Ontology Representation : design patterns and ontologies that make sense

Hoekstra, R.J.

Publication date
2009

Chapter 7

Design Patterns

> "Art is the imposing of a pattern on experience, and our aesthetic enjoyment is recognition of the pattern."
>
> *Alfred North Whitehead, Dialogues (1954)*

7.1 Introduction

This chapter is an investigation in the applicability and reuse of structured ontology design patterns as defined in Section 5.6.4. Experience in the development of the LKIF Core ontology has shown that constructing a large ontology in a middle out fashion facilitates the emergence of such patterns. This is not just due to the fact that practice and the subsequent and separate representation of clusters of basic concepts makes one aware of recurrent structures. But the strong methodological emphasis on *reuse* of existing definitions targets awareness of the *similarities*, rather than *dissimilarities* between clusters. Finally, the commonsense and knowledge representation perspectives of the preceding chapter allow the ontology engineer some distance from the theoretical intricacies of the domain being represented.

The discussion of patterns that emerged in the construction of LKIF Core has several purposes. It provides a description and discussion of common design patterns for the purposes of reuse. At the same time, it is a more in-depth introduction to several important notions in LKIF Core, such as subjective, social concepts on the one hand and physical concepts such as processes on the other. Secondly, the reusability of patterns exemplified by these notions are evaluated by applying them to diverse use cases.

A third consideration is that a thorough description of the application of these design patterns gives insight in the way OWL 2 constructs are applied in the construction of complex class descriptions. As we have seen in Chapter 5

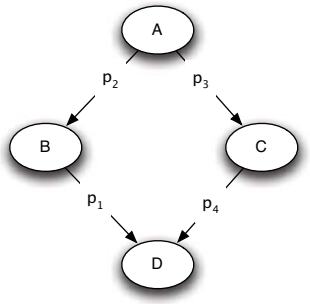

Figure 7.1: Diamond-shaped class description

the methodological and theoretical guidelines that apply to ontology development rarely if ever touch on the *use* of a knowledge representation language. Examples and guidelines for the application of OWL are reserved to either tutorials for novice users (Horridge et al., 2007), or an enumeration of common misunderstandings targeted to improve such educational resources (Rector et al., 2004). For sure, these efforts improve both the understanding of the language, and the ability to assess whether what is represented is an adequate representation of the domain, but they leave open the intermediate step: the representation *process* itself.

There are some examples of more practical 'cookbook' approaches, such as Seidenberg and Rector (2006)'s discussion of transitive properties in OWL, and work on practical structured design patterns such as in Noy and Rector (2006); Bobillo et al. (2007). Similarly, Gangemi (2005); Presutti et al. (2008) focus mainly on *content* patterns discussed in Section 5.6.4. However, these are often problem and domain specific, and present the design pattern as a given, readily reusable ontology modules. Furthermore, considering that design is a task, a *process*, it seems odd to describe design patterns only in terms of declarative structures, rather than by giving the procedure for applying them. Especially since the most valuable contribution of design patterns lies with the possibility to reuse and recombine them to form representations of new, uncharted domains.

7.1.1 Dealing with Models

The development of a large ontology such as LKIF Core is not only complex at a conceptual level, but involves numerous encounters with the limitations of the knowledge representation language as well. In some cases the combination of highly expressive language features is such that reasoner performance is severely reduced (cf. the GCI problem in Section 6.4), and sometimes the expressiveness required surpasses that of OWL 2 DL. The latter problem is fundamental, because the language puts a *hard* limit on what you can and can't express. Unfortunately, the limits of OWL 2 DL can suddenly pop up in the most unexpected places, and OWL representations often appear to be in good shape until a seemingly minor change brings the whole thing to shambles.

The semantics of a DL knowledge representation language defines a space

Class Membership	Property Assertions	Model of **A**
$a \in \mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{C} \sqcap \mathsf{D}$	a related to itself	7.2a
$a \in \mathsf{A} \sqcap \mathsf{D}$ and $b \in \mathsf{B} \sqcap \mathsf{C}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,b),$ $\mathsf{p}_1(b,a), \mathsf{p}_4(b,a)$	7.2b
$a \in \mathsf{A}$, $b \in \mathsf{B} \sqcap \mathsf{C}$ and $d \in \mathsf{D}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,b),$ $\mathsf{p}_1(b,d), \mathsf{p}_4(b,d)$	7.2c
$a \in \mathsf{A}$, $b \in \mathsf{B}$ and $d', d'' \in \mathsf{D}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,b),$ $\mathsf{p}_1(b,d'), \mathsf{p}_4(b,d'')$	7.2d
$a \in \mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{C}$ and $d \in \mathsf{D}$	$\mathsf{p}_2(a,a), \mathsf{p}_3(a,a),$ $\mathsf{p}_1(a,d), \mathsf{p}_4(a,d)$	7.2e
$a \in \mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{C}$ and $d', d'' \in \mathsf{D}$	$\mathsf{p}_2(a,a), \mathsf{p}_3(a,a),$ $\mathsf{p}_1(a,d'), \mathsf{p}_4(a,d'')$	7.2f
$a \in \mathsf{A} \sqcap \mathsf{C}$, $b \in \mathsf{B}$ and $d \in \mathsf{D}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,a),$ $\mathsf{p}_1(b,d), \mathsf{p}_4(a,d)$	7.2g
$a \in \mathsf{A}, b \in \mathsf{B}, c \in \mathsf{C}$ and $d, d', d'' \in \mathsf{D}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,c),$ $\mathsf{p}_1(b,d'), \mathsf{p}_4(c,d'')$	7.2h
$a \in \mathsf{A}, b \in \mathsf{B}, c \in \mathsf{C}$ and $d, d', d'' \in \mathsf{D}$	$\mathsf{p}_2(a,b), \mathsf{p}_3(a,c),$ $\mathsf{p}_1(b,d), \mathsf{p}_4(c,d)$	7.2i

Table 7.1: Valid models of the class A as depicted by figures 7.2a – 7.2i.

of describable valid models (Section 2.5.1). Given the set of language primitives provided by the language, we can go about carving up that space by ruling out certain models. For example, an owl:disjointWith relation between two classes A and B rules out all models where some individual *i* is a member of both. In general terms, the problem is that the surface structure of class level descriptions does not correspond to the structure of allowed models. There are no heuristics in place that assist a knowledge engineer in determining what primitives to use when.

As illustration, consider an ontology $\mathcal{O}$ with primitive classes A, B, C and D. We introduce roles $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3$ and p_4 with no domain or range defined, and give the following description of class A:

$$\begin{aligned} \mathsf{A} \quad \equiv \quad & \mathsf{p}_2 \textbf{ some } (\mathsf{B} \sqcap \mathsf{p}_1 \textbf{ some } \mathsf{D}) \\ & \sqcap \mathsf{p}_3 \textbf{ some } (\mathsf{C} \sqcap \mathsf{p}_4 \textbf{ some } \mathsf{D}) \end{aligned}$$

At the *class* level, this definition of A has a structure that resembles the shape of a diamond (see figure Figure 7.1), but this structure is not reflected at the *instance* level. When we assert the individuals a, b, c, d, d' and d'', the definition of A may allow (at least) all patterns of relations between these individuals depicted in Figure 7.2, given the assertions of Table 7.1.

Conversely, we might want to achieve a pattern similar to figure 7.2i at the *instance* level, using a much simpler structure at the *class* level. This is the case when one or more of the classes A, B, C or D are the same, or when some of the roles $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3$ and p_4 are equivalent. Achieving a representation of A that adequately covers the intended structure, comes down to finding ways to either prevent or ensure the possibility of a single individual to play multiple roles. It requires us to prune the set of possible models that satisfy the class description and achieve complete *lockdown* on unintended interpretations. As with a complicated phase in a game of Twister, the slightest slip can have significant

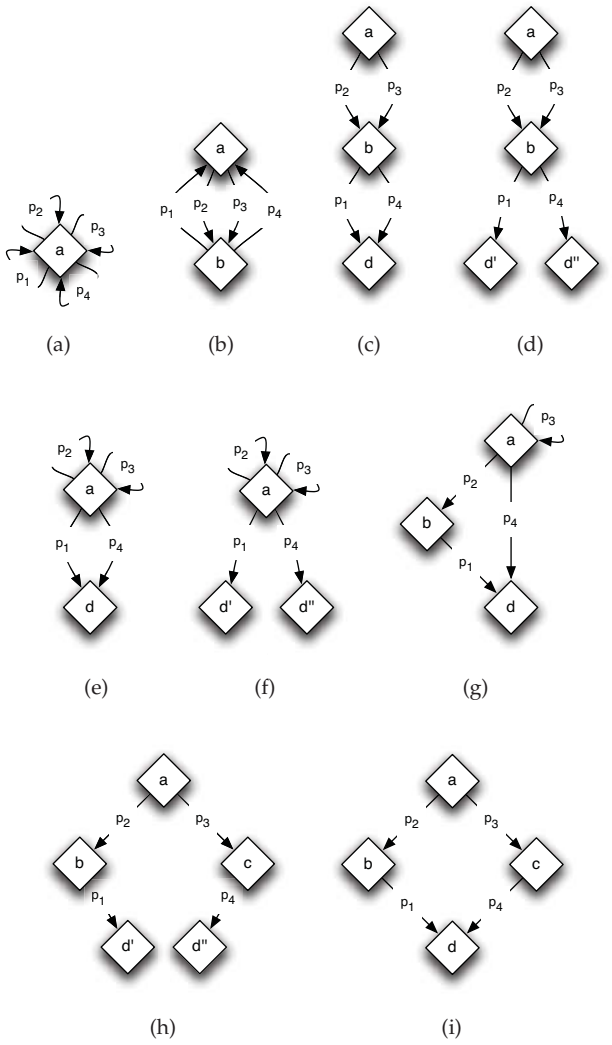

Figure 7.2: Valid models of class A

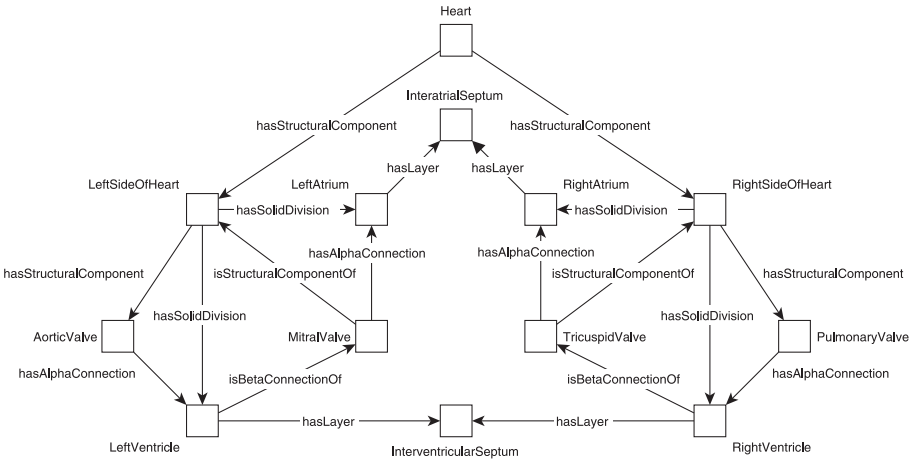

Figure 7.3: Structured object: the heart (Motik et al., 2007a)

consequences.

Fortunately, there is significant progress with respect to automatic debugging and explanation facilities in e.g. Pellet and Protégé 4 based on belief revision (Halaschek-Wiener et al., 2006). The Tweezers extension of Pellet, described in Wang and Parsia (2007), allows an engineer to examine the models generated by its tableaux algorithm.[1]

7.1.2 Structured Concepts

Arguably, the limits of expressiveness become increasingly more tangible as the complexity of the domain increases. In fact, many relatively simple domains involve structures that are not necessarily complex by themselves, but *do* push the limits of description logics and consequently result in intricate representation. For instance, the physical structure of organs, molecules, artefacts, devices, but also the interaction patterns between actions, processes and their participants are complex combinations of interdependent entities that are hard, if not impossible to represent in DL.

Motik et al. (2007a) emphasise the need to be able to describe *structured objects*, objects that are defined primarily by the (internal) composition of their parts. A good example, also employed by Motik et al. is the human heart (see Figure 7.3). The heart consists of four 'chambers', the left and right atria, and the left and right ventricles, each of these are separated by a septum, the interatrial and ventricular septa, respectively. However, in OWL DL it is impossible to express that e.g. the atria are separated by the *same* septum.

In SUMO, complex concepts are represented as rules in Common Logic (ISO/IEC, 2007), a highly expressive but undecidable language. This allows one to represent the reciprocity constraints in a direct way, by means of variables. As discussed by Grosof et al. (2003), decidable Horn logic programs

[1] See `http://www.mindswap.org/~tw7/work/profiling/code/`, and its recent incarnation as a Protégé 4 plugin, SuperModel, at `http://www.cs.man.ac.uk/~bauerj/supermodel/`.

allow the definition of non tree-like structures in a straightforward manner. For instance, a definition of the class A, based on the pattern of Figure 7.1 can be captured by the following rule:

$$\begin{aligned} \mathsf{A}(?a) \quad \leftarrow \quad & \mathsf{p}_2(?a,?c) \wedge \mathsf{p}_3(?a,?b) \wedge \mathsf{C}(?c) \wedge \mathsf{B}(?b) \\ & \wedge \mathsf{p}_1(?c,?d) \wedge \mathsf{p}_4(?b,?d) \wedge \mathsf{D}(?d) \end{aligned}$$

At first sight, it may seem beneficial to adopt an approach that combines a rule-based formalism with OWL 2 DL. However, this combination has several drawbacks. First of all, rules bring back the epistemological promiscuity and order dependence of control knowledge. Furthermore, like the procedural and interpretive attachments of e.g. KRL and KL-ONE (Bobrow and Winograd, 1976; Brachman, 1979, and Section 2.2.5), they can only *assert* new knowledge. Rules cannot aid in posing extra restrictions for classification: they cannot *prevent* classification in the way that integrity constraints in databases do (Hoekstra et al., 2006). Although Motik et al. (2007b) and others have made progress in combining DL with integrity constraints, these developments have not as yet found their way into applications.

Maintaining consistency between inferences of the classifier and the rule interpreter is not trivial. This is largely a consequence of the safeness condition necessary for decidable combinations of rule and DL formalisms (Motik et al., 2005). Because variables in DL-safe rules are only allowed to be bound to known, *named* individuals, a rule interpreter will not take the possibly infinite number of anonymous individuals inferred by an OWL 2 DL reasoner into account: rules are likely to quantify over incomplete models of the TBox.

Hybrid approaches that reconcile DL with rules, such as DLP (Grosof et al., 2003), Horn-$\mathcal{SHIQ}$ (Krötzsch et al., 2006), and the recent ELP (Krötzsch et al., 2008b) and $\mathcal{SROIQ}$-Rules (Krötzsch et al., 2008c) inevitably sacrifice expressive power to constrain reasoning complexity. For instance, DLP is defined as the intersection of DL and logic programming languages,[2] the ELP language requires variables in the body of a rule to be connected either via a tree-shaped graph or a set of such graphs, the $\mathcal{SROIQ}$-Rules language is restricted to those rule-like constructs expressible in the $\mathcal{SROIQ}$ DL. In short, languages either provide expressive rules with limited DL, or expressive DL with limited rules. As a consequence, complex concepts are often defined using a language of the former form, even though this is not always necessary. The question remains: what *can* we still meaningfully express in OWL 2 DL?

7.1.3 Three Patterns

This chapter discusses OWL 2 DL representations of three frequently occurring patterns. Rather than full fledged class definitions, these patterns are presented by the way they are assembled using the OWL 2 primitives discussed in Chapter 3. In this description, the various design decisions and limitations of the representation are discussed in detail, and for each pattern it is shown how it can be applied to various domains.

The first pattern concerns the *diamond* structure briefly discussed in the preceding section. Section 7.2 is an extended version of Hoekstra and Breuker

[2] This approach lies at the heart of the OWL 2 RL profile of OWL 2. See Section 3.5.2.

(2008), and investigates ways to approximate this structure using some of the more expressive language features of OWL 2 DL. Central to this pattern is the ability to infer identity relations between different participants in the pattern.

