hasManifestation ?x_t.
?x_t {variantProperty} ?val.
?x_t change:existsAt ?t_interval.
?t_interval time:hasBeginning ?t1.
?t1 time:inXSDDateTime ?dt1.
?t_interval time:hasEnd ?t2.
?t2 time:inXSDDateTime ?dt2.
FILTER(?dt <= {time}^^xsd:dateTime && ?dt2 >= {time}^^xsd:date
Time)
}
```

Query Pattern 3: 'get the time at which a specific variant property value holds' This type of query is concerned with retrieving the time during which the object in question demonstrates a specific property value. The basic form of this query is as follows, where {`object`} represents the object, {`value`} represents the target property object, and {`variantProperty`} represents the variant property in question:

```
SELECT  ?t_interval
WHERE {
{object} change:hasManifestation ?x_t.
?x_t {variantProperty} {value}.
?x_t change:existsAt ?t_interval.
}
```

The competency question CQ1-2 identified in Sec. 5.2: *What are the TTI_{Max} values that have been observed at some location?* conforms to Query Pattern 1, and may be formalized accordingly. On the other hand, CQ1-1: *What are the TTI_{Max} values that have been observed at some point of time?* conforms to Query Pattern 2. Finally, consider CQ2-3 identified in the motivating scenario for land use and travel simulations: *What is the age and occupation of the traveller who performed a particular trip?*. Indirectly, this query also confirms to Query Pattern 2 in that it is concerned with retrieving a variant property value(s) that hold at a specific time, (the time during which a specific trip is performed). In practice, the exact form of the queries required for a given ontology will vary due to such additional considerations. In addition, the other ontological commitments made, such as the representation of time or units of measure, will result in variances the final form of the queries from one ontology to another. Nevertheless, the approach to access temporal data remains relatively consistent owing to the adoption of the ODP. More details on the competency questions identified and formalized for the iCity Project may be found in the full project report [3].

5.7. Discussion

The ODP for capturing change has proven to be a useful tool in the design of ontologies to support transportation planning. In the context of the iCity project, the adoption of the ODP supported a repeatable representation of temporal data that simplified the task of data encoding and access across the various motivating scenarios.

Applying this design pattern in practice led to two key observations regarding the importance of terminology. While the approach was easily explained and understood, the labels assigned to the Manifestation and Time Varying Entity subclasses sometimes caused confusion among the domain experts. The suffix “-PD” is employed to name Time Varying Entity subclasses in the examples given here, however in practice the context of the implementation and the background knowledge of the users should be considered if they will be expected to interact with the ontology’s terms. For example, a suffix or prefix such as “Static-” or “Invariant-” might be more intuitive. In addition, it is important to clarify that adopting this approach need not require the adoption of a 4D *philosophy* for the domain as a whole. The intent of this pattern is only to adopt of 4D *approach* to support the representation of change as required for a given domain.

As discussed, the ODP presented here is a logical design pattern, thus unlike content design patterns which may be reused directly additional content such as the steps outlined in Fig. 2 are required to describe how the pattern should be adopted. Generic Ontology Design Patterns (GODPs), presented in further detail by Krieg-Brückner and colleagues in this volume [25], provide a more concise, formal approach to this. The design pattern for change is formalized as an example GODP, highlighting commonalities between this design pattern and others. This approach is also beneficial in that it enables an explicit representation of the design pattern, thus avoiding possible misinterpretations of the approach described in Sec. 5.4.2.

5.8. Conclusion

Change over time is an undeniable aspect of many areas, however there are few examples of domain ontologies capable of capturing change. This work provides an important resource for Semantic Web practitioners: a logical design pattern to support the capture of change in the design of new and (redesign of) existing theories. Beyond this, the pattern provides a guide for possible extensions to the ontology. This work also serves as a useful resource to introduce and familiarize non-ontologists with the representation of change. This is important when working with domain experts, such as the researchers in the urban informatics project that motivated this work.

The survey by Scheuermann and colleagues found the 4D approach to be lacking in intuitiveness. This pattern may help to address this; it is the intent of this work to facilitate the development of more expressive ontologies by simplifying the process of incorporating change over time and increasing its accessibility for a wider audience of

Semantic Web practitioners. By following the guidelines proposed here, the atemporal representation of a class is easily modified to account for change over time. This process was followed in the development of the iCity TPSO[8] for the iCity project. The Change design pattern provided a straightforward and uniform approach account for change over time throughout the ontology. As discussed in the previous section, the benefits of this approach extend beyond the task of ontology design to the task of query design. Overall, the consistent structure enabled by the ODP serves to improve ontology usability.

