Reuse with Domain and Process Ontologies

Bahar AAMERI [a], Michael GRUNINGER [b]
[a] *Department of Computer Science, University of Toronto, Canada*
[b] *Department of Mechanical and Industrial Eng,University of Toronto, Canada*

Abstract. Although several generic process ontologies have been proposed in the past, the design of process ontologies for specific domains remains a challenge. Earlier work by Aameri that provided a methodology for specifying a domain process ontology requires a characterization of the partial automorphisms of the models of an underlying static domain ontology. In this paper, we exploit the modularization of domain ontologies as a means for reusing the existing domain process ontologies through ontology combination and merging. The metatheoretic relationships among ontologies within a repository play a key role in the characterization of how a domain process ontology for a given static domain is related to the domain process ontologies based on the different modules of the static domain ontology.

Keywords. process ontology, domain ontology, reuse, reducibility, ontology design

1. Introduction

The design of ontologies which axiomatize classes of processes specific to a particular domain remains an important challenge in ontological engineering. A domain process ontology axiomatizes possible classes of change within the underlying domain, without enumerating the definable classes of activities. This distinguishes a process ontology from process descriptions and action theories, which axiomatize properties of specific activities. The focus is on domain process ontologies that only classify the atomic activities within a domain, those that characterize complex activities in the domain are more closely related to AI planning problems.

Aameri [1] proposes a four step methodology for designing domain process ontologies. The verification of the process ontology requires the identification of the partial automorphisms of the models that represent the underlying domain, and consequently needs a characterization of all models of the underlying domain ontology. Unfortunately, both of these tasks can be quite difficult. In fact, for most real-world ontologies it is currently unknown which classes of mathematical structures are isomorphic to their models; the identification of the partial automorphisms of their models is thus neither trivial nor available in the literature.

Suppose we need an ontology which axiomatizes sheet metal manufacturing processes, and we utilize the CardWorld and BoxWorld ontologies, presented in [2], for representing 2D and 3D shapes. The characterization of the models of both ontologies is not a straightforward task, since neither ontology is synonymous with a single known mathematical theory. Even if we did characterize the models of these ontologies, we would most probably get lost trying to identify the partial automorphisms. To make this problem easier, it should be possible to decompose the ontologies into modules, identify the partial automorphisms of the models of these modules, and then use this information in building the process ontology.

As another example, suppose we want to axiomatize abstract data types and their operations. Intuitively, we should be able to reuse the chains process ontology [1] for representing lists, since the ontology of lists is synonymous with the ontology of linear orderings, and the linear ordering ontology is an non-conservative extension of the ontology of chains (which are sets of linear orderings).

The focus of this paper is the presentation of formal techniques which address the idea of reusability and modularization in design and verification of domain process ontologies; in particular, we discuss how relationships between domain ontologies within an ontology repository can assist us in identifying the properties of ontologies' models which are required in developing new domain process ontologies.

The concepts of reusability and modularization in the characterization of the models of ontologies have been explored in [3]. Metatheoretical relationships (such as relative interpretation, definability, conservative and non-conservative extension) between first-order theories are exploited to construct models of a given ontology based on models of the related ontologies. At the base level of such techniques are ontologies that axiomatize general mathematical structures like graphs and lattices. However, these techniques cannot always be used in designing domain process ontologies, as the relationships between domain ontologies are not always preserved between the corresponding process ontologies. Our proposed solution is to exploit the same logical relationships between the domain ontologies, but in a weaker form of reuse; instead of directly importing the modules of ontologies that are related to a given domain ontology, we only employ certain properties of those modules in designing the new domain process ontology.

We start by describing the general properties of domain process ontologies, and reviewing the methodology that Aameri proposed for designing such ontologies. In Section 3, we review the relationships between domain ontologies within a repository, and show how existing domain process ontologies can be reused in designing new domain process ontologies.

2. Properties of Domain Process Ontologies

As we explained in the introduction, the objective in designing domain process ontologies is to have a complete classification of activities based on their effects, without enumerating all definable activity classes. By following the design methodology proposed in [1], one can develop domain process ontologies that satisfy these requirements. We first briefly review the design methodology, and then prove this claim.

The key idea behind the methodology is that all changes within a domain can be described based on a fixed set of (primitive or defined) relations. If we identify those relations and axiomatize activity classes that affect them, we can be sure that we have a complete classification. For a given domain, the methodology designs a first-order process ontology with three main modules – one is the ontology of the Process Specification Language (PSL) [4], and the other two are constructed based on the properties of the domain.

The PSL ontology[1] is a generic process ontology that axiomatizes fundamental concepts for describing processes. Within the PSL ontology, states are represented by the

[1] http://colore.oor.net/process%5Fspecification%5Flanguage/psl%5Foutercore.clif

Predicate	***Interpretation***
$activity(\mathbf{a})$	$\mathbf{a}$ is an activity.
$occurrence\_of(\mathbf{o},\mathbf{a})$	$\mathbf{o}$ is an occurrence of activity $\mathbf{a}$.
$prior(\mathbf{f},\mathbf{o})$	Fluent $\mathbf{f}$ holds before activity occurrence $\mathbf{o}$.
$holds(\mathbf{f},\mathbf{o})$	Fluent $\mathbf{f}$ holds after activity occurrence $\mathbf{o}$.
$changes(\mathbf{o},\mathbf{f})$	$(\neg prior(\mathbf{f},\mathbf{o}) \wedge holds(\mathbf{f},\mathbf{o})) \vee (prior(\mathbf{f},\mathbf{o}) \wedge \neg holds(\mathbf{f},\mathbf{o}))$.