The second pattern is concerned with the representation of relations and their reifications. Section 7.3 discusses how several use cases, such as n-ary relations and subjective entities, can be reduced to the problem of inferring a relation between two individuals that participate in the reification of that relation. The pattern describes *triangular* structures, which can be combined to define complex (social) relations.

The third pattern allows the representation of *sequences* of multiple entities. Section 7.4 presents the pattern and shows how it can be applied in the definition of causal relations and complex protein structures. The pattern relies heavily on the role inclusion axioms of OWL 2 and is compared to an alternative pattern described by Drummond et al. (2006).

Finally, in Section 7.5 the three patterns are combined in the representation of the Action class of LKIF Core. The wide applicability and reusability of this pattern-based representation shows significant benefits compared to the classical middle-out approach of Section 5.2. Nonetheless, it is quite possible that the patterns discussed here gather their applicability from the basic nature of the concepts that triggered their original representation in the first place.

The patterns discussed in this chapter are representations of notions that stand in the spotlight of theoretical discussion in both philosophy and AI. In line with the knowledge representation perspective of the preceding chapters, these representations are illustrations of the capabilities and usability of OWL 2 DL in a variety of domains and make no claim with respect to ontological, nor theoretical correctness. Knowledge representation is a means to an end, and we can only aim to *approximate* reality.

7.2 Grasping the Diamond: *The Reciprocity of Exchange*

Concepts in general ontologies are basic building blocks for the definition of more specific and compound notions. They are not only domain independent and generally applicable, they are necessarily also abstract and simple. In various papers we have advocated a crisp distinction between basic ontological categories, and more complex notions, captured in *frameworks* (Breuker and Hoekstra, 2004c; Hoekstra et al., 2007, and Section 5.5.2). Although this is the general rule, some compound concepts are of a clear ontological nature. Construction of the LKIF Core ontology prompted us to consider a term frequently occurring in law – e.g. in contracts – that is both very abstract, and composite: the *transaction*. Transactions bind two actions by *reciprocity*, which is particularly reflected by identity and other constraints on roles in the two constituting actions. It forms a graph pattern that is diamond-shaped (see Figure 7.4).

A generic use case (cf. Gangemi (2005)) that covers these reciprocal combinations is the notion of *exchange*. Exchange not only pervades the full range of physical, mental and abstract changes, but also takes a very abstract position in ontologies such as LKIF-Core and SUMO: it is an unavoidable concept and appears in various forms, e.g. in the definition of financial transaction of

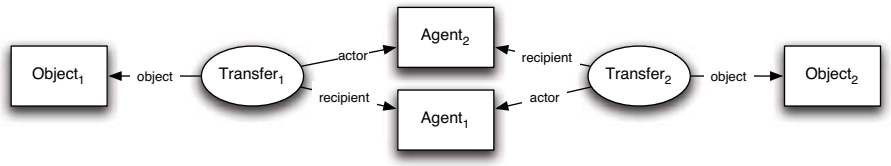

Figure 7.4: Structure of a transaction

SUMO (Niles and Pease, 2001). Entities such as objects and processes can undergo any of three kinds of change. In a *change of value*, the value of a property of an entity changes: e.g. a traffic light changes to green; a car accelerates. *Transfer* is a change of a value of a (anti-rigid) property that is not inherent to the entity: e.g. an object is moved to another position, a person graduates. Usually the property that changes is a position. A *transformation* is the change of a rigid property: e.g. a solid physical object melts to form a liquid. All three types of change can occur in the context of an exchange. A trivial example of ex-'change-of-value' is changing currencies. Metabolism is an example of an exchange consisting of transformations where material and energy are exchanged between an organism and its environment.

However, exchange involves additional constraints with regard to reciprocity. For instance, a sales transaction consists of two transfer actions: a buyer receives some goods for which he pays money, and the seller receives that *same* money and provides the *same* goods (Figure 7.4). For any exchange, the reciprocity constraints are:[3]

Identity
the active participants in both transfers are the same. For instance, the agent that is the actor of the money transfer (paying) is also the receiver of the goods. The agent that is the actor that transfers the goods, i.e. the seller, is the receiver of the money.

Balance
the objects in both transfers, i.e. the things being exchanged, should be comparable in some way. For instance, the value of the money should be balanced by the value of the goods.

A position metaphor can be applied to the participants of an exchange, e.g. goods and money are said to *trade places*. The same metaphor holds in a legal context for the exchange of ownership: ownership is a *legal* position, a right Hohfeld (1919). Again, legal positions themselves involve reciprocity as well: the right to own is balanced by a duty to pay.

Although indeed this section focuses on *exchange* as primary use case, similar structures can be found in more static notions such as physical (situation) descriptions, biological structures (Motik et al., 2009), dependency diagrams, and more theoretical notions such as Hohfeldian squares (Hohfeld, 1919).

In the next section, the representation of transactions is described in a step-by-step fashion, showing at the same time the practical knowledge pattern,

[3]Observe that the words 'buying' and 'selling' refer to the same conceptual entity. They are not synonymous; the choice of word is determined by the linguistic-pragmatic context.

and a hands-on methodology for approximating diamond-like structures using OWL 2 DL primitives.

7.2.1 Representing Transaction

A knowledgeable user of OWL will most likely only encounter real problems when dealing with elaborate, highly structured concepts. Often these problems are so big, or just insurmountable, that we resort to a rule-based representation. For the reasons discussed in Section 7.1.1, this is undesirable or at least should be avoided when possible. Only when the expressive power of either one of the formalisms is pushed to its limits, problematic interactions between the two formalisms can be avoided.

Just like the human heart, used as example by Motik et al. (2007a), the notion of transaction is a structured concept. The following sections take this use case to elaborate on the tools we can use to eliminate some of the undesirable models allowed by an initial, naive representation. The purpose of the final representation is twofold:

Identification
to be able to correctly *recognise* an individual transaction as a member of a transaction class, but also

Enforcement
to generate a violation when an individual is incorrectly asserted to be a member of that class.

The methodology consists of a succession of five steps:

Step 1: Initial Class Definition

The straightforward definition of a transaction is a Transaction class that has transfer actions Transfer as parts. In turn, each Transfer has an actor and recipient property with some agent Agent as value, and an object relation with an object Object:

$$\begin{aligned} \mathsf{Transaction} &\equiv \mathsf{part}\ \textbf{some}\ \mathsf{Transfer} \\ \mathsf{Transfer} &\equiv \mathsf{actor}\ \textbf{some}\ \mathsf{Agent} \sqcap \mathsf{recipient}\ \textbf{some}\ \mathsf{Agent} \\ &\quad \sqcap \mathsf{object}\ \textbf{some}\ \mathsf{Object} \end{aligned}$$

Step 2: Constrain the Number of Role Fillers

Evidently, the above definition only prescribes that there must be at least one part relation with a Transfer. We can fine tune this definition by specifying that a Transaction has only instances of Transfer as its parts and provide a cardinality restriction on the number of part relations allowed. However, this is too strict as it no longer allows for any other part relations either. In OWL DL 1.0 the only way out of this predicament was to introduce a specific has_transfer sub property of part for the Transaction class. For an individual to be inferred to be a Transaction, it should be expressed using the (class specific) has_transfer relation. However, using the qualified cardinality restrictions of OWL 2 DL

we can specify that a Transaction should have exactly two part relations to a Transfer individual:

$$\text{Transaction} \sqsubseteq \text{part } \textbf{exactly } 2 \text{ Transfer}$$

Identification of individuals can only be achieved by means of equivalence axioms in combination with the domain and range of a property. Some types of restrictions cannot be used for identification. Because of the open world assumption, an upper bound on the cardinality of a property at some class can not be used to *identify* an individual to be a member of that class, as there is no way to know whether the individual has additional properties of that type. For the same reason, universal restrictions cannot be used for identification using equivalence axioms.

Once the type of an individual is established, additional subclass axioms can be used to infer new properties of the individual: it is shaped to conform with the pattern expressed in the restriction. Of course, this approach can be problematic when the equivalence axiom is under specified, as too many individuals may be inferred to be a member of that class. Using this methodology, we can refine the Transaction class as follows:

$$\begin{aligned} \text{Transaction} &\equiv \text{part } \textbf{some } \text{Transfer} \sqcap \text{part } \textbf{min } 2 \text{ Transfer} \\ &\sqsubseteq \text{part } \textbf{only } \text{Transfer} \sqcap \text{part } \textbf{exactly } 2 \text{ Transfer} \end{aligned}$$

Admittedly, this definition of Transaction is quite strict, as it does not allow for anything having more than two Transfers as part. For any individual that does have more than two, the reasoner will infer some of the Transfer individuals to be the same. Asserting that these individuals are mutually different, will make the ABox incoherent. Of course, we can overcome this limitation by refining the definition of Transfer to include only those transfer actions taking part in a proper transaction. However, simply stating that every Transfer should be part of a transaction does not add any new information. Nonetheless, requiring that any Transaction should have (at least) two individual Transfers as part ensures that the definition is not 'triggered' for individuals that are not asserted to be mutually distinct.[4]

Step 3: Disambiguate Role Fillers

Let's turn our attention to Transfer; we restrict it in a similar fashion:[5]

$$\begin{aligned} \text{Transfer} &\equiv \text{actor } \textbf{some } \text{Agent} \sqcap \text{recipient } \textbf{some } \text{Agent} \\ &\quad \sqcap \text{object } \textbf{some } \text{Object} \\ &\sqsubseteq \text{actor } \textbf{exactly } 1 \text{ Agent} \sqcap \text{recipient } \textbf{exactly } 1 \text{ Agent} \\ &\quad \sqcap \text{object } \textbf{exactly } 1 \text{ Object} \end{aligned}$$

In fact, the definition of Transfer adds complexity because it involves *distinct properties* and different *ranges*. However, this heterogeneity allows us to drop

[4]Note that the distinctness of two individuals cannot be enforced by means of a class description, as the open world assumption allows for the existence of some other (unspecified) distinct individual.

[5]For the current purposes, the participant properties could be made *functional* thereby waiving the necessity for qualified cardinality conditions. However, functionality is a rather strong claim as it is a global restriction whereas cardinality restrictions operate locally on a class and its subclasses.

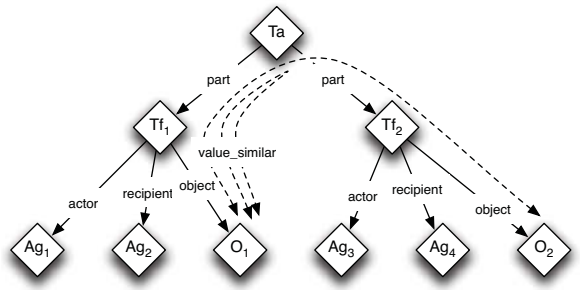

Figure 7.5: Tree-model as described by the definitions of Transaction and Transfer. The dotted arrows depict the path along the tree by the property 'value_similar'.

the owl:allValuesFrom axiom as the combination of exact cardinality and existential restriction has the same effect. The complexity resides in that we need a way to distinguish the actor from the recipient. The above definition does indeed ensure that we have at least an actor and a recipient for any Transfer, but these individuals may still be the same. Asserting disjointness between Agents in different roles is not desirable, as this eliminates all models where each Agent fulfils both roles at the same time. Fortunately, OWL 2 introduces the ability to make the extensions of the properties disjoint.[6]

Step 4: Traverse the Tree

Figure 7.5 shows the completion graph for the definitions of Transaction and Transfer we have so far. As one can see it is tree-shaped and e.g. does not require that the actor of transfer Tf_1 and the recipient of Tf_2, Ag_1 and Ag_4 respectively, are the same. Because of the tree-model property of $\mathcal{SROIQ}(D)$, we cannot further extend these definitions to enforce such a restrictions. However, we can still *infer* a lot of information that we can use to further restrict the various participants in the transaction.

We first shift our attention to the two Object individuals O_1 and O_2 appearing in the two transfers. As we discuss in Section 7.2, these two objects need to have a balance in 'value': good vs. money in transactions, kinetic vs. potential energy in collisions, etc. OWL 2 DL property chains specify the propagation of a property along some path of interconnected properties. We introduce the property value_similar that connects the two objects of the transaction as follows:

$$\mathsf{object}^- \circ \mathsf{part}^- \circ \mathsf{part} \circ \mathsf{object} \sqsubseteq \mathsf{value\_similar}$$

Because of the left-right symmetry of the tree-model of Transaction the value_similar property is symmetric and reflexive; because both O_1 value_similar O_2 and O_2 value_similar O_1 hold, and the two part relations connecting Ta to Tf_1 and Tf_2 cannot be distinguished.

[6] i.e. for any two individuals x, y and actor(x, y) there exists no recipient(x, y).

We could similarly specify a symmetric and transitive same_id_as property that specifies (in a custom manner) that two individuals share identity:

$$\begin{aligned} \text{actor}^- \circ \text{part}^- \circ \text{part} \circ \text{recipient} &\sqsubseteq \text{same\_id\_as} \\ \text{recipient}^- \circ \text{part}^- \circ \text{part} \circ \text{actor} &\sqsubseteq \text{same\_id\_as} \end{aligned}$$

Step 5: Introduce Asymmetry

Unfortunately, this chain is overly general as it infers same_id_as relations between the actor and recipient participating in a single action. Again, this is caused by the symmetry of the two part branches of the completion graph of Transaction. As in OWL 2, complex properties such as property chains cannot be disjoint, the only way to rule out models in which the two participants are inferred to be the same, is to make the tree itself asymmetric. We can do this in two ways.

Firstly, we could make the generic definition of Transaction intrinsically asymmetric by disambiguating the left-hand side and right-hand side of the tree using two distinct part properties. This has the drawback we had before, i.e. it is rather ad hoc and makes identification tautological. Secondly, rather than enforcing asymmetry, we can exploit available asymmetry by disambiguating the two sides of the tree via the nature of the participants in the two transfers. This solution has the drawback that the participants on both sides need to be suitably distinct, and that sameness can only be expressed in terms of the characteristics of those participants. In other words, it increases domain dependence of the representation: the same_id_as relation can not be inferred for the generic Transaction class.

Granted that the reader agrees that the first disadvantage outweighs the second, we continue to disambiguate between different participants. Suppose our transaction is a sales transaction where some Good is exchanged for a quantity of Money. We can use this information to discriminate between the two Transfers that make up our Transaction:

$$\text{Goods\_Transfer} \equiv \text{Transfer} \sqcap \text{object } \mathbf{some} \text{ Good}$$

Again, the equivalence axiom is used to grab the Transfer individuals that involve goods. We can then use our strict prior definition of Transfer to enforce object specific relations between the Transfer and the actor and recipient:

$$\begin{aligned} \text{recipient}_g &\sqsubseteq \text{recipient} \\ \text{actor}_g &\sqsubseteq \text{actor} \\ \text{Goods\_Transfer} &\sqsubseteq \text{recipient}_g \ \mathbf{some} \ \text{Agent} \sqcap \text{actor}_g \ \mathbf{some} \ \text{Agent} \end{aligned}$$

The money transfer Money_Transfer and properties can be defined accordingly. Because of the exact qualified cardinality constraints on Transfer for the participants object, actor and recipient, a DL reasoner will infer that for any Goods_Transfer g and Agent a, if g actor a then g actor_g a. Based on this inference, we

can rephrase the same_id_as relation in terms of context specific relations:

$$\begin{aligned}
\mathsf{actor}_g^- \circ \mathsf{part}^- \circ \mathsf{part} \circ \mathsf{recipient}_m &\sqsubseteq \mathsf{same_id_as} \\
\mathsf{actor}_m^- \circ \mathsf{part}^- \circ \mathsf{part} \circ \mathsf{recipient}_g &\sqsubseteq \mathsf{same_id_as} \\
\mathsf{recipient}_g^- \circ \mathsf{part}^- \circ \mathsf{part} \circ \mathsf{actor}_m &\sqsubseteq \mathsf{same_id_as} \\
\mathsf{recipient}_m^- \circ \mathsf{part}^- \circ \mathsf{part} \circ \mathsf{actor}_g &\sqsubseteq \mathsf{same_id_as}
\end{aligned}$$

Given the transaction Ta depicted in Figure 7.5, and O_1 : Money and O_2 : Good we can now infer the relations Ag_1 same_id_as Ag_4 and Ag_2 same_id_as Ag_3.[7]

7.2.2 Discussion

The notion of exchange, and more specifically transaction, represents a commonly recurring pattern in many domains. Although it is a concept primarily characterised by its structure, rather than inherent properties, it has a strong ontological flavour. The reciprocity of the various participants in an exchange is similar to constraints in representations of physical artefacts.

