Katsumi, M., and Fox, M.S., (2017), "A Logical Design Pattern for Representing Change Over Time in OWL", Proceedings of the Workshop on Ontology Patterns, Vienna.

# A Logical Design Pattern for Representing Change Over Time in OWL

Megan Katsumi and Mark Fox

University of Toronto katsumi@mie.utoronto.ca
msf@mie.utoronto.ca

**Abstract.** While there exist OWL ontologies that capture events and acknowledge the dynamic nature of certain domains, the possibility of change is neglected in many domain-specific ontologies. Solutions to the representation of fluents have been proposed, however there exists no guidance for the average Semantic Web practitioner on how to incorporate these solutions into existing ontologies or implement them in the development of new ontologies. This paper addresses a gap in the literature on the representation of change over time in OWL through the introduction of a logical design pattern for representing change. The aim of this work is to make the representation of change more accessible to a broad audience of Semantic Web practitioners.

## 1 Introduction

This work is motivated by a project on urban informatics, iCity [11], in which our role is to develop an ontology capable of representing the urban system – both the information that is collected, as well as information that is simulated and analyzed by various research groups. Owing to its popularity, tool support, and role as the de facto standard for the Semantic Web, OWL2 was selected as the representation language for the formalization of the ontology. To capture the urban domain, the notion of change over time is a critical requirement: the population, family and household structures, transportation networks, and the locations of individual transportation vehicles (buses, household vehicles, and so on) are all subject to change.

Change over time plays a role in many domains, and is by no means a new research topic. In fact, several approaches for capturing change in OWL have been proposed [15, 12, 9]. Despite these solutions, we have found that Semantic Web practitioners currently lack clear and precise methods for how to apply these approaches to capture change at a domain level, whether reusing an atemporal ontology or developing an ontology from scratch. The work presented here aims to fill this gap by providing a straightforward, logical design pattern for implementing a representation of change in any given domain. In particular, we provide consideration for the reuse of atemporal ontologies, as our experience has led us to believe that this is an important design task.

## 2 Background

The task of representing change over time in OWL has been addressed by way of the so-called N-ary relations approach [12], and the 4D approach introduced by Welty, Fikes, and Makarios [15].

In a traditional 3D approach we might have a fluent that describes the position of two blocks at some point in time: on(A,B,t), to describe that A is "on" B at time t. In the N-ary relations approach, the relation "on" becomes a thing in the domain, and we introduce a class to capture instances of the relation: onClass(on1). Three new relations are now required to capture the relationship between A,B,t, and onClass; for example: topOf(A,on1), bottomOf(B,on1), and holdsFor(on1,t). A comparison of the approaches is illustrated in Figure 1.

In an empirical study comparing representations of temporal information by Scheuermann and colleagues [13], the authors concluded that the N-ary relations representation was the more intuitive and most widely chosen representation approach to model a particular statement. However, it is our view that the *representations* resulting from the N-ary approach may be intuitive or not, depending on the fluent they are capturing 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 [5] (in this work, the authors refer to the N-ary approach as the Situations pattern).

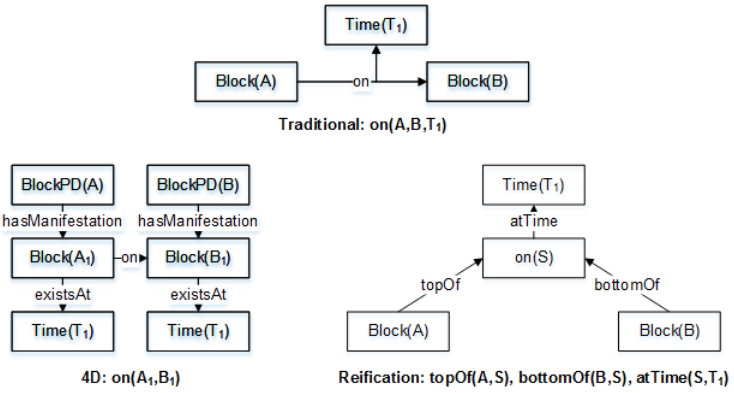


**Fig. 1.** Example of 4D and reification (N-ary) approaches for representing the fact that Block A is on Block B at time $T_1$.

The 4D approach for OWL was first introduced with the 4D Fluents ontology presented by Welty, Fikes and Makarios [15]. The authors propose a compact, reusable ontology that enables a representation of fluents by adopting the 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. This approach was later re-interpreted by Krieger [9]. 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 [9]. Of particular importance is the fact that this approach more easily supports the reuse of existing, atemporal ontologies. In the iCity project, we found that there were many relevant domain ontologies that were atemporal, but still desirable for reuse. Reuse is critical in this project, both to simplify our development efforts and to create opportunities for linked data, and we expect that this situation is not unique. Owing to these advantages, the logical design pattern presented here is based on the 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 [16], as well as new ontologies [14, 2]. Independent of the nature of the solutions, the issue existing work is that it fails to provide pragmatic guidance to support its adoption. When defining temporal concepts, whether from scratch or by reusing existing ontologies, we recognized a repeatable solution that we have distilled here in the form of a pattern for the implementation of a 4D approach to representing change over time.