Table 1. PSL Predicates

set of fluents that hold before or after activity occurrences, and only activity occurrences can cause state transitions. A model of the PSL ontology includes partially ordered sets of activity occurrences, called occurrence trees. The root of an occurrence tree is associated with an initial state, and the branches are all sequences of occurrences of the atomic activities. Table 1 briefly describes PSL predicates that we use in this paper.

The methodology begins by identifying the appropriate domain ontology that axiomatizes the concepts of the underlying domain and their relationships independent of the notion of change. To enable reasoning about change, the axioms of the domain ontology are translated into a set of state constraints which extend the PSL ontology. Translation definitions map each relation symbol in the signature of the domain ontology into a fluent symbol in the signature of the state constraints. If the domain ontology is not sorted, the set of state constraints must contain an axiom which indicates sorts of the elements in the models of the domain ontology. The set of state constraints together with the PSL ontology is called *domain state ontology*.

The results in [1] show that in a model of a domain state ontology, each activity occurrence $\mathbf{o}$ is associated with two models of the corresponding domain ontology; one that represents the state before the activity occurrence (and is denoted by $\mu(\mathbf{o})$) and the other that represents the state after the activity occurrence (and is denoted by $\eta(\mathbf{o})$). In the other words, an activity occurrence is a transition between two models of the domain ontology. In that sense, classifying activities is equivalent to identifying different ways of changing models of the underlying domain ontology. Instead of considering fluents that are changed, the methodology characterizes an activity occurrence $\mathbf{o}$ by fluents that are preserved. This means that we need a representation for invariant (or equivalently common) substructures of $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$. Since we want to characterize changes to a particular set of fluents, we use a special type of structure-preserving maps:

Definition 1 *Let $\mathcal{M}_1, \mathcal{M}_2$ be structures with signature L.*
An injective mapping $\varphi : \mathcal{M}_1 \rightarrow \mathcal{M}_2$ is a partial isomorphism loosed to a non-empty set $L^- \subseteq L$ iff for all relations $\mathbf{R} \in L^-$, $\langle \mathbf{x_1}, \ldots, \mathbf{x_n} \rangle \in \mathbf{R}^{\mathcal{M}_1}$ iff $\langle \varphi(\mathbf{x_1}), \ldots, \varphi(\mathbf{x_n}) \rangle \in \mathbf{R}^{\mathcal{M}_2}$.

A mapping $\varphi : \mathcal{M} \rightarrow \mathcal{M}$ is a loosed partial automorphism iff it is a loosed isomorphism between substructures of $\mathcal{M}$.[2]

In general, a structure $\mathcal{N}$ can be represented by the set of all its partial automorphisms denoted by $PAut(\mathcal{N})$. Note that $PAut(\mathcal{N})$ forms a monoid [5]. Since $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$ are models of the same theory with the same universe[3] their invariant substructures

[2]We denote structures by calligraphic font: $\mathcal{M}, \mathcal{N}, \ldots$; the extension of a relation $\mathbf{R}$ in a structure $\mathcal{M}$ by $\langle \mathbf{a_1}, \ldots, \mathbf{a_n} \rangle \in R^{\mathcal{M}}$; the domain of a mapping φ by $dom(\varphi)$; and the signature of a first-order theory T by $\Sigma(T)$.

[3]We assume that for all $\mathbf{o}$, the universes of $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$ are equal; when an element is destroyed, it is not annihilated from the universe. Rather, its sort will change from something that is indicated by the state constraints to $object$. Conversely, when an element is created a new sort is assigned to an $object$ of the universe.

can be represented by subsets of $PAut(\mu(\mathbf{o}))$. Moreover, the results in [5] and [6] show that the lattice of the substructures of $\mathscr{N}$ is isomorphic to the lattice of the partial identities in $PAut(\mathscr{N})$. Consequently, if $\mathscr{S}$ is an invariant substructure of $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$ (i.e. $\mathscr{S} \subset \mu(\mathbf{o})$ and $\mathscr{S} \subset \eta(\mathbf{o})$), then the submonoid of $PAut(\mu(\mathbf{o}))$ that represents $\mathscr{S}$ can be extracted by the partial identity in $PAut(\mu(\mathbf{o}))$ which its domain is equal to the universe of $\mathscr{S}$. The scaffold $\mathfrak{S}_{\mathbf{o}}^{L}$ is then the set of sets of partial automorphisms, loosed to L, that correspond to all invariant substructures (with respect to L) of $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$.

Definition 2 *(from [1]) Let $\mathscr{M}$ be a model of a domain state ontology.*
A scaffold $\mathfrak{S}_{\mathbf{o}}^{L}$ of an activity occurrence $\mathbf{o}$ in $\mathscr{M}$ is a set consisting of all sets $\mathbb{G}_i$ such that

1. *All $G_{ij} \in \mathbb{G}_i$ are submonoids of $PAut^{L}(\mu(\mathbf{o}))$, such that their identity elements $\mathbf{e}_{ij}$ are partial identities associated with invariant substructures (with respect to L) of $\mu(\mathbf{o})$ and $\eta(\mathbf{o})$;*
2. *The identity element $\mathbf{e}_{ij}$ of each $G_{ij} \in \mathbb{G}_i$ has a maximal domain; that is, there is no other identity mapping $\mathbf{e}'$ that satisfies property 1 and $dom(\mathbf{e}_{ij}) \subset dom(\mathbf{e}')$;*
3. *If $\mathbb{G}_i = \{G_{i1}, \ldots, G_{im}\}$, then $dom(\mathbf{e}_{i1}) \cup \cdots \cup dom(\mathbf{e}_{im}) \subseteq dom(\mu(\mathbf{o}))$, and $dom(\mathbf{e}_{i1}) \cap \cdots \cap dom(\mathbf{e}_{im}) = \varnothing$.*

The trivial scaffold $\mathscr{I}_{\mathbf{o}}$ is the set of set of all partial automorphisms of $\mu(\mathbf{o})$.