The structure of this complex concept can be approximated by exploiting a combination of the newly introduced language constructs in OWL 2 DL. The previous section describes a general methodology that can be applied to create approximate representations of *any* diamond shaped structure. It is composed of five straightforward steps. Create *initial class definitions* that provide a basic means for identifying individuals. Constrain the *number of role fillers* allowed for members of that class, by combining upper and lower bound cardinality restrictions using both subclass and equivalence restrictions, respectively. *Disambiguate role fillers* by making property extensions disjoint. Define property chains that *traverse the tree* to create (identity) relations between different participants in a transaction (or similar pattern). And finally, introduce *asymmetry* by incorporating contextual information, e.g. by using less generic concepts to define the more general relation.

These steps ensure both that candidate individuals in the ABox are appropriately *identified*, and that inferred property values for those individuals are *enforced*. Explicit violation of such inferences, e.g. by means of a negative object property assertion in OWL 2 DL for the same_id_as property, results in an incoherent ABox. However, the same_id_as and other relations introduced in this pattern are *inferred* on the basis of property chains, but they do not enforce the existence of a chain when they are *asserted*. A chain of properties can only be defined as a sub property of another property, but it cannot be stated to be equivalent: the super property does not *define* the chain. This is because if this were possible, we could specify a recursive property, e.g. $\mathsf{p} \circ \mathsf{r} \equiv \mathsf{r}$, and thereby force the existence of an infinite property chain using an existential or cardinality restriction.[8] Therefore, for the pattern to 'work' – and this holds

[7] The good and money in the example can be enforced to be different individuals by making the Money and Good classes disjoint. If this is undesirable (i.e. because money can be considered a good as well), this does raise the need for a specific Sales_Transaction class that consists only of a Money_Transfer of Money that is no Good. Alternatively, one can introduce a custom different_id_from property along the lines of same_id_as.

[8] Note that this is different from an existential restriction on a transitive property, as transitive properties *may* have an infinite length, but do not *have* to.

for all patterns in this chapter – care has to be taken to distinguish between inferable and 'assertable' properties.

Admittedly, because we can only approximate the diamond shape, the most conspicuous drawback is the need for an alternative to owl:sameAs to express that two individuals are 'the same'. On the other hand, in some cases a custom relation is even to be preferred over the owl:sameAs relation such as when it is used to relate two different states of the same object. For instance, transactions can be conceived of as involving *change*, where the state of affairs before the transaction should be distinguished from the result of the transaction (see Section 7.4). In this case, a same_id_as relation can express a shared identity between logically different individuals, where asserting the owl:sameAs relation would result in an incoherent ABox (Hoekstra et al., 2006).

An added advantage is that the pattern steers clear of problems arising from the overlapping expressiveness of DL and logic programs. To close the diamond, we need only define a single, relatively harmless DL-safe rule that maps the same_id_as property to owl:sameAs. However, such a rule cannot infer that some individual is a participant in a complementary action when the participants of that action are not explicitly asserted. An alternative is to use nominal classes as 'placeholders' on the different individuals in the pattern. For instance, the definition of Transaction could be amended in a way that it enforces all actor role fillers of its Transfer parts have the value of a single individual *transaction_actor*. A match of some individual with the Transaction class description will then infer owl:sameAs relations between the (possibly different) individuals that fill the actor roles in that particular case, and the *transaction_actor*. This is only useful in a controlled environment where the structure of individuals that matches a class only occurs once. When multiple transaction individuals match the class description, their actors will be inferred to be the same as well.

7.3 Reification and Summarisation: *Relations and Social Reality*

This section describes a pattern that allows for the *summarisation* of reified relations. It is argued that although a reified relation often conveys a stronger ontological commitment than the original relation, while at the same time the original relation is often more convenient for practical use. The two sides of the coin are discussed in the context of two use cases: the representation of *n-ary relations* (Noy and Rector, 2006) and a description of *social reality* (Searle, 1995).

7.3.1 N-Ary Relations

One of the design patterns discussed by the Semantic Web Best Practices and Deployment (SWBP) working group of the W3C concerns the representation of so-called n-ary relations: relations between more than two entities (Noy and Rector, 2006).[9] The need for a 'best practices document' on the representation of n-ary relations in OWL DL originates from two phenomena. Because OWL

[9] See `http://www.w3.org/2001/sw/BestPractices`.

does not provide means to express n-ary relations directly, many other representation languages do have built-in support for these relations: the document addresses a *translation problem* for users coming from a different background. For instance, formal ontologists tend to represent ontologies using expressive languages such as traditional first order logic or other languages that allow the definition of such predicates, cf. CommonLogic (ISO/IEC, 2007). At the other end of the spectrum, the TopicMaps[10] language allows for associations with multiple subjects.

Noy and Rector (2006) discuss three use cases where n-ary relations would be in order:[11]

> **Case 1**
> "Christine has breast tumour with high probability."
>
> This phrase hints at a binary relation between the person *Christine* and diagnosis *Breast_Tumor_Christine* with a qualitative probability value describing this relation as *high*: the binary relation really needs a further argument.
>
> **Case 2**
> "Steve has temperature, which is high, but falling."
>
> In this case, *Steve* has two values for two different aspects of a has_temperature relation: its magnitude is *high* and its trend is *falling*. The two binary properties always go together, and should be represented as one n-ary relation.
>
> **Case 3**
> "John buys a 'Lenny the Lion' book from `http://books.example.com` for $15 as a birthday gift."
>
> Here, there is a single relation in which individual *John*, entity `http://books.example.com` and the book *Lenny_the_Lion* participate. This relation has other components as well such as the purpose (*birthday_gift*) and the amount (*$15*).

The solution proposed by Noy and Rector is to represent such n-ary relations as a class with n properties that *reifies* the relation.[12] For instance, the has_diagnosis relation between *Christine* and *Breast_Tumor_Christine* is represented as an individual of type Diagnosis_Relation with properties value *Breast_Tumor_Christine* and probability *high*. Similarly, *Steve*'s has_temperature is represented as an individual Temperature_Observation, and *John*'s book purchase is represented as a Purchase individual (see Figure 7.21).

Regarding the first case, this solution is similar to the *association class* in class diagrams of UML that directly represents some association (relation) between two classes. The association class itself can then have additional properties, such as in Figure 7.6. Association classes can only be used to reify a *binary* relation where the relation itself has additional properties.

[10] See `http://www.topicmaps.org`

[11] Examples and text taken from `http://www.w3.org/TR/swbp-n-aryRelations/`. Noy and Rector describe a fourth use case where a single object is related to an ordered list of other objects, see Section 7.4.

[12] Noy and Rector (2006) avoid the use of this term because of differing interpretation of the term in TopicMaps and RDF.

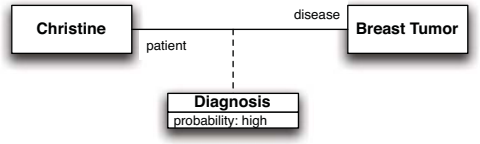

Figure 7.6: The association class 'Diagnosis' in UML, from `http://www.ibm.com/developerworks`

Noy and Rector do not address the fact that these cases are based on the a priori ontological commitment that the individuals involved are part of a *single relation*. How does such a commitment come about? In knowledge representation, and ontology construction in particular, there exists a rather prominent tradition that advocates isomorphism to natural language. This tradition is in part upheld by those having a background in natural language understanding and semantics, whose primary interest is to represent exactly what is expressed in a sentence. A stronger influence is the philosophical stance in ontology that the categories of existence lie at the basis of human cognition. It is furthermore argued in a semiotic tradition, that human cognition, or rather, the categories of human thought, are best conveyed through studying natural language (cf. the discussion of the DOLCE ontology in Section 6.2.1).

The decision to represent some description as a relation is often guided by the common rule of thumb that nouns (or noun phrases) represent concepts where verbs represent relations. Additionally, nouns preceded by 'has', as e.g. 'has temperature' are usually represented as properties as they indicate descriptions that cannot stand on their own: a temperature is always the temperature *of something* (Brachman et al., 1991; Sowa, 2000). A description preceded by a proposition such as 'with' is a modifier of some relation. Adjectives such as 'high' indicate values of relations.

It is these kinds of simple rules that are exploited by controlled natural languages for OWL such as Attempto Controlled English (Kaljurand and Fuchs, 2007, ACE), the Sidney OWL Syntax (Cregan et al., 2007) and Rabbit (Schwitter et al., 2008). There is a reason that these languages are *controlled*: natural language is itself highly mouldable. Consider for instance the case of Christine in Table 7.2. These examples all describe approximately the same thing; they are paraphrases, and casual reading of these sentences seems to convey the same information. However, when we adopt the rules of Brachman et al. (1991), the possible resulting models may become strikingly different for each of the sentences.

The gist of this exercise is to show that the need for a language construct, such as n-ary relations should be based on a conscious decision to interpret a use case in a particular way: it is an ontological commitment. A too direct connection to the way information is conveyed in linguistic expressions may introduce a commitment for cases where the conceptualisation underlying the expression makes no such commitment. Conversely, the proposed alternative representations of cases 1-3 make explicit two additional ontological commitments:

- a commitment to the existence of some intermediate entity through which

1. *"Christine has breast tumour with high probability"*
   Some unspecified n-ary relation between Christine and breast tumour, where the probability of this relation is high;
2. *"Christine has a high probability of breast tumour"*
   A binary relation 'has high probability' between Christine and breast tumour;
3. *"Christine has a diagnosis of breast tumour with high probability"*
   Some binary relation between Christine and diagnosis, and two relations between diagnosis and breast tumour and high probability, respectively;
4. *"Christine is diagnosed as having breast tumour with high probability"*
   An binary relation 'is diagnosed' between Christine and breast tumour, and a probability of this relation with value high;
5. *"Christine's diagnosis is breast tumour with high probability"*
   Some binary relation between Christine and diagnosis of type breast tumour where the probability of this diagnosis is high.

Table 7.2: Different wording, different representation?

the various relata of the n-ary relation are connected.

- a commitment to the non-existence of the n-ary relation itself, for the intermediate entity and the n-ary relation cannot exist at the same time. This does not exclude the possible existence of additional relations between the relata of the intermediate entity.

These commitments are not arbitrary: the reification makes explicit some knowledge about the domain which would be left implicit in the n-ary relation. For each of the cases, the reified class cannot be anything other than respectively a diagnosis, observation or purchase. The n-ary relation is a simplification, or summary, of the underlying conceptual structure. The restriction to binary relations in OWL thus enforces a more ontologically concise representation. The drawback, on the other hand, is that exactly because of this commitment, the n-ary relation itself can no longer be represented. Nonetheless, the design pattern described in Section 7.3.3 can reclaim some of the relational character of the reification.

7.3.2 Social Reality

A similar case for reification can be made on the basis of the considerations underlying the intentional and legal levels of LKIF Core (discussed in sections 6.3, 6.3.2 and 6.3.3). Remember that in line with the world knowledge in Valente's functional ontology, the legal level is an abstraction of commonsense knowledge. This abstraction is similar to the way in which intentional and functional notions, are generalised over physical phenomena by the design and intentional stance of Dennett (1987). These notions are *social constructs* that can be attributed to, or imposed on brute facts, phenomena the existence of which does not depend on human agreement (Searle, 1995).

According to Searle, institutional facts are constructed by means of *constitutive* and *regulative* rules, rules of the form:

$$X \text{ counts as } Y \text{ in context } C$$

Examples of constitutive and regulative rules are, respectively:

- Bills issued by the Bureau of Engraving and Printing (X) count as money (Y) in the United States (C).
- For the purposes of this law (C), a house boat (X) is deemed to be a house (Y).

Where a regulative rule imposes additional restrictions on something (i.e. a house boat) to create an institutional fact (the house-boat-as-house) which can exist independently from that rule, the constitutive rule determines (in part) the possibility of existence of the institutional fact. Clearly, where regulative rules have a normative character, constitutive rules are definitional. It is therefore not surprising that the *counts-as* relation has received a lot of attention in AI and Law, most recently in Grossi et al. (2005); Grossi (2007).

Although the counts-as relation in constitutive rules attributes the properties of Y to X in a similar fashion as ordinary subsumption, it is very distinct in three ways:

- The counts-as relation is ontologically *subjective* and observer relative, and therefore *contextual*. It only holds in relation to a certain context, within a system of constitutive rules created by (collective) intentionality.
- The counts-as relation limits inheritance, and does not permit *substitutability* of the syntactic elements of rules. For instance, the statement "Money is the root of all evil" in conjunction with the constitutive rule introduced before, does not make that "Bills issued by the Bureau of Engraving and Printing count as the root of all evil in the United States.". This is because the context of the counts-as relation is intensional; the institutional fact cannot be defined by reference to statements outside the context.
- The counts-as relation can be used to connect *anti-rigid* with *rigid* classes (Guarino and Welty, 2004). For example, where "bills issued by the Bureau of Engraving and Printing" is rigid, the "money" class is anti-rigid.

These characteristics of the counts-as relation make that it cannot be represented using the subsumption relation, which is contextless, extensional and subject to ontological restrictions concerning its relata (Section 5.5.1). Social, institutional facts can accumulate in layers, but eventually need to bottom-out in brute facts. This logical priority of brute facts coincides with the layered perspective of Dennett's stances and corresponds with primacy of physical reality in the LKIF Core ontology.

A fourth characteristic of social constructs is that, because of their context dependency, they are often conceived of as a *relation* between the brute fact and its context. A well known example is the duality of roles as relations and classes (Steimann, 2000; Loebe, 2003). Steimann identifies three ways to represent roles, as *named places* in relations, as *specialisation* or *generalisation*, or as *adjunct instances*. Clearly, of these only the first and last conform to the intensional character of the counts-as relation, and only the latter is directly compatible with OWL.

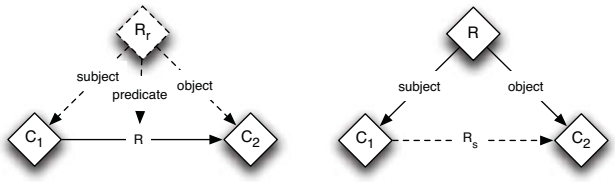

Figure 7.7: Reification versus summarisation of relation R.

Representing roles as adjunct instances means that the role itself and its player are defined as two distinct classes, with a played_by relation. For each occurrence where some entity plays a role, two individuals need to be present in the ABox of a description logic. However, for some purposes this representation may be non intuitive or overly verbose, and the relation between role player, role and context is 'flattened' to a relation between role player and context. This is essentially the named places approach, and although OWL does not support these, a similar effect can be reached by defining the named places as mutually inverse properties.

The role-as-relation approach is enticing in cases where the role-as-class approach involves an apparent circularity. To give an example, it is hard to consider the role Student independently from a student_of relation with a University class. Defining Student in terms of that relation is rather tautological. A similar example is the interplay between a propositional attitude Belief, and the proposition that is believed_by an agent. In the development of LKIF Core, we encountered various other occasions where this pattern emerged (see Section 7.3.4).

7.3.3 Representing Summarisation

Both the representation of social reality and n-ary relations involve an important ontological commitment to either a relation oriented approach (impossible in OWL for the n-ary case) or a class centred representation. As we have seen, in many cases the class centred approach is ontologically more concise, but also more verbose. For many applications it is useful, or simply convenient, to abstract away from this ontological commitment. This process of *summarisation* is in fact the reverse of reification. While reification turns a secondary entity into a primary entity – i.e. a relation is reified as a class – summarisation 'hides' a primary entity as a secondary entity. The use cases for n-ary relations of Section 7.3.1 show that a need for representing n-ary relations is usually motivated by relations that already hide an ontological commitment. Arguably, an explicit methodology for creating (or inferring) the summarisation of ontologically concise, but verbose representation can be very useful for e.g. ontology reuse. A reusing ontology may reuse just the abridged version of 'properties' without compromising its ontological commitment.