## 3 Capturing Change 101

One of many temporal concepts required for 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. Rather, 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

[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 the entire entity as it is extended through time. This terminological distinction is attributed to Lewis [10]. A detailed review of these concepts is out of the scope of this work.

a representation found in an existing ontology might be as follows:

$$\mathsf{Vehicle} \sqsubseteq =1\mathsf{hasVin}.\mathsf{Vin}$$
$$\mathsf{Vehicle} \sqsubseteq \forall\mathsf{hasColour}.\mathsf{Colour}$$
$$\mathsf{Vehicle} \sqsubseteq =1\mathsf{hasMake}.\mathsf{Manufacturer}$$

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: a concept is defined but the possibility of its properties changing is not acknowledged. This is problematic when the concept of time plays a role in an intended application, as some aspects of the domain concepts (e.g. a Vehicle) will change over time.

In the following sections, we 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 [4] because it is independent of a particular domain; much of its signature is empty, containing placeholders for classes and properties and thus providing a logical structure rather than content. This process may also be applied to define temporal concepts from scratch.

### 3.1 A Minimal Ontology of Change

The Foundational Ontology of Change[2] adopts the re-interpreted 4D view proposed by Krieger [9]. It is important to emphasize that the ontology itself is not the focus of 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, 4D 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 of 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, existing at some Instant (possibly Interval) in time during which the TimeVaryingEntity exists.

[2] https://w3id.org/icity/iCity-Change

[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.

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:

$$\text{manifestationOf o inverse(manifestationOf)} \rightarrow \text{sameTimeVaryingEntity}$$

Naturally, some ontology of time is required for this representation. In the implementation, OWL-Time [7] 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.

### 3.2 Implementation: A Logical Design Pattern for Capturing 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 Figure 2 for each concept (or class, if an existing atemporal ontology is being reused), C, that is subject to change.

1. Define the concept C as a subclass of Manifestation.
   **Axiom Type 1** $\underline{Manifestation_C} \sqsubseteq Manifestation$
2. Define the perdurant (TimeVaryingEntity) counterpart class for the concept.
   **Axiom Type 2** $\underline{PD_C} \sqsubseteq \texttt{TimeVaryingEntity}$
3. Include any invariant properties from the Manifestation subclass in the axioms for the TimeVaryingEntity subclass. Where invariantProperty is the invariant property and CE is the class expression:
   **Axiom Type 3** $\underline{PD_C} \sqsubseteq \underline{invariantProperty}.\underline{CE}$
   *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 (CE).
4. Restrict the hasManifestation relationship for this new pair of TimeVaryingEntity and Manifestation subclasses.
   **Axiom Type 4** $\underline{PD_C} \equiv \exists\ hasManifestation.\underline{Manifestation_C}$
   $\sqcap\ \forall\ hasManifestation.\underline{Manifestation_C}$
   **Axiom Type 5** $\underline{Manifestation_C} \equiv \exists\ manifestationOf.\underline{PD_C}$
   $\sqcap\ \forall\ manifestationOf.\underline{PD_C}$

**Fig. 2.** The process to apply the Logical OP for Change for a given concept. Underlined terms indicate place-holders for domain-specific properties and classes to be specified when implementing the pattern: $\underline{PD_C}$ denotes the subclass of TimeVaryingEntity, and $\underline{Manifestation_C}$ denotes the subclass of Manifestation for the concept, C.

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

The first step defines (or extends) the class to be a subclass of Manifestation. As a result, the class now also has 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 of these new subclasses of Manifestation, the designer must now define its perdurant counterpart. This new class (introduced in Step 2) is a subclass of the TimeVaryingEntity class. It captures the invariant aspects of each concept, while the set of Manifestations captures how it changes over time. For a vehicle, we might call the class defined in Step 2 "VehiclePerdurant" or "VehiclePD" for 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: VehiclePD $\sqsubseteq$ TimeVaryingEntity.

It is straightforward to see that 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 can only be 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 not change, and so we might define this as a property of both Vehicle and VehiclePD; we refer to this type of property 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 reasonable that colour is not a property of VehiclePD. Step 3 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: VehiclePD $\sqsubseteq= 1$ 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 3 for the hasMake property: VehiclePD $\sqsubseteq= 1$ hasMake.ManufacturerPD.