Activities can be categorized based on which fluents they change. However, since the objective is not to enumerate all activity classes, the methodology distinguishes the invariant substructures up to isomorphism, and more importantly, does not consider the number of fluents that activity occurrences change. The key question is how to identify a set of fluents that can describe all possible changes within the domain ontology?
Suppose for two activity occurrences $\mathbf{o_1}, \mathbf{o_2}$, we have $\mu(\mathbf{o_1}) \cong \mu(\mathbf{o_2})$, but $\eta(\mathbf{o_1}) \ncong \eta(\mathbf{o_2})$. Then, there is at least one fluent, denoted by the fluent symbol F, that $\mathbf{o_1}$ changes it, and $\mathbf{o_2}$ preserves it, and so $\mathfrak{S}_{\mathbf{o_1}}^{F}$ is not isomorphic to $\mathfrak{S}_{\mathbf{o_2}}^{F}$. Therefore, the answer to the above question is equivalent to find the set $\mathscr{C}_T$ defined in the following:

Definition 3 *Suppose a theory $T_{psl} \cup T_{st}$ is the domain state ontology for a domain ontology T, and let L be the set of all primitive and definable symbols in the language of $T_{psl} \cup T_{st}$ that correspond to the symbols in the language of T.*
$\mathscr{C}_T = \{L_1, ..., L_n\}$, is the minimal set such that

1. *$L_1 \subseteq L, ..., L_n \subseteq L$;*
2. *for all activity occurrences $\mathbf{o_1}, \mathbf{o_2}$ in models of $T_{psl} \cup T_{st}$*
   $\mu(\mathbf{o_1}) \cong \mu(\mathbf{o_2}), \mathfrak{S}_{\mathbf{o_1}}^{L_1} \cong \mathfrak{S}_{\mathbf{o_2}}^{L_1}, ..., \mathfrak{S}_{\mathbf{o_1}}^{L_n} \cong \mathfrak{S}_{\mathbf{o_2}}^{L_n}$ implies $\eta(\mathbf{o_1}) \cong \eta(\mathbf{o_2})$.

For the chains ontology T_{chains}, we have $\mathscr{C}_{T_{chains}} = \{\{point\}, \{lt\}, \{comparable\}\}$ since a theorem in [1] shows that for all activity occurrences $\mathbf{o_1}, \mathbf{o_2}$ in models of the chains state ontology, if $\mu(\mathbf{o_1}) \cong \mu(\mathbf{o_2})$ and the scaffolds of $\mathbf{o_1}$ and $\mathbf{o_2}$ that correspond to fluent symbols *point*, *lt* (stands for *"less than"*), and *comparable* are isomorphic, then $\eta(\mathbf{o_1}) \cong \eta(\mathbf{o_2})$.

Definition 4 summarizes the properties of domain process ontologies that are designed with respect to the methodology. Considering property 2 in Definition 3, a complete classification includes an activity class for each element L_i of $\mathscr{C}_T$ such that the members of the class change fluents in L_i. Thus, for each activity class the domain process ontology includes a definition that specifies which fluents are changed and which fluents are preserved by the members of the class. Since these definitions are in the signature

$$(\forall a)\, change\_point(a) \equiv ((\forall o)\, occurrence\_of(o,a) \supset (\exists x)\, changes(o, point(x))). \quad (1)$$

$$(\forall a)\, change\_lt(a) \equiv ((\forall o)\, occurrence\_of(o,a) \supset (\exists x,y)\, changes(o, lt(x,y))). \quad (2)$$

$$(\forall a)\, new\_chain(a) \equiv (change\_lt(a) \wedge$$

$$(\forall o,x,y)\, (occurrence\_of(o,a) \wedge changes(o, lt(x,y))) \supset changes(o, comparable(x,y))). \quad (3)$$

$$(\forall a)\, rearrange(a) \equiv (change\_lt(a) \wedge$$

$$(\forall o,x,y)\, (occurrence\_of(o,a) \wedge changes(o, lt(x,y))) \supset \neg changes(o, comparable(x,y))). \quad (4)$$

Figure 1. Axioms for $T_{chainsprocess}$[4](from [1]).

of the underlying domain state ontology, the domain process ontology is a definitional extension of the domain state ontology. This is captured by property 1 in Definition 4.

Note that $\mathfrak{S}_{\mathbf{o}}^{L_i}$ represents all changes that **o** makes to the fluents specified by symbols in L_i. Therefore, if an activity class χ_i is associated with L_i, the scaffolds that correspond to L_i and the occurrences of the members of χ_i are nontrivial. On the other hand, when an activity is not a member of χ_i, its occurrences do not change fluents in L_i, and so the corresponding scaffold is trivial. This is captured by property 2a in Definition 4.

Notice however that there might be constraints in the domain that enforce changing fluents in a set $L_j \in \mathscr{C}_T$ whenever the fluents in $L_i \in \mathscr{C}_T$ are changed. In that case, χ_i is a subclass of χ_j. Moreover, there are activities that change fluents in L_j without changing the fluents in L_i. An additional class is therefore required to distinguish these types of activities from activities in χ_i. For example, the fluent symbol *comparable* in T_{chains} is defined based on lt, and so any activity that changes *comparable* will also change lt. The activity class $\chi_{comparable}$ is therefore a subclass of χ_{lt}, and the chains process ontology (see Figure 1) includes a class of activities that change a fluent $\mathbf{lt}(\mathbf{a},\mathbf{b})$, but preserve $\mathbf{comparable}(\mathbf{a},\mathbf{b})$. Property 2b addresses this requirement.