Figure 7.7 shows the reification R_r of relation R as an individual with explicit subject, predicate and object relations, and the converse summarisation R_s of individual R as a relation between classes C_1 and C_2. The use of the term 'reification' was discredited by Noy and Rector (2006) because of its different

meaning in different languages (Section 7.3.1). Even when we commit to the single interpretation of reification in RDF, it is clear that the RDF-solution does not meet our needs:

- In RDF, for any property an rdf:Statement can exist that reifies it (Section 3.3.1). However, the converse does not hold: given some rdf:Statement one cannot automatically infer the existence of a property assertion between rdf:subject and rdf:object of the type specified by rdf:predicate. This means that RDF is not expressive enough to support both reification and summarisation using a single mechanism.
- RDF reification is indiscriminate with respect to the type of the relation being reified, whereas the current use cases are restricted to n-ary relations and constitutive rules.
- Though OWL has an RDF serialisation and thus can be said to support its reification mechanism, reified RDF statements cannot be used in OWL axioms without violating the expressiveness constraints of OWL DL.

Given these considerations, the following sections outline a design pattern that allows us to infer the summary of a reified relation. First, the general principle is described in terms of the construction of social roles. This is then elaborated to show its application for n-ary relations and other subjective constructs. Note that the purpose of this exercise is not to specify contextual subsumption along the counts-as relation (cf. Grossi (2007)) in DL, but rather to allow maximum usability of ontologically concise representations.[13]

Step 1: Initial Class Definition

Consider the definition of a simple social rule, a student is a person who is enrolled as such at some university. Using Searle's counts-as rule, we can rephrase this as:

> A *person* (X) counts as a *student* (Y) if enrolled at some *university* (C)

Casting this into OWL axioms, we get:

$$\begin{aligned} \mathsf{Student} \quad \equiv \quad & \mathsf{played\_by}\ \textbf{some}\ \mathsf{Person} \sqcap \\ & \mathsf{context}\ \textbf{some}\ \mathsf{University} \end{aligned}$$

Step 2: Constrain and Disambiguate Role Fillers

The above definition does not constrain the number of and type of entities that are valid values for the context and counts_as properties. We import the Subjective_Entity class from the LKIF Core ontology to import its restrictions. A subjective entity is defined as an entity imposed on another entity using a

[13] Section 7.5.1 describes how a role attribution can be used to 'backfire' certain property values to the brute fact.

counts-as rule:[14]

$$\begin{aligned} \mathsf{Subjective\_Entity} &\equiv \mathsf{imposed\_on}\ \textbf{some}\ \mathsf{owl{:}Thing} \sqcap \\ &\quad\ \mathsf{context}\ \textbf{some}\ \mathsf{owl{:}Thing} \\ &\sqsubseteq \mathsf{imposed\_on}\ \textbf{exactly}\ 1\ \mathsf{owl{:}Thing} \sqcap \\ &\quad\ \mathsf{context}\ \textbf{exactly}\ 1\ \mathsf{owl{:}Thing} \end{aligned}$$

$$\mathsf{imposed\_on} \equiv \mathsf{counts\_as}^{-}$$

Note the way in which the equivalent class and subclass of axioms on Subjective_Entity are used to ensure that any entity that is attributed in some context is both recognised as a Subjective_Entity, but also enforced to be attributed to exactly one entity in a single context (Section 7.2.1). We reuse the definition of Social_Role in a similar fashion. A social role is a role that can only be played by a single agent, in a single context, it inherits a cardinality restriction on the played_by and context properties from Subjective_Entity. Where played_by is defined as the inverse of a plays subproperty of counts_as. The relevant axioms from LKIF Core are:

$$\begin{aligned} \mathsf{Role} &\equiv \mathsf{played\_by}\ \textbf{some}\ \mathsf{owl{:}Thing} \\ &\sqsubseteq \mathsf{Mental\_Entity} \\ \mathsf{Social\_Role} &\sqsubseteq \mathsf{Role} \sqcap \\ &\quad\ \mathsf{played\_by}\ \textbf{some}\ \mathsf{Agent} \end{aligned}$$

$$\begin{aligned} \mathsf{played\_by} &\equiv \mathsf{plays}^{-} \\ \mathsf{played\_by} &\sqsubseteq \mathsf{imposed\_on} \\ \mathsf{plays} &\sqsubseteq \mathsf{counts\_as} \end{aligned}$$

If we adopt these axioms, the Student role can only be played by a single person at a single university. It should furthermore be clear that the redefined definition of Student is a conservative extension of the LKIF ontology:

$$\mathsf{Student} \sqsubseteq \mathsf{Social\_Role}$$

At this point, the same problem as with our definition of Transaction in Section 7.2.1 arises. Currently, we have said nothing about how the classes Person and University are related. The cardinality restrictions on the played_by and context properties allow us to infer that, since Student is a subclass of Social_Role, the Person class must be subclass of Agent. The current representation is silent as to whether a University is an agent as well, and a university could in principle be a student in the context of itself.[15] We can use our knowledge of the domain to prevent this from happening and disambiguate universities from persons, i.e. by making both classes disjoint. However, this disambiguation between context and brute fact is a *global* restriction, and indeed the properties imposed_on and context are defined as disjoint in the LKIF ontology.

[14] For a more restrictive definition we could define imposed_on and context as functional properties.

[15] In fact, a University is itself a Role played by some Organisation (which is an Agent).

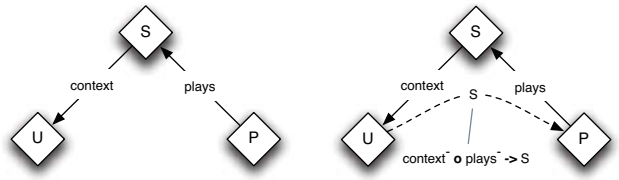

Figure 7.8: Tree-model as described by the definition of Student

Step 3: Traverse the Tree

Figure 7.8 shows the completion graph for the definition of Student, where the student role S is played by a person P within the university context U. As discussed earlier, this structure constitutes a reified relation expressing studentship between the university and the person. A straightforward summarisation of the reification can be specified as simple role inclusion axioms that simulate the respective named places:

$$\begin{aligned} \mathsf{context}^- \circ \mathsf{plays}^- &\sqsubseteq \mathsf{student} \\ \mathsf{plays} \circ \mathsf{context} &\sqsubseteq \mathsf{university} \end{aligned}$$

That is for the structure in Figure 7.8, a DL reasoner will infer a student relation between university U and person P, and its inverse. Because in role inclusion axioms the property chain is a sub property of the property, the summarisation is subject to a similar limitation as RDF reification, but in the exact opposite direction. Where in summarisation we can only infer the relation given the reified structure; RDF reification only infers the reified structure, given the relation. Also, summarisation cannot be used to express an explicit predicate relation between the summarised class (Student) and the relation (student).

Step 4: Introduce Domain Dependence

For the purposes of our example, this role inclusion axiom is overly ambitious as it will infer the relation for *any* context and brute fact connected through a Role. The way out of this predicament is to define the student relation in terms of some of the elements particular to the structure of the Student class, i.e. to introduce domain dependence. There are two ways to do this: either by refining the properties involved, or by using **self** restrictions.

We can refine the properties in the Student definition by adding the following axioms:

$$\begin{aligned} \mathsf{Student} &\sqsubseteq \mathsf{student\_context}\ \textbf{some}\ \mathsf{University} \sqcap \\ &\quad\ \mathsf{student\_played\_by}\ \textbf{some}\ \mathsf{Person} \\ \\ \mathsf{student\_context} &\sqsubseteq \mathsf{context} \\ \mathsf{student\_played\_by} &\sqsubseteq \mathsf{played\_by} \end{aligned}$$

Because of the cardinality restrictions on the context and imposed_on properties, the properties student_context and student_played_by will correspond to

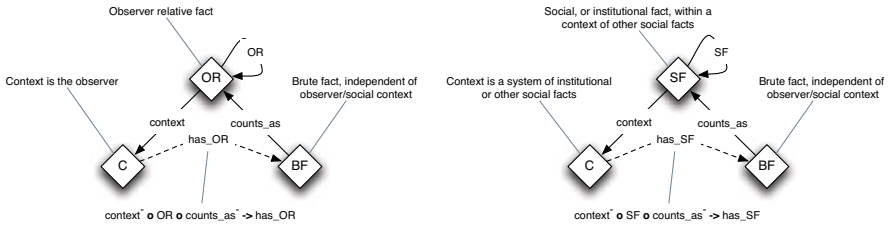

Figure 7.9: Observer-relative and Institutional facts

the context and played_by properties for any individual of type Student. Since the restriction on the domain dependent properties is only specified in a subclass axiom, inferring membership of the Student class does not depend on these properties. The student property can now be restated as:

$$\text{student\_context}^{-} \circ \text{student\_played\_by} \sqsubseteq \text{student}$$

The main drawback of this approach is that for every type of Role, we need to add a domain dependent property and axiom for each of the properties in its definition. Furthermore, these domain dependent properties can only be used local to the Role being defined. Since both properties cannot be used to infer class membership of Student, we can infer the student relation for any three individuals connected in the prescribed manner. Using equivalence instead of subsumption in the definition of Student does not help matters, as that may make the domain *in*dependent and dependent restrictions of the Student class equivalent.

A more economical approach is to reduce the opaqueness of the distinction between the TBox and RBox by introducing a single property that uniquely identifies a class in the RBox. For our Student role, we introduce the is_student property, that will relate any student individual to itself. We can ensure this by specifying a **self** restriction on the Student class:

$$\text{Student} \sqsubseteq \text{is\_student } \mathbf{some\ self}$$

Any individual that is related to itself via the is_student property will be identified as an instance of Student, and any individual asserted as instance of Student will be related to itself via that property. We can now rephrase the student summarisation axiom as follows:

$$\text{context}^{-} \circ \text{is\_student} \circ \text{played\_by} \sqsubseteq \text{student}$$

The university summarisation property can be expressed by reversing the direction of the context and played_by properties. Figure 7.9 shows the general structure of this pattern for observer relative and institutional facts. This approach suffers from the same restriction regarding the equivalence axiom. Generally, the attribution of social constructs is difficult to define independently. A possible way out is to give a more specific definition of the context of the social construct, e.g. the set of actions typically associated with students. Nonetheless, the use of a *marker property* such as is_student is consistent with the fact that social facts are created via conscious acts, such as statements and declarations (Searle, 1995).

$$
\begin{aligned}
\text{Subjective\_Entity} &\equiv \text{imposed\_on } \textbf{some } \text{owl:Thing} \sqcap \\
&\quad\ \ \text{context } \textbf{some } \text{owl:Thing} \\
&\sqsubseteq \text{imposed\_on } \textbf{exactly } 1 \text{ owl:Thing} \sqcap \\
&\quad\ \ \text{context } \textbf{exactly } 1 \text{ owl:Thing} \\
\text{Role} &\equiv \text{played\_by } \textbf{some } \text{owl:Thing} \\
&\sqsubseteq \text{Mental\_Entity} \\
\text{Social\_Role} &\sqsubseteq \text{Role} \sqcap \\
&\quad\ \ \text{played\_by } \textbf{some } \text{Agent} \\
\text{Student} &\equiv \text{played\_by } \textbf{some } \text{Person} \sqcap \\
&\quad\ \ \text{context } \textbf{some } \text{University} \\
&\sqsubseteq \text{Social\_Role} \\
&\sqsubseteq \text{is\_student } \textbf{some self}
\end{aligned}
$$

$$
\begin{aligned}
\text{imposed\_on} &\equiv \text{counts\_as}^{-} \\
\text{played\_by} &\equiv \text{plays}^{-} \\
\text{played\_by} &\sqsubseteq \text{imposed\_on} \\
\text{plays} &\sqsubseteq \text{counts\_as}
\end{aligned}
$$

$$
\begin{aligned}
\text{context}^{-} \circ \text{is\_student} \circ \text{played\_by} &\sqsubseteq \text{Student} \\
\text{plays} \circ \text{is\_student} \circ \text{context} &\sqsubseteq \text{University}
\end{aligned}
$$

Figure 7.10: Axiomatisation of the Student class

Step 5: Punning

To complete the picture of summarisation, the property and class names of the subjective entity being defined can be made *the same*. OWL 2 punning allows us to treat the named entity as either an object or a class depending on how it is used (see Section 3.5.2). This way, roles such as Student can be used both as property and as class without compromising ontological commitment, or burdening (re)users of the ontology with overly verbose vocabulary. The resulting axiomatisation is depicted in Figure 7.10.

7.3.4 Discussion and Examples

The preceding section gives an overview of a general structure design pattern for representing the summarisation of reified relations. This summarisation is established in five steps, that partially coincide with the steps in Section 7.2.1. Create *initial class definitions* that provide a basic means for identifying individuals. Constrain the *number and type of role fillers* by applying the techniques from Section 7.2.1. Define the summarisation property as a property chain that *traverses the tree* from context to brute fact. Introduce *domain dependence* by incorporating contextual information, most notably using a *marker property* that uniquely identifies members of a class in the RBox. Optionally use *punning* to lift the syntactic distinction between the summarisation property and the corresponding reification.

The following sections give an overview of how this pattern can be applied

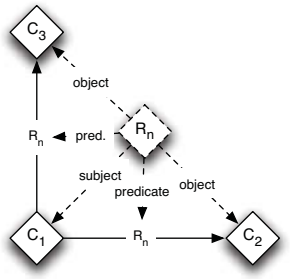

Figure 7.11: Reification R_n of an n-ary relation.

to various different contexts, such as the representation of n-ary relations of Section 7.3.1, and various mental entities in the LKIF Core ontology.

Representing N-Ary Relations

Although we have seen in Section 7.3.1 that n-ary relations cannot be represented directly in OWL DL, we can summarise them in a fashion similar to how we represent social roles. The main difference between the reification of binary relations and n-ary relations is that the latter have multiple subjects or objects, but share a single predicate. Compare the reified relation R_n of Figure 7.11 with that of R in Figure 7.7.

As Noy and Rector (2006) point out, n-ary relations typically have multiple objects, but a single subject (Cases 1 and 2 from Section 7.3.1). Consider their representation of the Diagnosis_Relation:

$$\begin{aligned} \mathsf{Diagnosis\_Relation} &\sqsubseteq \mathsf{diagnosis\_value}\ \textbf{some}\ \mathsf{Disease} \\ &\sqsubseteq \mathsf{diagnosis\_prob}\ \textbf{some}\ \mathsf{Probability} \\ \mathsf{Person} &\sqsubseteq \mathsf{has\_diagnosis}\ \textbf{only}\ \mathsf{Diagnosis\_Relation} \end{aligned}$$

Because diagnosis_value and diagnosis_probability are functional properties, no cardinality restrictions are defined. Figure 7.12 shows an instance of this relation DR as it is related to a person P, a disease D and its probability P. To make the representation more uniform, we add a restriction on the inverse of has_diagnosis as subclass axiom to Diagnosis_Relation (no diagnosis without a patient):

$$\mathsf{Diagnosis\_Relation} \sqsubseteq \mathsf{patient}\ \textbf{some}\ \mathsf{Person}$$

Following the pattern described in the previous section, we add a marker property is_diagnosis to the definition of Diagnosis_Relation:

$$\mathsf{Diagnosis\_Relation} \sqsubseteq \mathsf{is\_diagnosis}\ \textbf{some self}$$

Given these definitions, we can define the summarisation property diagnosis with domain Person and range Disease $\sqcup$ Probability as the following property inclusion axiom:

$$\begin{aligned} \mathsf{has\_diagnosis}\ \textbf{o}\ \mathsf{is\_diagnosis}\ \textbf{o}\ \mathsf{diagnosis\_value} &\sqsubseteq \mathsf{diagnosis} \\ \mathsf{has\_diagnosis}\ \textbf{o}\ \mathsf{is\_diagnosis}\ \textbf{o}\ \mathsf{diagnosis\_prob} &\sqsubseteq \mathsf{diagnosis} \end{aligned}$$

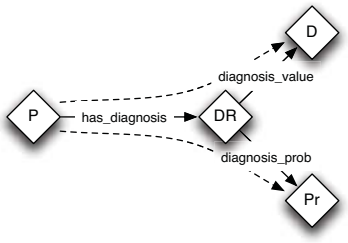

Figure 7.12: The n-ary relation DR between person P, disease D and probability Pr.