Acknowledgements

This work was supported by the Ontario Ministry of Research and Innovation through the ORF-RE program. We would also like to thank Bernd Krieg-Brückner and his co-authors of [25] in this volume, for their valuable feedback and helpful comments on this work.

References

[1] Katsumi M, Fox M. A logical design pattern for representing change over time in owl. In: 8th Workshop on Ontology Design and Patterns; 2017.

[2] Klein MC, Fensel D. Ontology versioning on the semantic web. In: SWWS; 2001. p. 75–91.

[3] Katsumi M, Fox M. icity transportation planning suite of ontologies. University of Toronto; 2020. Available from: http://ontology.eil.utoronto.ca/icity/iCityOntologyReport_1.2.pdf.

[4] Miller EJ. icity: Urban informatics for sustainable metropolitan growth; a proposal funded by the ontario research fund, research excellence, round 7. University of Toronto Transportation Research Institute; 2014.

[5] McHugh B. Pioneering open data standards: The gtfs story. Beyond transparency: open data and the future of civic innovation. 2013;:125–135.

[6] Elshenawy M, Abdulhai B, El-Darieby M. Towards a service-oriented cyber–physical systems of systems for smart city mobility applications. Future Generation Computer Systems. 2018;79:575–587.

[7] Miller EJ, Salvini PA. The integrated land use, transportation, environment (ilute) microsimulation modelling system: Description and current status. Travel behaviour research: The leading edge. 2001; :711–724.

[8] Roorda MJ, Miller EJ, Habib KM. Validation of tasha: A 24-h activity scheduling microsimulation model. Transportation Research Part A: Policy and Practice. 2008;42(2):360–375.

[9] Noy N, Rector A, Hayes P, et al. Defining n-ary relations on the semantic web ; 2006. Available from: https://www.w3.org/TR/swbp-n-aryRelations/.

[10] Welty C, Fikes R, Makarios S. A reusable ontology for fluents in owl. In: Formal Ontology in Information Systems (FOIS); Vol. 150; 2006. p. 226–236.

[11] Scheuermann A, Motta E, Mulholland P, et al. An empirical perspective on representing time. In: Proceedings of the seventh international conference on Knowledge capture; ACM; 2013. p. 89–96.

[12] Gangemi A, Presutti V. A multi-dimensional comparison of ontology design patterns for representing n-ary relations. In: SOFSEM 2013: Theory and Practice of Computer Science: 39th International Conference on Current Trends in Theory and Practice of Computer Science; Vol. 7741; Springer; 2013. p. 86.

[13] Krieger HU. Where temporal description logics fail: Representing temporally-changing relationships. In: Annual Conference on Artificial Intelligence; Springer; 2008. p. 249–257.

[14] Lewis D. On the plurality of worlds. Vol. 322. Oxford; 1986.

[8] http://ontology.eil.utoronto.ca/icity/UrbanSystem/

[15] Zamborlini V, Guizzardi G. On the representation of temporally changing information in owl. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International; IEEE; 2010. p. 283–292.

[16] Trypuz R, Kuzinski D, Sopek M. General legal entity identifier ontology. In: Formal Ontology in Information Systems (FOIS) Ontology Competition; 2016.

[17] Batres R, West M, Leal D, et al. An upper ontology based on iso 15926. Computers & Chemical Engineering. 2007;31(5):519–534.

[18] Moltmann F. Metaphysics, meaning, and modality. themes from kit fine. Oxford University Press (forthcoming); 2016. Chapter Variable Objects and Truthmaking.

[19] Robert Trypuz DK. Why use rdf/owl rather than xml to represent and share global legal entity identifiers (leis) and related lei reference data. MakoLab S.A. Semantic Web Research and Development Team; 2016.

[20] Gangemi A, Presutti V. Ontology design patterns. In: Handbook on ontologies. Springer; 2009. p. 221–243.

[21] Cox S, Little C. Time ontology in owl, 2017. URL: https://www w3 org/TR/owl-time. 2017;.

[22] Horrocks I, Patel-Schneider PF, Boley H, et al. Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission. 2004;21:79.

[23] Guarino N, Welty CA. An overview of ontoclean. In: Handbook on ontologies. Springer; 2009. p. 201–220.

[24] Harris S, Seaborne A. Sparql 1.1 query language. W3C; 2013. Available from: https://www.w3.org/TR/sparql11-query/

[25] Advances in pattern-based ontology engineering. IOS Press, Amsterdam; •. Chapter Generic Ontology Design Patterns: Roles and Change Over Time.