Definition 4 *Let $T_{psl} \cup T_{st}$ be a domain state ontology for a domain ontology T.*
Let $\lambda(T_{process}) = \Sigma(T_{process}) \setminus \Sigma(T_{psl} \cup T_{st})$.
$T_{psl} \cup T_{st} \cup T_{process}$ is a domain process ontology for T iff

1. *$T_{psl} \cup T_{st} \cup T_{process}$ is a definitional extension of $T_{psl} \cup T_{st}$ such that for each $\chi_i \in \lambda(T_{process})$ there is an axiom in $T_{process}$ of the following form,*

$$(\forall a)\chi_i(a) \equiv ((\forall o)\, occurrence\_of(o,a) \supset \Psi_i(o),$$

where $\Psi_i(o)$ is a formula in the language of $T_{psl} \cup T_{st}$ with a unique free variable o

2. *Let $Act(\chi_i) = \{\mathbf{a} : \langle \mathbf{a} \rangle \in \chi_i^{\mathscr{M}}\}$, where $\mathscr{M} \in Mod(T_{psl} \cup T_{st} \cup T_{process})$, and $SubClass(\chi_i) = \{Act(\chi_j) : Act(\chi_j) \subset Act(\chi_i)\}$.*
There is an injective mapping $\varphi : \mathscr{C}_T \to \lambda(T_{process})$ such that
   (a) *if $\chi_i = \varphi(L_i)$, then $\langle \mathbf{a} \rangle \in \chi_i^{\mathscr{M}}$ iff for all occurrences **o** of **a**, $\mathfrak{S}_{\mathbf{o}}^{L_i} \neq \mathscr{I}_{\mathbf{o}}$.*
   (b) *if there is no $L_i \in \mathscr{C}_T$ such that $\chi_i = \varphi(L_i)$, then exists $\chi_j = \varphi(L_j)$, such that $SubClass(\chi_j) \neq \varnothing$, and $Act(\chi_j) \setminus \bigcup SubClass(\chi_j) \subset Act(\chi_i)$.*

We call $T_{process}$ the domain process schemata *for T.*

[4]http://colore.oor.net/chains%5Fprocess/definitions/chains%5Fprocess.clif

From this point forward, we will be using the phrase "domain process ontology" exclusively when referring to process ontologies that satisfy the properties specified by Definition 4.

Let **o** be an occurrence of an activity **a** which satisfies property 2 in Definition 3, but **a** is not a member of any activity class defined by the domain process ontology. According to the above definition, all scaffolds of **o** are trivial, and since **o** satisfies property 2, **a** has no effects in the underlying domain. In the other words, all nontrivial activities within a domain are members of at least one activity class defined by the respective domain process ontology. This implies that the classifications specified by domain process ontologies are complete.

A domain process ontology classifies activities and defines the corresponding hierarchy in such a way that classes in the leaves of the hierarchy tree includes activities that have same effects on isomorphic substructures of the respective domain. This ensures that the classification is neither enumerative, nor too general.
Theorem 1 proves these two important properties of domain process ontologies.

Theorem 1 [5] *Let $T_{psl} \cup T_{st} \cup T_{process}$ be the domain process ontology for T, and $\chi_1, ..., \chi_n$ be the activity classes defined in $T_{process}$.*

1. $T_{psl} \cup T_{st} \cup T_{process} \models (\forall a) activity(a) \supset \chi_1(a) \vee ... \vee \chi_n(a)$,
2. *Let $\mathbf{a_1}, \mathbf{a_2}$ be activities in a model $\mathscr{M}$ of $T_{psl} \cup T_{st} \cup T_{process}$ such that the effects of $\mathbf{a_1}$ and $\mathbf{a_2}$ on the isomorphic substructures of a model of T are not isomorphic. Then, either exists a leaf activity class χ_i in $T_{process}$ such that $\langle \mathbf{a_1} \rangle \in \chi_i^{\mathscr{M}}$ and $\langle \mathbf{a_2} \rangle \notin \chi_i^{\mathscr{M}}$, or $\mathbf{a_1}$ and $\mathbf{a_2}$ are members of more than one leaf activity class.*

The chains process ontology, for example, specifies a complete classification of possible activities, but does not enumerate all definable sets of activities. In the meantime, it is not too general as the members of each leaf activity class have same effects. To make this more clear, consider this example: let $\mathscr{M} \in Mod(T_{chains})$, where $M = \{\mathbf{x}, \mathbf{y}\}$, and $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbf{lt}^{\mathscr{M}}$. Suppose $\mathbf{a_1}$ changes $\mathscr{M}$ to $\mathscr{M}_1$, where $\langle \mathbf{y}, \mathbf{x} \rangle \in \mathbf{lt}^{\mathscr{M}_1}$, and $\mathbf{a_2}$ changes $\mathscr{M}$ to $\mathscr{M}_2$, where $M_2 = \{\mathbf{x}, \mathbf{y}\}$, but $\langle \mathbf{x}, \mathbf{y} \rangle \notin \mathbf{lt}^{\mathscr{M}_2}$ and $\langle \mathbf{y}, \mathbf{x} \rangle \notin \mathbf{lt}^{\mathscr{M}_2}$. Clearly, both $\mathbf{a_1}$ and $\mathbf{a_2}$ are members of *change_lt*, however, this class is too general since $\mathbf{a_1}$ and $\mathbf{a_2}$ change isomorphic substructures of $\mathscr{M}$, but have different effects. The chains process ontology resolves this problem by defining two subclasses, *new_chain* and *rearrange*, for *change_lt*.

3. Designing Process Ontologies Using Repositories

In designing domain process ontologies we face two main challenges. First, we need to find the smallest set of fluent symbols $\mathscr{C}$, that satisfies property 2 in Definition 3. Second, we have to identify the set of partial automorphisms for the relations that correspond to those fluents. For domains with more complex structures this is even harder, since the number of combinations that have to be examined with respect to property 2 increases exponentially. Fortunately, in practice we have found that elements (L_i) of the fluents set $\mathscr{C}$ usually includes at most three fluent symbols. Therefore, in most cases we can find mathematical structures, with well-studied sets of partial automorphisms, that represent axioms that correspond to elements in L_i. This would address the second challenge.