In this case, the marker property is_diagnosis may be a bit superfluous. However we can easily extend this pattern to express more specific kinds of diagnosis:

$$\begin{aligned} \text{Cancer\_Diagnosis} &\equiv \text{diagnosis\_value } \textbf{some } \text{Cancer} \\ &\sqsubseteq \text{is\_cancer\_diagnosis } \textbf{some self} \end{aligned}$$

and informative properties such as a has_cancer_prob property that relates the patient to the probability of such a particular diagnosis:

$$\text{has\_diagnosis } \textbf{o } \text{is\_cancer\_diagnosis } \textbf{o } \text{diagnosis\_prob} \sqsubseteq \text{has\_cancer\_prob}$$

These axioms allow us to infer for e.g. $Christine$ (P) that if she has a has_diagnosis relation with a Cancer_Diagnosis DR which has a diagnosis_prob with value $high$ (Pr), then $Christine$ has_cancer_prob $high$.

This representation can be extended in a straightforward manner to cover more complex n-ary relations, as e.g. case 3 of Section 7.3.1 where $John$ buys a $Lenny\_the\_Lion$ book. His Purchase is in fact a type of Transaction (Section 7.2.1). We can use the familiar role inclusion axiom and marker property is_purchase to express relations such as buys, sells and earns:[16]

$$\begin{aligned} \text{actor}_m^- \textbf{ o } \text{part}^- \textbf{ o } \text{is\_purchase } \textbf{o } \text{part } \textbf{o } \text{object}_g &\sqsubseteq \text{buys} \\ \text{actor}_g^- \textbf{ o } \text{part}^- \textbf{ o } \text{is\_purchase } \textbf{o } \text{part } \textbf{o } \text{object}_g &\sqsubseteq \text{sells} \\ \text{actor}_g^- \textbf{ o } \text{part}^- \textbf{ o } \text{is\_purchase } \textbf{o } \text{part } \textbf{o } \text{object}_m &\sqsubseteq \text{earns} \end{aligned}$$

Figure 7.13 shows how $John$ (Ag_1) can be inferred to have a buys relation with the $Lenny\_the\_Lion$ book (G).

Representing Mental Entities

Besides roles, other subjective and mental entities play an important role in the LKIF Core ontology. Propositional attitudes, such as beliefs, intentions and convictions are central to the representation (and resolution) of legal cases, and in particular *mens rea*. For instance, the state of mind 'malice aforethought'

[16] Provided that the objects of both Transfer actions are suitably distinguished.

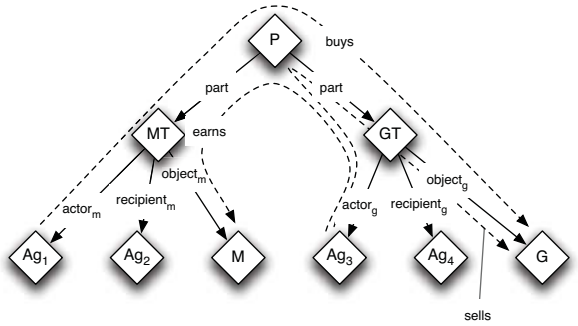

Figure 7.13: The sells, buys and earns relations in a Purchase transaction

necessary for the qualification of murder, is defined as the *intent* to kill, inflict serious bodily harm or a state of reckless indifference. Also, persons can be held liable for the actions of another person if they caused that person to hold a belief that led to an illicit act (Hart and Honoré, 1985).

Propositional attitudes are not only subjective entities – they are observer relative – but have a relational character that is very similar to that of roles. For instance, consider the following sentence:

"Mary believes that John killed Susan."

Here the propositional *content* of the belief is "John killed Susan". The pronoun 'that' is used to mark a reification of the propositional content of the belief. However, we cannot say that "John killed Susan" *is* the belief, rather the belief is inclusive of both the attitude and the content, i.e. it is "*that* John killed Suzan". We can rephrase the example as "Mary *holds* a belief *towards* the proposition 'John killed Suzan'.", as a consequence of which "Mary" *believes* "John killed Suzan".

The LKIF Core takes this duality into account and represents the classes Propositional_Attitude and Proposition as follows:

$$
\begin{aligned}
\text{Propositional\_Attitude} &\equiv \text{towards } \textbf{some } \text{Proposition} \\
&\sqsubseteq \text{Subjective\_Entity} \sqcap \text{Mental\_Object} \sqcap \\
&\quad \text{held\_by } \textbf{some } \text{Agent} \\
\text{Proposition} &\sqsubseteq \text{Mental\_Object} \sqcap \\
&\quad \text{attitude } \textbf{some } \text{Propositional\_Attitude}
\end{aligned}
$$

$$
\begin{aligned}
\text{towards} &\sqsubseteq \text{imposed\_on} \\
\text{attitude} &\sqsubseteq \text{counts\_as} \\
\text{attitude} &\equiv \text{towards}^{-} \\
\text{holds} &\sqsubseteq \text{context}^{-} \\
\text{held\_by} &\equiv \text{holds}^{-}
\end{aligned}
$$

A propositional attitude is anything held by an Agent towards some Proposition. The properties towards and holds are sub properties of the properties we used to

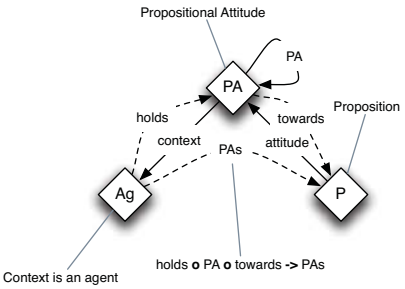

Figure 7.14: The propositional attitude PA held by an agent Ag towards a proposition P.

construct subjective entities. This definition is easily refined to cover the belief example, by adding a subclass Belief, its corresponding marker property, and the summarisation property chain:

$$\begin{aligned} \mathsf{Belief} &\sqsubseteq \mathsf{is\_belief}\ \textbf{some self} \\ &\sqsubseteq \mathsf{Propositional\_Attitude} \end{aligned}$$

$$\mathsf{holds}\ \textbf{o}\ \mathsf{is\_belief}\ \textbf{o}\ \mathsf{towards} \sqsubseteq \mathsf{believes}$$

Using this definition, we can infer for any agent Ag that holds a Belief (PA) towards a Proposition (P), that Ag believes P. The general pattern is depicted in Figure 7.14).

Some attitudes are not merely internal to some agent, but are externalised by means of a speech act. The Speech_Act class in LKIF Core is defined as an action that creates a Communicated_Attitude such as declarations, assertions and promises. Communicated attitudes are those attitudes that convey an Expression, a proposition that is mediated through some Medium (e.g. a book):

$$\begin{aligned} \mathsf{Communicated\_Attitude} &\equiv \mathsf{states}\ \textbf{some}\ \mathsf{Expression} \\ &\sqsubseteq \mathsf{Propositional\_Attitude} \\ \mathsf{Speech\_Act} &\equiv \mathsf{creates}\ \textbf{some}\ \mathsf{Communicated\_Attitude} \\ &\sqsubseteq \mathsf{Creation} \end{aligned}$$

$$\begin{aligned} \mathsf{states} &\sqsubseteq \mathsf{imposed\_on} \\ \mathsf{creates} &\sqsubseteq \mathsf{object} \end{aligned}$$

As Creation is a subclass of Action, the Speech_Act inherits its restriction on the participants actor and result. An Agent is said to utter a Communicated_Attitude if it is created through some Speech_Act:

$$\mathsf{Speech\_Act} \sqsubseteq \mathsf{is\_speech\_act}\ \textbf{some self}$$

$$\mathsf{actor}^{-}\ \textbf{o}\ \mathsf{is\_speech\_act}\ \textbf{o}\ \mathsf{creates} \sqsubseteq \mathsf{utters}$$

The attentive reader may have noticed that the utters property is not defined as a sub property of the inverse of context. The reason is that if this relation were

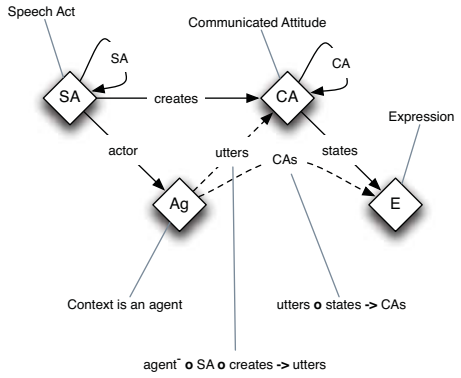

Figure 7.15: The communicated attitude CA that states expression E, created by speech act SA and uttered by agent Ag

asserted, the context property would become a *complex* property, and we would lose the ability to constrain its cardinality (see Section 3.5.1). We can circumvent this limitation by *wrapping* the context property into the definition of utters. This done by additionally defining the utters relation as a super property of the concatenation of context$^-$ and a marker property is_communicated_attitude:

$$\text{context}^- \circ \text{is\_communicated\_attitude} \sqsubseteq \text{utters}$$

This way, we can infer the utters relation in two ways: via the specification of a speech act, or by the explicit assertion of a communicated attitude. Unfortunately, the tree model property of DL again does not allow us to enforce that simultaneous application of both methods results in the inference that the speech acts are uttered by a *single* Agent. The same_id_as relation of Section 7.2.1 can be extended to simulate an owl:sameAs relation for this pattern:

$$\text{actor}^- \circ \text{is\_speech\_act} \circ \text{creates} \circ \text{utters}^- \sqsubseteq \text{same\_id\_as}$$

The new vocabulary can be used to express the relational character of various communicated attitudes, such as declarations and statements. For instance, the Declaration class is defined as follows:

$$\begin{aligned} \text{Declaration} &\sqsubseteq \text{is\_declaration } \textbf{some self} \\ &\sqsubseteq \text{Communicated\_Attitude} \end{aligned}$$

The marker property is_declaration is then used to define the declares relation (Figure 7.15):

$$\text{utters} \circ \text{is\_declaration} \circ \text{states} \sqsubseteq \text{declares}$$

Implications for Reuse

In the discussion of Chapter 6 it was hinted at that summarisation could contribute to a more practical reusability of ontologies. The broad range of use

cases for the summarisation pattern discussed in this section clearly shows that its applicability is far reaching. Without summarisation, the choice between a *relation* or *class* oriented representation of e.g. roles is an important ontological commitment. And more importantly, this choice would have to be made individually, for any of the use cases we discussed. The result is often a hodgepodge of relation and class oriented solutions, within a single ontology.

Reuse of the LKIF Core ontology in applications that use expressive features of OWL 2, such as the versioning mechanism of Klarman et al. (2008) and normative assessment described in Hoekstra et al. (2008), is quite demanding because of the large number of complex class axioms in the LKIF ontology itself (see Section 6.4). Summarisation allows us to combine the conciseness of relations with the verbose ontological correctness of classes within an ontology. A lightweight version of the ontology can be constructed by creating direct class axioms for summary properties, while discarding the axioms related to their reification. The resulting ontology is much more succinct, both qua ontological commitment and qua expressiveness. While summary properties are complex in the original ontology, the lightweight ontology represents them as simple. The latter consequently does not involve any cardinality constraints nor does it assert expressive property types over these properties. Of course ontologies that reuse the lightweight ontology may add such restrictions on those properties without violating any of the global restrictions in Motik et al. (2009), but this would not constitute safe reuse.

7.4 Sequences: *Change and Causation*

> "You see there is only one constant. One universal. It is the only real truth. Causality. Action, reaction. Cause and effect. "
> "And this is the nature of the universe. We struggle against it, we fight to deny it but it is of course. Pretend it is a lie, beneath our poised appearance the truth is we are completely out of control. Causality, there is no escape from it, we are forever slaves to it. Our only hope, our only peace is to understand it, to understand the why. "
>
> *Merovingian, The Matrix Reloaded*

The representation of sequences of entities is central to many domains. Examples are the representation of physically or conceptually linked structures such as chains, trains, roads, and routes, or chapters in a book, respectively (Drummond et al., 2006). For instance, Noy and Rector (2006) describe the a case for an n-ary relation that involves a route: "United Airlines flight 3177 visits the following airports: LAX, DFW, and JFK." The flight has a relation with three airports, and the sequence indicates the order in which its visits these airports. Similarly, sequences play an important role in the qualitative representation of quantities, such as relative speed or temperature. However, it is particularly essential in the representation of anything related to *time* and *change*, for clearly, temporal relations have a directed sequential structure:

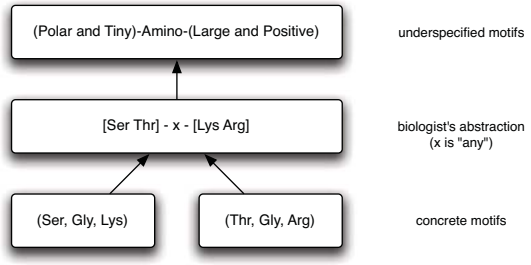

Figure 7.16: Alternative classifications of under specified protein sequence motifs (Drummond et al., 2006, Fig.2)

> Time is the conscious experiential product of the processes that allow the (human) organism to adaptively organise itself so that its behaviour remains tuned to the sequential (i.e. order) relations in its environment.
>
> (Michon, 1990, p.40)

A perceived temporal ordering of events is thus a by-product of our interaction with the order relations in reality. In Breuker and Hoekstra (2004b); Hoekstra and Breuker (2007) we have argued that this order is dependent on how processes in reality *causally* interact. The fact that the occurrence of some process is a necessary and sufficient condition for the occurrence of another process, implies a causal relation that coincides with a temporal ordering of the processes: causation is what makes time tick.

Other than the transitivity of properties, OWL does not provide a built-in construct for representing (temporal) sequences. Though OWL 2 will include an owl:datetime datatype property for representing times, a similar datatype-oriented approach is not always applicable. For instance, Drummond et al. (2006); Villanueva-Rosales and Dumontier (2007) focus on the representation of sequences of molecular structures such as protein sequences and functional groups, respectively. Proteins are sequences of amino acids, and are categorised and referred to according to properties of the patterns of amino acids – or *motifs* – they contain. Drummond et al. present a design pattern for sequences inspired by the data structure commonly used to represent *lists* (as e.g. in rdf:List, Prolog and Lisp list, and the abstract List class in Java). Their OWL-List class functions as a built-in construct, and can be used to classify proteins according to the patterns they contain (see Section 7.4.3 for a discussion of its definition). Figure 7.16 shows an example classification task in biochemistry.

The OWL representation of Villanueva-Rosales and Dumontier has a similar goal. The chemical properties of large carbon based molecules (chemical compounds) are described by experts in terms of the interaction between functional groups: partial molecular structures. Although we could envision a mapping from functional groups and motifs to unique numbers (or even a custom datatype), this moves all semantics outside of OWL itself and we can no longer adequately represent the groups themselves. Arguably, a representation of time in terms of points or intervals has a more straightforward mapping to a datatype, but again, such an approach does not define these temporal occurrences themselves.

The following sections introduce a design pattern for representing sequences, based on a use case for causal explanation. The next section introduces the perspective on causation of Hoekstra and Breuker (2007) and illustrates requirements for its representation as part of LKIF Core. Section 7.4.3 compares the pattern with that of Drummond et al. and discusses its advantages and limitations.

7.4.1 Causation

Causality plays a central role in virtually all scientific disciplines. In all cases its use is more precise than in commonsense – for instance, it is distinguished from covariance, correlation and coincidence – but essentially it does not differ in its capacity to explain dependencies between events. Lehmann (2003) adopts a distinction between the notions of causality and *causation*; he uses the term 'causality' to denote the ontological view, and reserves the term 'causation' for the *occurrence* of causality. The definition of causality is a major issue in philosophical metaphysics, and has been for many centuries (Kim, 1998; Davidson, 2001; Schaffer, 2003). Philosophy primarily concerns itself with what causality *is*, i.e. it is focused on ontological questions regarding the existence and properties of causality. Epistemological questions arise concerning how we can know about the occurrence of causality in the real world and, once we do know, what inferences we can draw from the causal relation between two events.