The Change Ontology recognizes a constraint that any TimeVaryingEntity should have manifestations (and only manifestations) in the hasManifestation relation, and vice versa for any Manifestation belonging to a TimeVaryingEntity. Similar constraints should be specialized for all corresponding pairs of Manifestation and TimeVaryingEntity subclasses. This is enforced by Step 4. For the classes Vehicle and VehiclePD, the result is: VehiclePD $\equiv \exists$hasManifestation.Vehicle $\sqcap \forall$hasManifestation.Vehicle and
Vehicle $\equiv \exists$manifestationOf.VehiclePD $\sqcap \forall$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, as illustrated in Figure 3. 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.

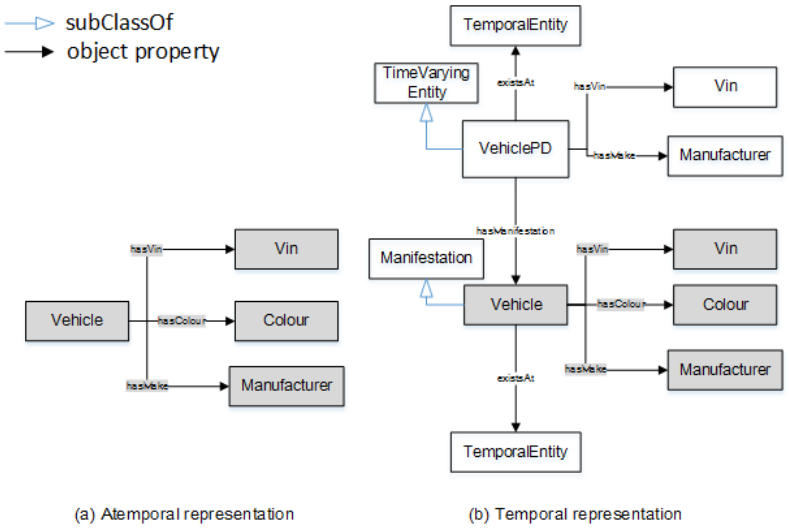


**Fig. 3.** A representation of the vehicle example (a) before and (b) after the guidelines are applied. Shaded classes and properties indicate reuse from the atemporal representation.

## 4 Additional Semantics for Change

Applying the pattern in the Section 3.2 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, from a dataset of vehicle locations where each observation corresponds to a Vehicle, but can we recognize specifically which Vehicle?

**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 in fact correct (or possible). Perhaps the record of some Vehicle has been incorrectly annotated as a manifestation of a different (incorrect) Vehicle; can we recognize this?

The semantics required for these tasks cannot be enforced in OWL directly, due to its representation limitations. Instead, the design pattern is extended in SWRL [8] to indicate of how the ontology can 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. Here, we recognize two distinct types of properties: invariant and variant. It is straightforward to see that a property's type is 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. Revealing 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 [6]. It is important to first make the distinction that the "properties" described in OntoClean are in fact not the same as the properties we discuss 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.

### 4.1 Invariant Property

The pattern presented previously recognizes a type of property that is not subject to change over time for a concept, termed *invariant*. An invariant property holds for a concept, 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, these underlined terms are placeholders 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** $\underline{PD_C(?x)}$, $manifestationOf(?x_t,?x)$, $\underline{invariantProperty_C(?x,?y)}$, $(not(TimeVaryingEntity))(?y) \rightarrow \underline{invariantProperty_C(?x_t,?y)}$

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

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) → 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 as a kind of sanity constraint. Note that by omitting this constraint, this 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** $\underline{PD_C(?x)}$, *manifestationOf(?$x_t$,?x)*, $\underline{invariantProperty_C(?x,?y)}$, *manifestationOf(?$y_t$,?y)*, *existsAt(?$x_t$,?t)*, *existsAt(?$y_t$,?t)* → $\underline{invariantProperty_C(?x_t,?y_t)}$

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

Returning to the Vehicle example, a manufacturer is something that may be subject to change over time (consider: it's 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) → hasMake(?$x_t$,?$y_t$)
– VehiclePD(?x), manifestationOf(?$x_t$,?x), manifestationOf(?$y_t$,?y), hasMake(?$x_t$,?$y_t$) → 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.

### 4.2 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** $\underline{PD_C}$ *HasKey* $\underline{invariantPropertyKey_C}$

**Axiom Type 11** $\underline{Manifestation_C}$ *HasKey* $\underline{invariantPropertyKey_C}$

For such properties we can identify when an invariant property key indicates that an individual is a manifestation of a particular perdurant, and similarly when it indicates that 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 12** $\underline{Manifestation_C(?x_t)}$, $\underline{invariantPropertyKey_C(?x_t,?n)}$, $\underline{PD_C(?y)}$, $\underline{invariantPropertyKey_C(?y,?n)}$, $\underline{not(TimeVaryingEntity(?n))}$ $\rightarrow$ $manifestationOf(?x_t,?y)$