For the first challenge, one solution is to decompose the domain ontology into modules, determine the fluent set for each module, and then find the fluent set for the original

[5]Proofs of the theorems can be found at http://stl.mie.utoronto.ca/publications/reuseprocess.pdf

ontology with respect to the fluent sets of its modules. In the following, we explore this idea further. We investigate three types of relationships among domain ontologies in a repository, namely extensions, synonymy and reducibility, and show how these relationships can help us in developing domain process ontologies.

3.1. Extensions in Hierarchies

We start by the simplest relationship between first-order ontologies.

Definition 5 *Let T_1, T_2 be two first-order ontologies such that $\Sigma(T_1) \subseteq \Sigma(T_2)$. T_2 is an extension of T_1 iff for any sentence Φ in the language of T_1, $T_1 \models \Phi$ implies $T_2 \models \Phi$.*
T_2 is a conservative extension of T_1 iff $T_1 \not\models \Phi$ implies $T_2 \not\models \Phi$.

Unfortunately, extension (or reduction) between domain ontologies is not preserved between the corresponding domain process ontologies. The reason is that in classifying activities, we consider partial symmetries between the models of the domain ontology (partial symmetries are captured by partial automorphisms), while an arbitrary extension might both increase or decrease partial symmetries among the models. However, we can take advantage of extensions within the hierarchies of an ontology repository.

Definition 6 *(from [7]) A hierarchy $\mathbb{H} = \langle \mathscr{H}, \leq \rangle$ is a partially ordered finite set of ontologies $\mathscr{H} = T_1, ..., T_n$ such that $\Sigma(T_i) = \Sigma(T_j)$, for all i, j, and $T_1 < T_2$ iff T_2 is a non-conservative extension of T_1.*

We know that $T' \leq T$ implies $Mod(T) \subseteq Mod(T')$. That means that there might be axioms in the extension which excludes transitions to models that are related to a specific type of change, and so eliminates some of the activity classes of T'. The extension might also change the definitions of the activity classes.

Theorem 2 *Let $T' \leq T$. Then $\mathscr{C}_T \subseteq \mathscr{C}_{T'}$.*

Consider the ontology T_{chains} , and the ontology $T_{linorder}$ of linear orderings. $T_{linorder}$ is an extension of T_{chains}, and they are in the same hierarchy. As we described earlier, the chains process ontology includes three leaf activity classes, where the first class includes activities that create or destroy elements in the orderings, the second one includes activities which reorder elements in a single chain, and the third one is the set of activities that break a chain into a group of chains, or join a group of chains to make a single chain. However, the domain process ontology for $T_{linorder}$ only includes the first two classes since there is an axiom in $T_{linorder}$ which states that models of $T_{linorder}$ are single chains.

3.2. Definitional Extensions and Synonymous Ontologies

Recall that every constant, function, and relation in a model of a definitional extension of an ontology is definable in a model of the ontology. Consequently, every model of the ontology can be expanded to a unique model of its definitional extension, and two models of the ontology are isomorphic iff their expansions are isomorphic. This leads us to conclude that a domain ontology and all its definitional extensions share one single domain process ontology.

Theorem 3 *Let P_1 be the domain process schemata corresponds to T_1, and T_2 be a definitional extension of T_1. Then P_1 is the domain process schemata for T_2.*

Definition 7 *Two ontologies T_1 and T_2 are synonymous iff there exists an ontology T_3 with the signature $\Sigma(T_1) \cup \Sigma(T_2)$ that is a definitional extension of T_1 and T_2.*

Translation definitions are sentences that interpret non-logical symbols of an ontology, in the language of another ontology. It can be easily proved that for two synonymous ontologies T_1 and T_2, there exists a set of translation definitions Δ_{21} from T_2 into T_1 and a set of translation definitions Δ_{12} from T_1 into T_2 such that $T_1 \cup \Delta_{21}$ is logically equivalent to $T_2 \cup \Delta_{12}$. The other direction is true as well.

Definition 8 *Let T_0 and T_1 be two ontologies such that $\Sigma(T_0) \cap \Sigma(T_1) = \varnothing$. Translation definitions for T_0 into T_1 are sentences in $\Sigma(T_0) \cup \Sigma(T_1)$ of the form*

$$\forall \bar{x}\, p_i(\bar{x}) \equiv \Phi(\bar{x}),$$

where $p_i(\bar{x})$ is a symbol in $\Sigma(T_0)$ and $\Phi(\bar{x})$ is a formula in the language of T_1.

By definition, every pair of synonymous ontologies have a common definitional extension. Thus, the following theorem is an immediate consequence of Theorem 3.

Theorem 4 *Let P_1 be the domain process schemata corresponds to T_1, and T_2 be synonymous with T_1. P_1 is synonymous with the domain process schemata for T_2.*

Suppose two ontologies T_1, T_2 are synonymous, and the process ontology P_1 for T_1 already exists in the repository. We can translate the axioms of P_1 into the language of T_2, using Δ_{12}, and the result is the process ontology for T_2.