Causation is an abstract, reflective concept that summarises a more qualitatively distinct relation. An account of causation involves an understanding of relationships between events in terms of *processes*. This intermediary role of processes can be conceived of as a dependency: as causation between events. Causation is no more and no less than the (abductive) inference that the occurrence of event B can be explained by a (sub-)process that is implied by an earlier or simultaneous event A. In other words, after we perceive a collision between two billiard balls, we infer a transfer of force from the moving ball to the static ball. This allows us to say that the collision caused the second ball to move. In a nutshell, recognising the transfer of power at the collision is sufficient for our understanding; the assertion that the collision caused the ball to move is superfluous.

This view furthermore exposes the *overloading* of causal relations common in many domains, such as the interspersion of liability and causality in legal causation (Hoekstra and Breuker, 2007). In general, we can distinguish two types of causation: *physical causation* that describes physical processes and *agent causation*, which describes the actions of rational agents. Legal theory introduces two additional forms of causation (Lehmann, 2003; Hart and Honoré, 1985): *interpersonal causation*, which describes the effect one agent might have on another (e.g. as a consequence of communication), and *negative causation*, representing the connection between an effect and some agent *not* acting (e.g. negligence). As discussed in Hoekstra and Breuker (2007), the latter are legal constructs rather than basic causal relations.

Approaches

In Artificial Intelligence, causation has been the direct and indirect object of study in a number of specialised areas. Worth mentioning in this respect is the work of Pearl (2000), which finds its basis in a *probabilistic* representation of physical causation; more knowledge-intensive approaches to *physical causation* from the fields of Qualitative Reasoning (QR) and Model Based Reasoning (MBR) (e.g. Horn (1990)), *ontology-based* approaches, and a multitude of formalisms for *agent causation* (communication) within multi-agent systems.

Formal Causal Reasoning The work by Pearl (2000) stands out in AI as the most comprehensive and explicit modelling of what could be called "computational causality". His main contribution lies with probabilistic (Bayesian) modelling of quantifiable dependencies between occurrences, e.g. the use of drugs and their effects on patients. In this sense it fits the tradition in science to enable the identification of causes or causal factors in a well founded, formal way. This work is also relevant for legal reasoning about cases that have a probabilistic basis, as for instance in claims about compensation for damage of health suffered due to the consumption of certain industrial products. However, his approach is not particularly well suited for a more qualitative rather than quantitative conception of causation, which Pearl (2000, Ch.10) coined *actual cause*:

> "an event recognised as responsible for the production of a certain outcome [...] Human intuition is extremely keen in detecting and ascertaining this type of causation and hence is considered the key to construct explanations [...] and the ultimate criterion (known as "cause in fact") for determining legal responsibility"
> [...]
> "Clearly, actual causation requires information beyond that of necessity and sufficiency: the actual process mediating between the cause and the effect must enter into consideration."
>
> Pearl (2000, p.309)

However, Pearl sees the conceptual basis for actual causation as a mere additional element, rather than primary and sufficient for deciding on causation in fact. His somewhat exploratory work on factual causation is aimed at a formal, 'correct' modelling of causal dependency rather than a more qualitative commonsense perspective.

Qualitative and Model Based Reasoning Another potential source of inspiration for the representation of causal relations is the representation and reasoning involved in modelling the structure and behaviour of systems (Bredeweg and Struss, 2004). Qualitative reasoning (QR) started as an approach to commonsense reasoning and initially coined *naive physics*. In his influential second naive physics manifesto, Hayes (1985) argues that causality is not a "useful, self-contained theory", but "that it is an umbrella term for a large variety of particular relationships" (Hayes, 1985, p.19). In Hoekstra and Breuker (2007) we underwrite his stance that causality is not a primary term in an ontology

of common sense, e.g. it is not defined by LKIF Core. Rather, the variety of causal relationships between events considered to correspond to the relations between different kinds of processes.

In QR, a model of the structure of a system is used to simulate the propagation of changes through the system, including structural changes of the system itself. This propagation yields chains of events, which represent a prediction of the behaviour of the system, given some initial state. Events in QR are connected by property values induced by two types of relations: *influences* and *proportionalities*. Proportionalities represent definitional or inherent dependencies between property values. For instance, an increase in volume of a substance has a linear correspondence to its weight.[17] On the other hand, influences represent an actual *causal* effect on the value of properties, for instance the proximity of a heat source will influence the temperature of an object. These causal relations follow from processes (Forbus, 1984), which are represented as *model fragments*.

QR is particularly useful for modelling well known and stable physical structures. For instance, model based reasoning – an applied branch of QR – is used mainly in the representation and diagnosis of devices. Although QR techniques have been used in other domains than physics, e.g. in modelling ecological and social systems (Bredeweg et al., 2006), it is limited to purely *physical causation* and does not provide a means to model agents, let alone the processes initiated by them. Furthermore, as discussed in Hoekstra et al. (2006), the model fragments that play a central role in QR cannot be represented in OWL in a straightforward manner, as they are highly structured concepts (Motik et al., 2007a, and Section 7.1).

Agent Technology Agent causation is not explained just by causes but by reasons as well, and in particular the *intention* to perform an action. The notion of intention, although more recent, has given rise to as much philosophical controversy as causation. In AI the notion has been operationalised to model actions, and in particular communication between artificial agents. The languages by which these agents communicate typically provide constructs for representing belief-states, actions, plans, data etc. The most prominent example being the belief, desire, intention model of Georgeff et al. (1999, BDI). The BDI model is a representation of the inner workings of agents (including people), and is intended to contribute to a better understanding of how intention influences action. Nonetheless, there are a number of reasons why agent technology cannot be used for detecting commonsense causation. First of all, agent technology is concerned with *simulation*, the realtime behaviour of agents, where our use case relies on post-diction. Furthermore, the BDI model does not cover physical causation.

Ontology The ontological approach to causation in fact described in Lehmann et al. (2004); Lehmann and Gangemi (2007), defines causal dependencies within the framework of the DOLCE ontology, see e.g. Gangemi et al. (2002, and Section 6.2.1).

Causal relations relate very simple events that change a single aspect of a single object. Between these events, three existential dependencies are identi-

[17] Provided that the substance is not a contained gas.

fied: *structural*, *causality* and *circumstantial* dependencies. Physical causation is defined as the relation that holds between two individual events that satisfy both the causality and the circumstantial constraints. The framework allows for the classification of some description of a number of events as an instance of *physical causation in fact*.

The ontology contains no theory of the (physical) world; i.e. descriptions are not expressed using domain knowledge. For instance, the semantics of a state is not represented intensionally, but only through its name (e.g. *being-wounded*), it contains no description of *what it means* to be in that state. The lack of such descriptions makes this approach less useful for explaining the existence of a causal relation: the basic causal relations between events are *asserted* rather than inferred on the basis of domain knowledge. A system built on such a representation is less flexible in dealing with incomplete knowledge, as it can only infer causal propagation on the basis of a fully specified, explicit causal model.

7.4.2 Representing Causal Change

This section introduces a content pattern for describing the linear ordering of events that lies at the heart of causal relations in LKIF Core. The approach differs from Drummond et al. (2006) in that it captures the sequential nature of these relations, similar to how transitivity relates to the part of relation, whereas Drummond et al. use a nested structure. The pattern is based on the principle that an account of subsequent states of objects, expressed in terms of domain knowledge, can be recognised as the occurrence of one or more processes. The interaction between these processes is both necessary and sufficient for the identification of causal relationships (Hoekstra and Breuker, 2007). Given the sequential ordering of these states and processes we can furthermore infer temporal relations.

Step 1: Initial Class Definition

Events and states occur at some time and place, i.e. they happen against a four dimensional canvas of space and time (Davidson, 2001). These co-ordinates determine the possibility of causal relationships: time and location limit causal propagation. The intensional definition of Change expresses a difference between before and after its instances occur, and thus involves a requirement and a result situation. A particular situation may be any valid configuration of entities. Since these entities have to be *related* in order for them to form a single situation, it can be generically captured by an OWL class description that takes one of the entities participating in the situation as focal point. In other words, where the definition of a change at class level describes required and resulting *situations*, a particular occurrence of a change, i.e. an *event*, is related to single individuals.[18] Processes are changes that consist of other changes, though not every change that is composed of other changes is a process.

[18]States and events are *occurrences* and are used to respectively refer to individual objects and changes.

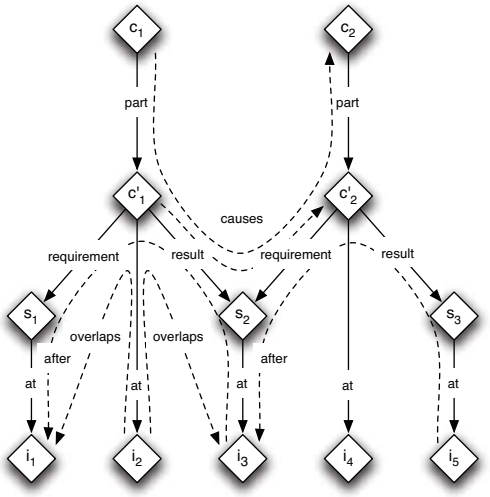

Figure 7.17: Change and causal relations.

The initial class definitions of Change and Process are thus as follows:

$$\begin{aligned}
\text{Change} \quad &\equiv \quad \text{requirement } \textbf{some } \text{owl:Thing} \sqcap \text{result } \textbf{some } \text{owl:Thing} \\
&\sqsubseteq \quad \text{requirement } \textbf{exactly } 1 \sqcap \text{result } \textbf{exactly } 1 \\
&\sqsubseteq \quad \text{at } \textbf{some } \text{Place} \sqcap \text{at } \textbf{some } \text{Temporal_Occurrence} \\
&\sqsubseteq \quad \text{part } \textbf{only } \text{Change} \\
\\
\text{Process} \quad &\sqsubseteq \quad \text{Change}
\end{aligned}$$

Step 2: Traverse the Tree

The above definition already allows us to define a number of interesting causal and temporal relations. For instance, consider the situation in Figure 7.17. The changes c_1 and c_2 each have a part c_1' and c_2' where the result o_2 of c_1' is a requirement for c_2'. Here, clearly c_1' causes c_2', but c_1 can be said to cause c_2 as well. These causal relations are straightforwardly defined using two role inclusion axioms:

$$\begin{aligned}
\text{result } \textbf{o } \text{requirement}^- \quad &\sqsubseteq \quad \text{causes_directly} \\
\text{causes_directly} \quad &\sqsubseteq \quad \text{causes} \\
\text{part } \textbf{o } \text{causes } \textbf{o } \text{part}^- \quad &\sqsubseteq \quad \text{causes}
\end{aligned}$$

where causes is a transitive property and causes_directly is not. We can furthermore remark that if two situations are the requirement and result of a change, then a temporal ordering relation must hold between the intervals at which they hold:

$$\begin{aligned}
\text{at}^- \textbf{ o } \text{result}^- \textbf{ o } \text{requirement } \textbf{o } \text{at} \quad &\sqsubseteq \quad \text{after} \\
\text{at}^- \textbf{ o } \text{requirement}^- \textbf{ o } \text{result } \textbf{o } \text{at} \quad &\sqsubseteq \quad \text{before}
\end{aligned}$$

These inclusion axioms allow us to infer e.g. that the interval i_1 holds before i_3, and i_3 before i_5. However, for a change to affect its conditional situations, we may require its temporal occurrence to coincide with an interval i_2 that partially overlaps the time of its required and resulting situation:

$$\begin{aligned} \mathsf{at}^- \circ \mathsf{requirement} \circ \mathsf{at} &\sqsubseteq \mathsf{overlaps} \\ \mathsf{at}^- \circ \mathsf{result} \circ \mathsf{at} &\sqsubseteq \mathsf{overlaps} \end{aligned}$$

A more rigorous adoption of Allen (1984) might specify that i_2 *finishes* the interval i_1 and *starts* interval i_3. However, this would mean that i_2 coincides with an overlap between i_1 and i_2, which is a significant ontological commitment to the simultaneous coexistence of the situations s_1 and s_2. To exclude this possibility, the requirement and result properties are made disjoint. This also removes models where a change is inferred to directly cause itself.

We can now also specify that a Process (or any other entity) occurs during an interval that covers, i.e. encompasses, all intervals of all changes it is composed of:

$$\mathsf{at}^- \circ \mathsf{part} \circ \mathsf{at} \sqsubseteq \mathsf{covers}$$

Corresponding spatial relations, such as overlaps can be constructed in a similar fashion.

Step 3: Disambiguate Role Fillers

The causes property allows us to define Process as the class of causal changes:

$$\begin{aligned} \mathsf{Process} &\sqsubseteq \mathsf{Change} \sqcap \mathsf{causes}\ \mathbf{some}\ \mathsf{owl{:}Thing} \\ &\sqsubseteq \mathsf{part}\ \mathbf{only}\ (\mathsf{Change} \sqcap \mathsf{causes}\ \mathbf{some}\ \mathsf{owl{:}Thing}) \end{aligned}$$

However, this definition leaves room for changes that are causal without being (part of) a process. In fact, as argued in Hoekstra and Breuker (2007) causation does not exist independently from processes. Whether some change is causal is determined by whether it is either a process or part of a process, and not the other way around. The causes relation should therefore only hold between entities that meet these criteria. We alter the definition of Process and causes by including a marker property is_process (see Section 7.3.3):

$$\mathsf{Process} \sqsubseteq \mathsf{is\_process}\ \mathbf{some}\ \mathbf{self}$$

$$\mathsf{is\_process} \circ \mathsf{part} \circ \mathsf{causes} \circ \mathsf{part}^- \circ \mathsf{is\_process} \sqsubseteq \mathsf{causes}$$

The above property chain infers a causes relation between two processes, but it does not specify a causal relation between the changes that are part of these process. To do this, we introduce the class Causal_Change and a marker property is_causal_change that captures those changes that are part of processes. The definition of Process is altered accordingly:

$$\mathsf{part\_of} \equiv \mathsf{part}^-$$

$$\begin{aligned}
\text{Causal_Change} &\equiv \text{part_of } \textbf{some } \text{Process} \\
&\sqsubseteq \text{is_causal_change } \textbf{some self} \\
&\sqsubseteq \text{Change} \\
\text{Process} &\sqsubseteq \text{is_process } \textbf{some self} \\
&\sqsubseteq \text{Change} \sqcap \text{causes } \textbf{some } \text{owl:Thing} \\
&\sqsubseteq \text{part } \textbf{only } \text{Causal_Change}
\end{aligned}$$

Anything that is part of some process, is a causal change, and processes consist only of causal changes. As a side note, we can now define Cause as an Epistemic_Role played only by causal changes:

$$\text{Cause} \equiv \text{Epistemic\_Role} \sqcap \text{played\_by } \textbf{some } \text{Causal\_Change}$$

To define the causal relation between causal changes, we replace the property chain for causes relations between changes (the second role chain in Step 2) with the following role inclusion:

$$\text{is\_causal\_change } \textbf{o } \text{causes\_directly } \textbf{o } \text{is\_causal\_change} \sqsubseteq \text{causes}$$

The inverse of the causes property is defined in a straightforward manner. The transitivity of the causes relation allows us to infer causal propagation both over a chain of connected result and requirement_of properties, and between processes that are connected via such a chain. Furthermore, the current definition ensures that causal relations only hold between changes and processes at the same aggregation level. A single change cannot cause a process, but rather the process (as a whole) that contains that change does.

Step 4: Introduce Asymmetry

The representation presented in the previous step allows us to infer causal relations between changes and processes provided that we *know* some change constitutes a process. However, as we have also said, the recognition of a process constitutes a causal *explanation*; the causal relations follow from this recognition. We therefore need some way to qualitatively distinguish different types of processes.

To illustrate this, consider a very simple blocks world that consists of a single Block (in three states b_1, b_2 and b_3), and two Surface individuals (s_1 and s_2). Any Object can be positioned on a single other individual of type Object, and the only operation available is to move an object to a different position. Our Move process will consist of lifting the object from s_1 and putting it on s_2.[19]

[19]Of course this is a simplification over the complexity of even the simplest processes in reality, such as the interplay between mass, speed, forces, and potential and kinetic energy in a collision between billiard balls. However, incorporating these properties does not alter the structure presented here.