**Axiom Type 13** $\underline{Manifestation_C(?x_{t1})}$, $\underline{invariantPropertyKey_C(?x_{t1},?n)}$, $\underline{Manifestation_C(?x_{t2})}$, $\underline{invariantPropertyKey_C(?x_{t2},?n)}$, $\underline{not(Manifestation(?n))} \rightarrow sameTimeVaryingEntity(?x_{t1},?x_{t2})$

The vin of a Vehicle is an example of such a property. Axiom Types 10–13 can be applied to capture the following rules:

- Vehicle HasKey hasVin
- 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 12 and 13 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 14** $\underline{Manifestation_C(?x_t)}$, $\underline{invariantPropertyKey_C(?x_t,?n_t)}$, $\underline{PD_C(?y)}$, $\underline{invariantPropertyKey_C(?y,?n)}$, $\underline{manifestationOf(?n_t,n)}$ $\rightarrow$ $manifestationOf(?x_t,?y)$

**Axiom Type 15** $\underline{Manifestation_C(?x_{t1})}$, $\underline{invariantPropertyKey_C(?x_{t1},?n_{t1})}$, $\underline{Manifestation_C(?x_{t2})}$, $\underline{invariantPropertyKey_C(?x_{t2},?n_{t2})}$, $\underline{sameTimeVaryingEntity(?n_{t1},?n_{t2})}$ $\rightarrow$ $sameTimeVaryingEntity(?x_{t1},?x_{t2})$

## 5 Conclusion

By following the guidelines proposed here, the atemporal representation of the Vehicle class is easily modified to account for change over time. This process was followed in the development of an initial version of an urban system ontology[5] for the iCity project. The Change design pattern allowed us to easily and uniformly account for change over time throughout the ontology. This was particularly useful for the reuse of domain ontologies to provide definitions for the concepts of Households, Persons, Organizations, and so on.

Change over time is an undeniable aspect of many areas, however there is little evidence 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. Certainly, other useful types of properties may be identified, and more rules could be pursued. This pattern does not cover specialized notions of change such as those required to capture activities and resource consumption.While we do not claim this extended semantics to be exhaustive, the provided axiom types support reasoning tasks of entity recognition and property inheritance, and all of these additional semantics serve to support more robust consistency checking.

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.

We gratefully acknowledge support provided by the Ontario Ministry of Research and Innovation through the ORF-RE program.

## References

1. Barrachina, J., Garrido, P., Fogue, M., Martinez, F.J., Cano, J.C., Calafate, C.T., Manzoni, P.: Veacon: A vehicular accident ontology designed to improve safety on the roads. Journal of Network and Computer Applications 35(6), 1891–1900 (2012)
2. Batres, R., West, M., Leal, D., Price, D., Masaki, K., Shimada, Y., Fuchino, T., Naka, Y.: An upper ontology based on iso 15926. Computers & Chemical Engineering 31(5), 519–534 (2007)
3. Bernhard Lorenz, Hans Jürgen Ohlbach, L.Y.: Ontology of transportation networks. Tech. rep. (2005)
4. Gangemi, A., Presutti, V.: Ontology design patterns. In: Handbook on ontologies, pp. 221–243. Springer (2009)
5. 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, p. 86. Springer (2013)

[5] https://w3id.org/icity/iCity-UrbanSystem/1.1

6. Guarino, N., Welty, C.A.: An overview of ontoclean. In: Handbook on ontologies, pp. 201–220. Springer (2009)
7. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Transactions on Asian Language Information Processing (TALIP) 3(1), 66–85 (2004)
8. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al.: Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission 21, 79 (2004)
9. Krieger, H.U.: Where temporal description logics fail: Representing temporally-changing relationships. In: Annual Conference on Artificial Intelligence. pp. 249–257. Springer (2008)
10. Lewis, D.: On the plurality of worlds, vol. 322. Oxford (1986)
11. Miller, E.J.: icity: Urban informatics for sustainable metropolitan growth; a proposal funded by the ontario research fund, research excellence, round 7. Tech. rep., University of Toronto Transportation Research Institute (2014)
12. Natasha Noy, Alan Rector, P.H.C.W.: Defining n-ary relations on the semantic web (2006), https://www.w3.org/TR/swbp-n-aryRelations/
13. Scheuermann, A., Motta, E., Mulholland, P., Gangemi, A., Presutti, V.: An empirical perspective on representing time. In: Proceedings of the seventh international conference on Knowledge capture. pp. 89–96. ACM (2013)
14. Trypuz, R., Kuzinski, D., Sopek, M.: General legal entity identifier ontology. In: Formal Ontology in Information Systems (FOIS) Ontology Competition (2016)
15. Welty, C., Fikes, R., Makarios, S.: A reusable ontology for fluents in owl. In: Formal Ontology in Information Systems (FOIS). vol. 150, pp. 226–236 (2006)
16. 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. pp. 283–292. IEEE (2010)