Consider the DOLCE direct quality ontology T_{dir_qual}[6] (discussed in [8])

$$(\forall x, y_1, y_2)\, dqt(x, y_1) \wedge dqt(x, y_2) \supset y_1 = y_2 \tag{5}$$

$$(\forall x_1, x_2, y)\, TQ(x_1) \wedge TQ(x_2) \wedge PD(y) \wedge dqt(x_1, y) \wedge dqt(x_2, y) \supset x_1 = x_2 \tag{6}$$

$$(\forall x)\, TQ(x) \supset (\exists y)\, PD(y) \wedge dqt(x, y) \tag{7}$$

Using an automated theorem prover, it can be shown that T_{dir_qual} is synonymous with the ontology of injection bipartite incidence structures $T_{inj\_bipartite}$[7] under the following translation definitions

$$\Pi: \quad (\forall x, y)\, dqt(x, y) \equiv in(x, y) \wedge point(x) \wedge line(y) \tag{8}$$

$$(\forall x)\, TQ(x) \equiv point(x) \tag{9}$$

$$(\forall x)\, PD(x) \equiv line(x) \tag{10}$$

$$\Delta: \quad (\forall x, y)\, in(x, y) \equiv dqt(x, y) \vee dqt(y, x) \vee x = y \tag{11}$$

$$(\forall x)\, point(x) \equiv TQ(x) \tag{12}$$

$$(\forall x)\, line(x) \equiv PD(x). \tag{13}$$

Therefore, the process ontology for T_{dir_qual} can be achieved by translating $T_{injbipart\_proc}$[8], which is the process ontology for $T_{inj\_bipartite}$, into the language of T_{dir_qual} using Δ. The result is depicted in Figure 2.[9]

[6] http://colore.oor.net/quality/direct%5Fquality.clif

[7] http://colore.oor.net/bipartite%5Fincidence/injection%5Fbipartite.clif

[8] http://colore.oor.net/injection%5Fbipartite%5Fprocess/definitions/injection%5Fbipartite%5Fprocess.clif

[9] http://colore.oor.net/direct%5Fquality%5Fprocess/definitions/direct%5Fquality%5Fprocess.clif

$$(\forall a)\, change\_PD(a) \equiv ((\forall o)\, occurrence\_of(o,a) \supset (\exists x)\, changes(o, PD(x))). \tag{14}$$

$$(\forall a)\, change\_quality(a) \equiv ((\forall o)\, occurrence\_of(o,a) \supset (\exists x,y)\, changes(o, dqt(x,y))). \tag{15}$$

Figure 2. Axioms of $T_{dqtyprocess}$.

3.3. Reducible Ontologies

So far we have considered relationships between two ontologies. In the remaining of this section, we investigate reusing domain process ontologies of theories that are synonymous with the constructing modules of a domain ontology. We exploit the the notion of faithful interpretation [3], which is the generalization of the notion of conservative extension to ontologies with distinct signatures.

Definition 9 *The mapping π is an interpretation of an ontology T_1 into an ontology T_2 iff for all sentence Φ in the language of T_1, $T_1 \models \Phi$ implies $T_2 \models \pi(\Phi)$.*
π is a faithful interpretation of T_1 into T_2 iff $T_1 \not\models \Phi$ implies $T_2 \not\models \pi(\Phi)$.

Synonymy is a relation between two ontologies. The notion of reducibility (from [3]) extends it to a relationship among a set of ontologies.

Definition 10 *An ontology T is reducible to a set of ontologies $T_1, ..., T_n$ iff*

1. *T faithfully interprets each ontology T_i;*
2. *$T_1 \cup ... \cup T_n$ faithfully interprets T;*
3. *T is synonymous with $T_1 \cup ... \cup T_n$.*

The following lemma shows the construction of a domain process ontology from the domain process ontologies of the modules of the underlying domain ontology.

Lemma 1 *Suppose $T = T_1 \cup ... \cup T_n$ is a consistent theory, and $P, P_1, ..., P_n$ are domain process schemata correspond to $T, T_1, ..., T_n$ respectively.*
If $\Sigma(T_i) \cap \Sigma(T_j) = \varnothing$, for all $1 \le i, j \le n$, then $P = P_1 \cup ... \cup P_n$.

Suppose in a reduction $T_1, ..., T_n$ for T, all modules except T_1 and T_2 have disjoint signatures. We can construct a new reduction $T_1', T_3, ..., T_n$ for T, where $T_1' = T_1 \cup T_2$, so that the signatures of theories in the new reduction is disjoint. The following theorem, which is a consequence of Definition 10 and Theorem 4, shows the construction of the domain process ontology for T using the domain process ontologies of its disjoint reductive modules. Note that the domain process ontologies for T_1 and T_2 can be reused in constructing the domain process ontology for $T_1 \cup T_2$.

Theorem 5 *Suppose T is reducible to a set of ontologies $T_1, ..., T_n$ such that $\Sigma(T_i) \cap \Sigma(T_j) = \varnothing$, for $1 \le i, j \le n$. Suppose $P, P_1, ..., P_n$ are domain process schemata correspond to $T, T_1, ..., T_n$ respectively.*
Then P is synonymous with $P_1 \cup ... \cup P_n$.

Consider a version of the Blocks World problem, in which at most one block can be placed on another, and each block has one unique color [9]. It is easy to verify that the colored blocks world ontology is reducible to the bounded chains ontology $T_{bounded\_chains}$[10] and T_{dir_qual} under the following translation definitions

[10]http://colore.oor.net/orderings/bounded%5Fchains.clif

$$(\forall x,y,z)\, C(x,y,z) \supset ((\exists l) line(l) \wedge in(x,l) \wedge in(y,l) \wedge in(z,l)). \tag{20}$$

$$(\forall x,y,z,l)\, line(l) \wedge in(x,l) \wedge in(y,l) \wedge in(z,l) \wedge (x \neq y) \wedge (x \neq z) \wedge (y \neq z) \supset (C(x,y,z) \vee C(z,y,x)). \tag{21}$$

Figure 3. Two axioms in the residue of T_{mcg}.

$$(\forall x,y)\, on(y,x) \equiv lt(x,y) \tag{16}$$

$$(\forall x)\, block(x) \equiv TQ(x) \equiv point(x) \tag{17}$$

$$(\forall x)\, color(x) \equiv PD(x) \tag{18}$$

$$(\forall x,y)\, colored\_as(x,y) \equiv dqt(x,y) \tag{19}$$

For the bounded chains ontology, we reuse the verified chains process ontology from [1] (Figure 1). Considering Theorem 5, the colored blocks process ontology[11] can be obtained by translating the axioms in Figure 1 and Figure 2 using axioms 16-19.