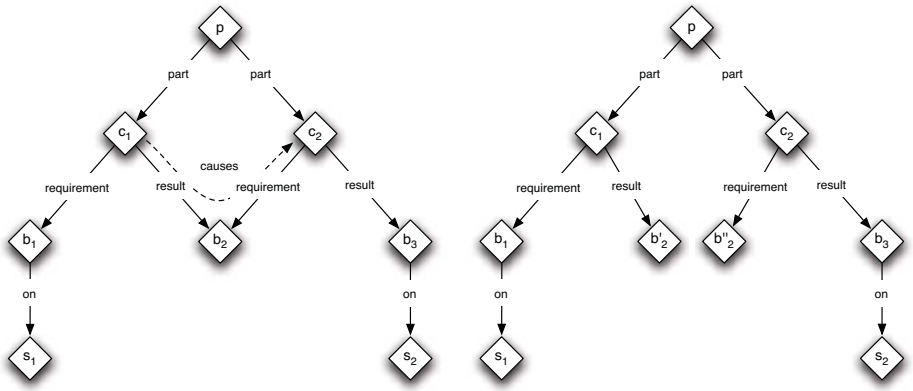

Figure 7.18: Two valid models of the class Move.

The corresponding class descriptions and individual assertions are as follows:

$$
\begin{aligned}
\text{Move} &\equiv \text{part } \textbf{some } \text{Lift} \sqcap \text{part } \textbf{some } \text{Drop} \\
&\sqsubseteq \text{Process} \\
\text{Lift} &\equiv \text{requirement } \textbf{some } (\text{Object} \sqcap \text{on } \textbf{some } \text{Object}) \\
&\quad \sqcap \text{result } \textbf{some } (\text{Object} \sqcap \textbf{not}(\text{on } \textbf{some } \text{Object})) \\
\text{Drop} &\equiv \text{requirement } \textbf{some } (\text{Object} \sqcap \textbf{not}(\text{on } \textbf{some } \text{Object})) \\
&\quad \sqcap \text{result } \textbf{some } (\text{Object} \sqcap \text{on } \textbf{some } \text{Object}) \\
\text{Object} &\equiv \{b_1, b_2, b_3, s_1, s_2\} \\
&\sqsubseteq \text{on } \textbf{max } 1 \\
\text{Block} &\sqsubseteq \text{Object} \\
\text{Surface} &\sqsubseteq \text{Object}
\end{aligned}
$$

Note that for the definition of Lift and Drop we need to close the world by enumerating all possible members of the Object class. This allows us to specify that b_2 is not on *any* other object using negative property assertions. Without this, the class restriction **not**(on **some** Object) would never be satisfied. The process p consisting of changes c_1 and c_2 on the blocks is defined as in Figure 7.18. Because b_2 is both the result of c_1 and the requirement for c_2, a causal relation is inferred between these two changes. However, the definition of Move has a valid model where the two changes are not connected in this way (Figure 7.18), and no causal relation is inferred.

As it is exactly the sequential character of changes that defines processes, we make the recognition of the process entirely dependent on a property chain that expresses its characteristic causal sequence of changes. This chain is (again) defined using marker properties that allow us to recognise classes inside the

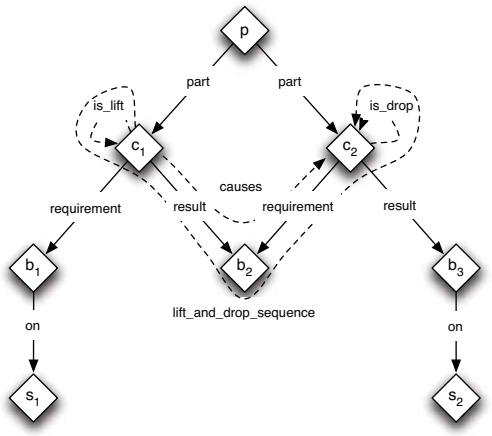

Figure 7.19: Example of a process sequence.

RBox:

$$
\begin{aligned}
\mathsf{Lift} &\sqsubseteq \mathsf{is\_lift}\ \textbf{some self}\\
\mathsf{Drop} &\sqsubseteq \mathsf{is\_drop}\ \textbf{some self}\\
\mathsf{Move} &\equiv \mathsf{part}\ \textbf{some}\ (\mathsf{lift\_and\_drop\_sequence}\ \textbf{some}\ \mathsf{owl{:}Thing})\\
&\sqsubseteq \mathsf{part}\ \textbf{some}\ \mathsf{Lift}\\
&\sqsubseteq \mathsf{part}\ \textbf{some}\ \mathsf{Drop}\\
&\sqsubseteq \mathsf{Process}
\end{aligned}
$$

$$\mathsf{is\_lift}\ \textbf{o}\ \mathsf{causes}\ \textbf{o}\ \mathsf{is\_drop} \sqsubseteq \mathsf{lift\_and\_drop\_sequence}$$

Of course, a process can be defined by any number of such sequences. However, an obvious limitation of the representation is that it does not enforce the changes in the process sequence to be a part of the process itself. Parthood can be inferred, but this requires the representation of a separate role chain for every step in the process sequence as traversing the next property would include all changes (indirectly) caused by the process.

Figure 7.19 shows how the lift_and_drop_sequence traverses the decomposition of process p. The marker properties allow us to distinguish between the different branches of the decomposition in the same way that we introduced asymmetry in Section 7.2.1.

7.4.3 Discussion

The previous section presents an approach to the representation of sequences based on four familiar steps. First an *initial class definition* gives the basic structure for the elements of the sequential structure. Secondly, a transitive property is defined by *traversing* the basic structure. Additionally, the connected nature of the structure allows us to infer more relations between its elements (e.g. the temporal relations between the occurrence of changes). The third step is to *disambiguate* role fillers by introducing marker properties that ensure the sequence

relation only holds between appropriate elements. Finally, *asymmetry* is used to expose the sequential character of the structure. We mark a starting element that defines the sequence as a composition of property chains.

Drummond et al. (2006) and Noy and Rector (2006) take a different approach, and represent sequences as *lists*. The OWLList class of Drummond et al. is defined as follows:

$$\begin{aligned} \mathsf{OWLList} &\sqsubseteq \mathsf{isFollowedBy}\ \textbf{only}\ \mathsf{OWLList} \\ \mathsf{EmptyList} &\equiv \mathsf{hasContents}\ \textbf{max}\ 0 \\ &\equiv \mathsf{OWLList} \sqcap \neg(\mathsf{isFollowedBy}\ \textbf{some}\ \mathsf{owl{:}Thing}) \\ \\ \mathsf{hasNext} &\sqsubseteq \mathsf{isFollowedBy} \end{aligned}$$

where hasContents and hasNext are functional properties, and isFollowedBy is transitive. In this definition the hasContents property has the same role as marker properties in the pattern presented of the previous section, where the hasNext property is a generalisation of the result and requirement properties, and the transitivity of isFollowedBy corresponds to the causes relation. The OWLList pattern is more verbose as for every member of the sequence, an additional OWLList individual needs to be created that links the contents to the next member. To see the difference, consider a (Ser, Gly, Lys) protein sequence from Figure 7.16 using the OWLList pattern:

$$\begin{aligned} \mathsf{Ser\text{-}Gly\text{-}Lys} \equiv\ & \mathsf{OWLList} \sqcap \mathsf{hasContents}\ \textbf{some}\ \mathsf{Ser} \sqcap \\ & \mathsf{hasNext}\ \textbf{some}\ (\mathsf{OWLList} \sqcap \mathsf{hasContents}\ \textbf{some}\ \mathsf{Gly} \sqcap \\ & \mathsf{hasNext}\ \textbf{some}\ (\mathsf{OWLList} \sqcap \mathsf{hasContents}\ \textbf{some}\ \mathsf{Lys} \sqcap \\ & \mathsf{EmptyList})) \end{aligned}$$

With some appropriate definition of the classes Ser, Gly and Lys. Using the role-based pattern, we get:[20]

$$\begin{aligned} \mathsf{Ser} &\sqsubseteq \mathsf{is\_ser}\ \textbf{some self} \\ \mathsf{Gly} &\sqsubseteq \mathsf{is\_gly}\ \textbf{some self} \\ \mathsf{Lys} &\sqsubseteq \mathsf{is\_lys}\ \textbf{some self} \\ \mathsf{Ser\text{-}Gly\text{-}Lys} &\equiv \mathsf{part}\ \textbf{some}\ (\mathsf{serglylys\_seq}\ \textbf{some}\ \mathsf{Lys}) \end{aligned}$$

$$\mathsf{is\_ser}\ \textbf{o}\ \mathsf{next}\ \textbf{o}\ \mathsf{is\_gly}\ \textbf{o}\ \mathsf{next}\ \textbf{o}\ \mathsf{is\_lys} \sqsubseteq \mathsf{serglylys\_seq}$$

Compared to Drummond et al. (2006) the approach presented here has several advantages. First of all, the pattern is much less verbose at both the class and individual level (see Figure 7.20). Furthermore, the pattern does not rely on the representation of a data structure: it feels conceptually odd to call Ser-Gly-Lys a *list*, where it is really a molecular structure composed of three substructures. This is because the OWLList reflects an epistemological, rather than an ontological perspective.

[20] For clarity, the two approaches use distinct property names. Similarly, marker properties are prefixed with 'is_' though more intuitive names are allowed through punning, e.g. Ser instead of is_ser.

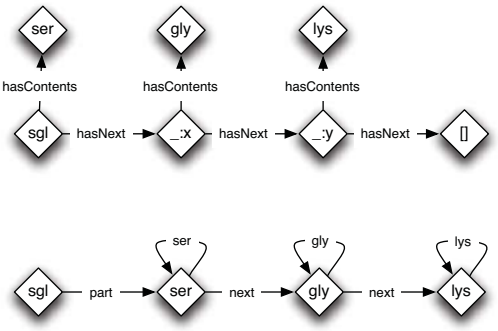

Figure 7.20: The (Ser,Gly,Lys) sequence represented as OWLList and as serglylys_seq.

One of the main problems with the OWLList class definition reported by Drummond et al. is that some of its valid models are not proper lists: it does not exclude the additional branches that can be defined using isFollowedBy instead of the functional hasNext property. Although the intended interpretation of the isFollowedBy property is that it is the transitive closure of hasNext, this cannot be enforced in OWL. The role-based approach suffers from the same problem, e.g. causes and serglylys_seq are super properties of their respective chains, but are not *equivalent* to it: an individual can be asserted to have a serglylys_seq property to some Lys individual without being connected through each consecutive step of the property chain (see Section 7.2.2). However, here the problem is less salient as the properties are specific to the class being defined (e.g. Ser-Gly-Lys), where isFollowedBy is domain independent and transitive. Similarly, a limitation shared by both approaches is that neither can exclude cycles, as this would require the complex sequence properties to be antisymmetric.

Where OWLList depends on the sequence to have no branches, and uses the functional hasNext property to (partially) ensure this, the role approach does allow for arbitrary branching, by enforcing or lifting local cardinality constraints on the next property (or corresponding alternative). For instance, the causal propagation defined in the previous section allows a state to be connected to multiple changes via the requirement property. Two process sequences that partially overlap can be superimposed by maintaining an exact cardinality restriction of 1 on the part property, while defining an existential restriction on both process sequence properties:

$$\begin{aligned} \mathsf{SomeProcess} \quad &\equiv \quad \mathsf{part}\ \textbf{some}\ (\mathsf{process\_sequence}_1\ \textbf{some}\ \mathsf{A} \sqcap \\ &\qquad\quad \mathsf{process\_sequence}_2\ \textbf{some}\ \mathsf{B}) \\ &\sqsubseteq \quad \mathsf{part}\ \textbf{exactly}\ 1 \end{aligned}$$

Alternatively, we can prohibit branching over the next property by defining next as functional, or specifying a maximum cardinality restriction on the property for each protein class.

Furthermore, sequences of a specific length are hard to define in either approach as no cardinality restrictions are allowed on transitive and other complex properties (See Section 3.5.1). The OWLList pattern requires an exhaustive

representation of each member class, while the role pattern needs the definition of a role inclusion axiom of the intended length. The advantage of the latter solution is that it can simply be stated in terms of the generic next property, whereas the former requires a rather complex class definition. The reason is that where role inclusion axioms are *sequential*, and can easily be concatenated, class descriptions are *nested* and do not allow direct reuse of generic structures.

Suppose the definition of simple sequence $S = [a, b, x, x, e]$, where x is an arbitrary element. The OWLList pattern for this sequence would be:

$$S = [a, [b, [x, [x, [e, []]]]]]$$

and it can readily be seen that the representation of the sequence $[x, x]$, namely $[x, [x, []]]$ cannot be directly inserted in the pattern. On the other hand, the role pattern representation of the sequence is:

$$S = a \circ b \circ x \circ x \circ c$$

where $x \circ x$ can be substituted with $Xs = x \circ x$, resulting in $S = a \circ b \circ Xs \circ c$. In a similar fashion, OWLList is limited in the representation of sub-lists, and requires the explicit representation of the entire list. The role approach does not have this limitation.

The procedure for causal explanation described in Breuker and Hoekstra (2004b); Hoekstra and Breuker (2007) was based on the stance of e.g. Michon (1990) that the experience of time and causation are cognitive by-products of our interaction with reality. The discovery of causal relations is therefore largely a matter of 'making sense' of changes in the world around us (cf. the cognitive perspective outlined in Section 6.2.2), the experience of time is a by-product of this process. Admittedly, the representation of processes and causal propagation presented here reflects a simplified view on causation. For instance, because of the limitations of OWL, it does not cover causal relations between a change where the *non* existence of a certain object is a requirement, and a change that ensures its *non* existence (i.e. a change that does not have a result).[21]

The system presented in Breuker and Hoekstra (2004b) relied on rules and a search over the knowledge base to 'probe' for possible processes in a bag of separate individuals. A match with a known process would indicate a causal and temporal ordering. The pattern introduced in the previous section can be used to achieve the same goal, but infers the types of processes occurring in the knowledge base by using an OWL 2 DL reasoner.[22]

[21] In fact, although it is possible to assert an individual that represents an object's non-existence, we cannot prevent through DL constructs that an 'existence' and 'non-existence' individual are asserted at the same time.

[22] Given the expressiveness of the pattern, this is bound to be computationally expensive for larger sets of states.

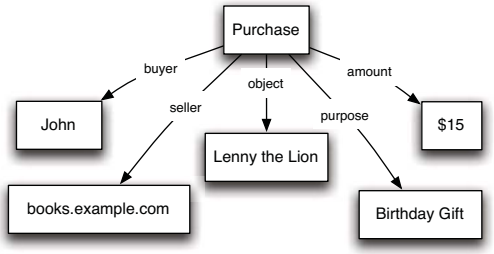

Figure 7.21: Transaction example from Noy and Rector (2006).

7.5 Patterns in Practice: *Action!*

The notion of *action* is a complex concept that brings together the perspectives of all three patterns defined in the preceding sections. The Transaction described in Section 7.2 involved two Transfer actions. The Purchase of Section 7.3 is such a transaction, and was conceived as a single reified n-ary relation. Its representation involved the inference of several relations between participants in the transaction. Furthermore, actions can be seen as the result of attributing an intention to some agent in starting a process (see also Section 6.3.2). In other words, an action is a Subjective_Entity that is imposed_on a Process. The connection between actions and processes is tight to such an extent, that the connection between an agent and the consequences of its actions is conceived of as *causal* (Davidson, 2001; Lehmann, 2003). In this light it is important to distinguish *reasons* from *causes*, i.e. "Sally hit Jimmy on the head *because* she was angry" gives an explanation of Sally's intention to hit Jimmy, and *not* of a causal relation between her anger and the hitting.

One might argue that since actions and processes *coincide*, action-processes cannot cease to be actions, the Action class must be rigid and should thus be defined as a subclass of Process. This would be possible if actions (and transactions) had the spider-like structure of Joe's Purchase in Noy and Rector (2006, and Section 7.3.1) (see Figure 7.21).[23] However, Breuker (1981, Ch.6) shows that the surface structure of linguistic expressions is a deceptively simplistic abstraction of a more complex conceptual structure. This conclusion is based on an analysis of mental processes that take place when people read a pronoun, such as 'it', and find its referent. The difference in response time for subjects in performing this reference assignment gives insight in the conceptual structure that exists in their mind.