3.4. Weakly Reducible Ontologies

Many ontologies are not reducible to standard mathematical ontologies, however, they have subtheories that are. These cases can be captured by the notion of weak reducibility.

Definition 11 *An ontology T is weakly reducible to ontologies $T_1, ..., T_n$ iff*

1. *T faithfully interprets each ontology T_i;*
2. *$T_1 \cup ... \cup T_n$ faithfully interprets a subtheory $T' \leq T$;*
3. *T' is synonymous with $T_1 \cup ... \cup T_n$.*

The domain process ontologies of the reductive modules of a weakly reducible ontology T can be reused in designing the domain process ontology for the subtheory T' of T which is synonymous with the weak reduction. The new domain process ontology would be helpful in constructing the domain process ontology for T, as T and T' are in the same hierarchy and T is an extension of T'.

Theorem 6 *Let T be weakly reducible to $T_1, ..., T_n$.
Then $\mathscr{C}_T \subseteq \mathscr{C}_{T'}$, where $T' \leq T$ and T' is synonymous with $T_1 \cup ... \cup T_n$.*

Consider the ontology T_{mcg}[12] of Megiddo cyclic geometries. T_{mcg} is weakly reducible to the ontology T_{cyclic}[13] of cyclic orderings and the ontology T_{weak_bipart}[14] of weak bipartite incidence geometries. Figure 3 shows the axioms of T_{mcg} that are not in T_{cyclic} and T_{w_bipart}. The domain process ontologies for T_{cyclic}[15] and T_{w_bipart}[16] already exist in the repository, and so we know that $\mathscr{C}_{T_{cyclic}} = \{\{point\}, \{C\}, \{cmp\}\}$ (where $cmp(x,y,z)$ holds iff $C(x,y,z)$ or $C(z,y,x)$ hold) and $\mathscr{C}_{T_{w\_bipart}} = \{\{point\}, \{line\}, \{in\}\}$. Consequently, we have $\mathscr{C}_{T_{mcg}} \subseteq \{\{point\}, \{line\}, \{in\}, \{C\}, \{cmp\}\}$. However, Axioms

[11] http://colore.oor.net/colored%5Fblocks%5Fprocess/definitions/colored%5Fblocks%5Fprocess.clif
[12] http://colore.oor.net/cyclic%5Fgeometry/mcg.clif
[13] http://colore.oor.net/cyclic%5Fordering/cyclic.clif
[14] http://colore.oor.net/bipartite%5Fincidence/weak%5Fbipartite.clif
[15] http://colore.oor.net/cyclic%5Fprocess/definitions/cyclic%5Fprocess.clif
[16] http://colore.oor.net/weak%5Fbipartite%5Fprocess/definitions/weak%5Fbipartite%5Fprocess.clif

20 and 21 prevent any changes in the fluent *cmp* without changing the fluent *in*, and therefore $\mathscr{C}_{T_{mcg}} = \{\{point\},\{line\},\{in\},\{C\}\}$. Theorem 7 formally proves this claim.[17]

Theorem 7 (MCG Classification Theorem) *Suppose $T_{psl} \cup T_{stmcg}$ is the domain state ontology for T_{mcg}, and $\mathscr{M} \in Mod(T_{psl} \cup T_{stmcg})$.*
For any activity occurrences $\mathbf{o_1}, \mathbf{o_2}$ in $\mathscr{M}$ with $\mu(\mathbf{o_1}) \cong \mu(\mathbf{o_2})$, if $\mathfrak{G}^{point}_{\mathbf{o_1}} \cong \mathfrak{G}^{point}_{\mathbf{o_2}}$, $\mathfrak{G}^{line}_{\mathbf{o_1}} \cong \mathfrak{G}^{line}_{\mathbf{o_2}}$, $\mathfrak{G}^{in}_{\mathbf{o_1}} \cong \mathfrak{G}^{in}_{\mathbf{o_2}}$, $\mathfrak{G}^{C}_{\mathbf{o_1}} \cong \mathfrak{G}^{C}_{\mathbf{o_2}}$, then $\mathfrak{G}^{cmp}_{\mathbf{o_1}} \cong \mathfrak{G}^{cmp}_{\mathbf{o_2}}$, and consequently $\eta(\mathbf{o_1}) \cong \eta(\mathbf{o_2})$.

3.5. Decomposable Ontologies

Decomposability is the weakest relation among the ontologies in a repository, but it can still assist us in developing domain process ontologies.

Definition 12 *An ontology T is decomposable into ontologies $T_1,...,T_n$ iff*

1. *T interprets each ontology T_i;*
2. *$T_1 \cup ... \cup T_n$ faithfully interprets a subtheory $T' \leq T$;*
3. *T' is synonymous with $T_1 \cup ... \cup T_n$.*

We can reuse the domain process ontologies of the reductive modules of a decomposable ontology in the same way as they are used for weakly reducible ontologies.

Theorem 8 *Let T be decomposable to $T_1,...,T_n$.*
Then $\mathscr{C}_T \subseteq \mathscr{C}_{T'}$, where $T' \leq T$ and T' is synonymous with $T_1 \cup ... \cup T_n$.

Consider the theory T_{rcc}[18] which axiomatizes the mereotopology Region Connection Calculus (RCC) [10]. The results in [3] shows that T_{rcc} is decomposable to the theory T_{BA}[19], which is a first-order axiomatization of the Boolean algebras, and a connection theory $T_{connection}$.[20]

We know that $\mathscr{C}_{T_{BA}} = \{\{element\}\}$ and $\mathscr{C}_{T_{connection}} = \{\{C\}\}$. Considering the translation definitions between T_{rcc} and T_{BA}, we can rename *element* to *region*, and therefore $\mathscr{C}_{T_{rcc}} \subseteq \{\{region\},\{C\}\}$. By the following theorem we have $\mathscr{C}_{T_{rcc}} = \{\{region\},\{C\}\}$. [21]

Theorem 9 (RCC Classification Theorem) *Suppose $T_{psl} \cup T_{strcc}$ is the domain state ontology for T_{rcc}, and $\mathscr{M} \in Mod(T_{psl} \cup T_{strcc})$.*

For any activity occurrences $\mathbf{o_1}, \mathbf{o_2}$ in $\mathscr{M}$ with $\mu(\mathbf{o_1}) \cong \mu(\mathbf{o_2})$, if $\mathfrak{G}^{region}_{\mathbf{o_1}} \cong \mathfrak{G}^{region}_{\mathbf{o_2}}$, $\mathfrak{G}^{C}_{\mathbf{o_1}} \cong \mathfrak{G}^{C}_{\mathbf{o_2}}$, then $\eta(\mathbf{o_1}) \cong \eta(\mathbf{o_2})$.

4. Conclusion and Future Work

By designing a domain process ontology based on its underlying static domain ontology, we lay the foundations for the verification of the domain process ontology with respect to its intended models. Nevertheless, there is more to the evaluation of a domain process ontology than a characterization of its models. One of the challenges in designing process ontologies is to avoid the pitfall of enumerating all definable classes of activities while still showing that the classes in the ontology capture all of the necessary distinctions. To this end, we have provided a formal definition of a domain process ontology and have

[17]MCG process ontology: http://colore.oor.net/mcg%5Fprocess/definitions/mcg%5Fprocess.clif
[18]http://colore.oor.net/mereotopology/rcc.clif
[19]http://colore.oor.net/lattices/boolean%5Flattice.clif
[20]http://colore.oor.net/connections/connection.clif
[21]RCC process ontology: http://colore.oor.net/rcc%5Fcontinuous%5Fprocess/definitions/rcc%5Fprocess.clif

shown that this enables us to prove the completeness of the classification specified by the ontology with respect to all possible changes within the domain.

A second challenge is that the methodology for designing process ontologies also requires a characterization of the partial automorphisms of the models that represent the underlying domain, and such a characterization is not known in many cases. To overcome this difficulty, we have proposed techniques for using the decomposition of the ontologies into modules, identifying the partial automorphisms of the models of these modules, and then using this information to build the process ontology. Furthermore, we applied these techniques to design three new domain process ontologies to demonstrate the feasibility of the approach.

In developing formal ontologies, reusability is exploited in different forms; existing ontologies can be imported, extended, specialized, combined [11], or translated [7] to build new ontologies. Likewise, reusability has been explored for the purpose of ontology verification [3], and in the context of ontology design patterns [12]. The type of reuse that we apply for designing domain process ontologies (except for the case of synonymy and reducibility) is not similar to the forms that are mentioned above. It is weaker, in the sense that we do not directly use the existing domain or process ontologies, but is still effective since it simplifies the design challenges and saves us from redoing a complex and error-prone task.

All of these process ontologies have been designed by starting with some existing domain ontology (such as orderings, geometries, or mereotopology). In practice, there also exist taxonomies of process classes that are part of upper ontologies or ontologies for areas such as chemical pathways and manufacturing. In these cases, there is no explicit static domain ontology which was used in the design of the taxonomies. Future work will need for evaluating such process taxonomies through the identification of the correct static domain ontology.

References

[1] B. Aameri. Using Partial Automorphisms to Design Process Ontologies. In *Proceedings of the 7th International Conference on Formal Ontology in Information Systems*, pages 309–322. IOS Press, 2012.

[2] M. Gruninger and S. Bouafoud. Thinking Outside (and Inside) the Box. In *SHAPES 1.0*, 2011.

[3] M. Gruninger, T. Hahmann, A. Hashemi, and D. Ong. Ontology Verification with Repositories. In *Proc. of the 6th FOIS Int. Conference*, pages 317–330, Toronto,Canada, 2010. IOS Press.

[4] M. Gruninger. Ontology of the Process Specification Language. In *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003.

[5] M.V. Lawson. *Inverse Semigroups: The Theory of Partial Symmetries*. World Scientific, 1998.

[6] D.A. Bredikhin. Inverse semigroups of local automorphisms of universal algebras. *Siberian Mathematical Journal*, 17(3):386–393, 1976.

[7] M. Grüninger, T. Hahmann, A. Hashemi, D. Ong, and A. Ozgovde. Modular First-order ontologies via repositories. *Applied Ontology*, 7(2):169–209, 2012.

[8] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. WonderWeb Deliverable D18–Ontology Library (final). *Technical Report, Laboratory for Applied Ontology, ISTC-CNR, Trento, Italy*.

[9] F. Lin and R. Reiter. State Constraints Revisited. *Journal of logic and computation*, 4(5):655–677, 1994.

[10] D. A. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *Proceedings of the Third Int. Conference on Principles of KR*, pages 165–176. San Mateo, California, 1992.

[11] H.S. Pinto and J.P. Martins. Reusing Ontologies. In *AAAI 2000 Spring Symposium on Bringing Knowledge to Business Processes*, pages 77–84. AAAI Press, 2000.

[12] A. Gangemi and V. Presutti. Ontology Design Patterns. In *Handbook on Ontologies*. Springer, 2009.