Actions are often represented in terms of relationships with certain *thematic roles*, such as actor (or agent), object, instrument, location, recipient, patient, theme, etc. Over the years, many alternative representations of the relation between actions and thematic roles have been, and still are, investigated. Breuker's research corroborated the structure of conceptual dependency graphs, or *scripts*, of Schank and Abelson (1975), where characteristically the *instrument* of an action has a subordinate role. In particular, Breuker suggests that thematic roles exist at three distance levels from the action:

[23] Figure 7.13 in Section 7.3.4 already gave an alternate representation based on the Transaction pattern.

- The first level contains the *agent* and *object* roles, these are the most central to the action.
- At the second level, its *recipient* and *location* are specified. These roles can be used both in the sense of antecedent and consequent of the action, e.g. the from-location vs. the to-location.
- The *instrument* role is specified at the third level, and reflects a subordinate state or action necessary for the action to take place.

A second reason to conceive actions as subjective entities, rather than processes, is that action frames can range from very simple, as in "The butcher cuts the meat", to very complex actions such as "With a knife, the butcher cuts the meat into convenient pieces for the nice old lady." A representation of such sentences should adequately cover the structure and intentional perspective of the action expressed by it, without interfering with the structure and causal perspective of processes. Thirdly, the ability to conceive of actions without them actually taking place in reality, such as in planning, is an indication that they are mental entities. The role of verbs in the creation of metaphors (Pinker, 2007, and Section 5.6.1) furthermore suggests that the default relata of actions are *roles*. They are slots that have certain default fillers, but can be reapplied to non-standard categories to create a metaphor.

The following section shows how the three patterns are combined to create a consistent representation of the class Action.

7.5.1 Representing Action

As identified by Breuker (1981), the most basic participants of an Action are its actor and object. Only Agents can be the actor of an Action. Furthermore, Actions may be imposed_on a Process. We specify its basic class definition as follows:

$$\begin{aligned} \mathsf{Action} &\equiv \mathsf{actor}\ \mathbf{some}\ \mathsf{Agent} \sqcap \mathsf{object}\ \mathbf{some}\ \mathsf{owl{:}Thing} \\ &\sqsubseteq \mathsf{Subjective\_Entity} \sqcap \mathsf{actor}\ \mathbf{only}\ \mathsf{Agent} \sqcap \\ &\quad\ \mathsf{imposed\_on}\ \mathbf{max}\ 1\ \mathsf{Process} \\ \\ \mathsf{object} &\sqsubseteq \mathsf{participant} \\ \mathsf{actor} &\sqsubseteq \mathsf{participant} \end{aligned}$$

Note that this definition differs from the representation of Transfer in Section 7.2.1. It is less strict as it does not involve cardinality restrictions on the properties involved. Because we will *infer* the various roles of participants in the action using role inclusion axioms, we can no longer use number restrictions. In fact, as participants may reoccur at multiple stages of the process, e.g. the various states a piece of meat is in while being cut, there is a good non-technical reason for lifting the number restriction as well. However, it does mean that we need to specify additional conditions for the same_id_as property:

$$\begin{aligned} \mathsf{object}^- \circ \mathsf{object} &\sqsubseteq \mathsf{same\_id\_as} \\ \mathsf{actor}^- \circ \mathsf{actor} &\sqsubseteq \mathsf{same\_id\_as} \end{aligned}$$

We can still enforce the disjointness between participant properties as e.g. required to distinguish between recipient and actor. However, the disjointness of participants is not a general requirement, as agents may perform actions on themselves.

Because there is no way to automatically determine the participant roles of an action by its process description, they need to be explicitly *marked* as fulfilling a thematic role. In the most basic case, we need at least the Object_Role and Actor_Role. Their class definitions, including corresponding marker properties are as follows:

$$\begin{aligned} \mathsf{Thematic\_Role} &\sqsubseteq \mathsf{Role} \\ \mathsf{Object\_Role} &\sqsubseteq \mathsf{Thematic\_Role} \sqcap \mathsf{is\_object\_role}\ \mathbf{some\ self} \\ \mathsf{Actor\_Role} &\sqsubseteq \mathsf{Thematic\_Role} \sqcap \mathsf{is\_actor\_role}\ \mathbf{some\ self} \end{aligned}$$

Where is_object_role and is_actor_role are sub properties of the is_role marker property. Given a imposed_on assertion between an action a and a process p, and roles attributed to the situations that are part of p's causal structure, we can infer the individuals that *participate* in the action:

$$\begin{aligned} \mathsf{imposed\_on}\ \mathbf{o}\ \mathsf{is\_process}\ \mathbf{o}\ \mathsf{part}\ \mathbf{o}\ \mathsf{is\_change}\ \mathbf{o}\ \mathsf{requirement} &\sqsubseteq \mathsf{participant} \\ \mathsf{imposed\_on}\ \mathbf{o}\ \mathsf{is\_process}\ \mathbf{o}\ \mathsf{part}\ \mathbf{o}\ \mathsf{is\_change}\ \mathbf{o}\ \mathsf{result} &\sqsubseteq \mathsf{participant} \end{aligned}$$

The fillers of the actor and object properties can only be gathered by taking into account the roles they play. To accomplish this, the thematic roles need to 'backfire' to their role fillers:[24]

$$\begin{aligned} \mathsf{Object\_Role} &\sqsubseteq \mathsf{played\_by}\ \mathbf{only}\ (\mathsf{is\_object}\ \mathbf{some\ self}) \\ \mathsf{Actor\_Role} &\sqsubseteq \mathsf{played\_by}\ \mathbf{only}\ (\mathsf{is\_actor}\ \mathbf{some\ self}) \end{aligned}$$

Every individual that plays a particular thematic role is now related to itself via a marker property that indicates this role playing. The actor and object participants can now be gathered by the following property inclusion axiom:

$$\begin{aligned} \mathsf{participant}\ \mathbf{o}\ \mathsf{is\_object} &\sqsubseteq \mathsf{object} \\ \mathsf{participant}\ \mathbf{o}\ \mathsf{is\_actor} &\sqsubseteq \mathsf{actor} \end{aligned}$$

By our earlier definition of Subjective_Entity (of which Action and Role are subclasses), the subjective entity only holds within a certain context. Unfortunately, as with the utters relation in the definition of Communicated_Attitude in Section 7.3.4, we cannot specify this context via a property chain (since every subjective entity has exactly one context). However, we can again partially circumvent this problem by providing two alternate representations of a specific action_context property:

$$\mathsf{Action} \sqsubseteq \mathsf{is\_action}\ \mathbf{some\ self}$$

$$\begin{aligned} \mathsf{participant}\ \mathbf{o}\ \mathsf{plays} &\sqsubseteq \mathsf{action\_context}^- \\ \mathsf{is\_action}\ \mathbf{o}\ \mathsf{context}^-\ \mathbf{o}\ \mathsf{is\_object\_role} &\sqsubseteq \mathsf{action\_context}^- \\ \mathsf{is\_action}\ \mathbf{o}\ \mathsf{context}^-\ \mathbf{o}\ \mathsf{is\_actor\_role} &\sqsubseteq \mathsf{action\_context}^- \end{aligned}$$

[24] Of course, the relation can also be inferred in the other direction. For instance, the plays relation between some individual and its thematic role can also be inferred by specifying that e.g. every individual with an is_object relation pointing to itself plays an Object_Role.

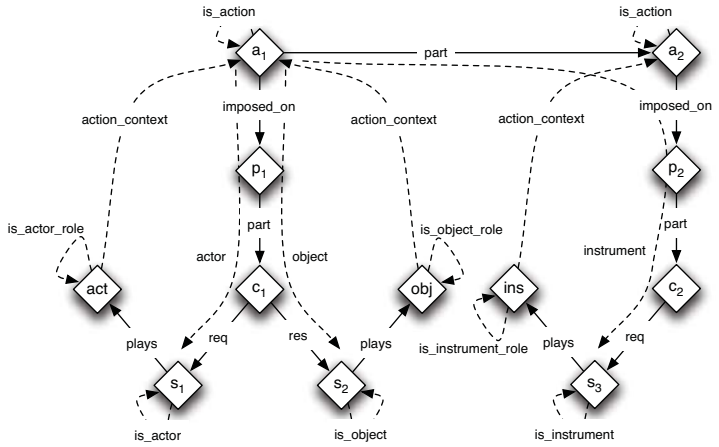

Figure 7.22: Action structure for inferring the actor, object and instrument participant roles.

The action_context property allows alternate definitions of object and actor:

$$\begin{aligned} \text{action\_context}^- \circ \text{played\_by} \circ \text{is\_object} &\sqsubseteq \text{object} \\ \text{action\_context}^- \circ \text{played\_by} \circ \text{is\_actor} &\sqsubseteq \text{actor} \end{aligned}$$

Following Schank and Abelson (1975); Breuker (1981) the instrument participant is not directly linked to the action, but is rather the object of a subordinate action. Given an explicit assertion of the plays relation between such an object and an Instrument_Role, we can infer the instrument given a composition of the part and object relations:

$$\begin{aligned} \text{Instrument\_Role} \sqsubseteq\ & \text{Thematic\_Role} \sqcap \text{is\_instrument\_role } \mathbf{some\ self} \\ & \sqcap \text{played\_by } \mathbf{only}\ (\text{is\_instrument } \mathbf{some\ self}) \end{aligned}$$

$$\begin{aligned} \text{instrument} &\sqsubseteq \text{participant} \\ \text{is\_action} \circ \text{part} \circ \text{is\_action} \circ \text{participant} \circ \text{is\_instrument} &\sqsubseteq \text{instrument} \\ \text{is\_instrument\_role} \circ \text{context} \circ \text{part}^- \circ \text{is\_action} &\sqsubseteq \text{action\_context} \\ \text{is\_action} \circ \text{part} \circ \text{is\_action} \circ \text{actor} &\sqsubseteq \text{actor} \\ \text{actor}^- \circ \text{part} \circ \text{is\_action} \circ \text{actor} &\sqsubseteq \text{same\_id\_as} \end{aligned}$$

The action_context of instrument roles is the action for which they are an indirect participant. The actor of the subordinate action is also the actor of the containing action, and consequently shares its identity. Figure 7.22 shows the action structure resulting from the definitions presented in this section, given a fictive action a_1 that consists of a process p_1 for which the requirement s_1 and the result s_2 of its constituting change c_1 are respectively the actor and object of action a_1. The instrument s_2 is the requirement of c_2 in process p_2 that forms the causal structure of action a_2, which is part of action a_1.

An Intentional Stance

The relation between an action and its actor is somewhat harder to flesh out. We have said that actions are subjective entities imposed on a process, given the intention of some agent. At the same time, this intention is exactly to perform the action, and consequently to execute the process. However, to accommodate for the possibility that an agent fails to initiate the action he *intends* to perform, the intention and action cannot be the same thing, nor can the propositional content of the intention coincide with the process. Furthermore, where an action is objective in the sense that it does not exist merely in the mind of its actor, the intention is not.

To relate intention to action, we thus need two subjective entity summarisation triangles that both have the same actor as context. First, we represent the Intention analogous to the Belief class of Section 7.3.4:

$$\begin{aligned} \mathsf{Intention} &\sqsubseteq \mathsf{is\_intention}\ \mathbf{some\ self} \\ &\sqsubseteq \mathsf{Propositional\_Attitude} \end{aligned}$$

$$\mathsf{holds}\ \mathbf{o}\ \mathsf{is\_intention}\ \mathbf{o}\ \mathsf{towards} \sqsubseteq \mathsf{intends}$$

Similarly, we summarise the relation between Agent and Process by defining an initiates role chain:[25]

$$\begin{aligned} \mathsf{performs} &\sqsubseteq \mathsf{actor}^- \\ \mathsf{performs}\ \mathbf{o}\ \mathsf{imposed\_on} &\sqsubseteq \mathsf{initiates} \end{aligned}$$

In other words, the actor of an Action is said to perform it, and executes the process that counts_as the action. The Agent that executes the process is its agentive_cause, and thus counts_as the Epistemic_Role of Cause in the context of the process.

$$\begin{aligned} \mathsf{Cause} &\sqsubseteq \mathsf{is\_cause}\ \mathbf{some\ self} \\ &\sqsubseteq \mathsf{Epistemic\_Role} \\ \mathsf{Actor\_Role} &\sqsubseteq \mathsf{played\_by}\ \mathbf{only}\ (\mathsf{initiates}\ \mathbf{some}\ (\mathsf{context\_of}\ \mathbf{some}\ \mathsf{Cause})) \end{aligned}$$

$$\begin{aligned} \mathsf{agentive\_cause} &\sqsubseteq \mathsf{initiates}^- \\ \mathsf{executes}\ \mathbf{o}\ \mathsf{context\_of} &\sqsubseteq \mathsf{imposed\_on} \end{aligned}$$

Note how the mere playing of the Actor_Role by some Agent forces it to be the initiator of some process for which it is a Cause. In the same way, the playing of this role can be used to force the Agent to hold an Intention_to_Act:

$$\begin{aligned} \mathsf{Actor\_Role} &\sqsubseteq \mathsf{played\_by}\ \mathbf{only}\ (\mathsf{holds}\ \mathbf{some}\ \mathsf{Intention\_to\_Act}) \\ \mathsf{Intention\_to\_Act} &\sqsubseteq \mathsf{is\_intention\_to\_act}\ \mathbf{some\ self} \\ &\sqsubseteq \mathsf{Intention} \end{aligned}$$

This mechanism is very powerful in that it solves the inheritance problem of the counts_as relation discussed in Section 7.3.2. Subjective entities can restrict the properties of classes they are imposed on by a nested universal restriction on the counts_as property (or its sub properties).

[25] Note that this chain does not depend on the marker property for Action as the actor property already takes this into account.

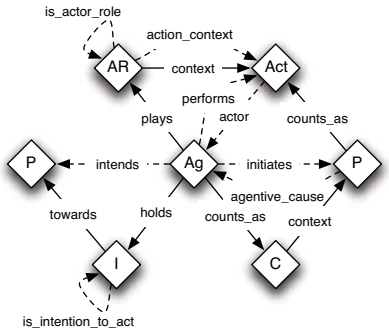

Figure 7.23: Four summarisation triangles for Action.

7.5.2 Discussion

The preceding section presents a representation of a complex concept, Action, by combining the three patterns described earlier. First, part of the *diamond* shape of the Transaction was extended to connect to the *sequence* pattern of Process. We had to lift the cardinality constraints on the participants in actions, to be able to define them as the *summarisation* property of the *triangular* pattern of subjective entities. This drawback is amended by adding additional same_id_as property chains, but otherwise the Transaction pattern remained in tact. The structure of the sequence pattern was used to constrain the allowed participants in an action, and identify the agentive cause of the process that counts as action.

It is rather easy, and tempting, to extend this structure even further. The composition of marker properties, roles, subjective entities, causes and actions gives an intuitive handle on the definition of more complex social constructs in LKIF Core, such as legal notions related to liability. Following Lehmann (2003) the agent that initiates a process which causes an event that results in some damage can be said to be *potentially liable* for that damage. Without giving a full definition of these concepts, a straightforward definition of this relation is expressible as a role chain that uses the caused_by, agentive_cause and played_by properties:

$$\mathsf{Damage} \sqsubseteq \mathsf{Legal\_Role} \sqcap \mathsf{played\_by}\ \mathbf{only}\ (\mathsf{is\_damage}\ \mathbf{some}\ \mathbf{self})$$

$$\mathsf{is\_damage} \circ \mathsf{result\_of} \circ \mathsf{caused\_by} \circ \mathsf{agentive\_cause} \sqsubseteq \mathsf{potentially\_liable}$$

The four summarisation triangles involved in the intentional perspective on actions are depicted in Figure 7.23.

7.6 Discussion

This chapter presented and discussed three design patterns inspired by, but not limited to the LKIF Core ontology presented in the previous chapter. Rather than *content* patterns, these emphasise the *structure* common to many representations. The first pattern, of Section 7.2, is inspired by the general inability

to constrain the models of a DL class description in such a way that they correspond to a *diamond*-like structure (Section 7.1.1). A typical use case for this pattern is the notion of *exchange*, but similar application can be found in the description of physical, structured objects, such as the human heart, and artefacts, such as tables and electronic devices. Whether the resulting class description is an *ontological* definition or rather a *framework* depends on its object, rather than its structure.

The design pattern was introduced in an incremental fashion, emphasising the procedural nature of design, and elucidating the decisions and trade-offs involved in the representation of the Transaction class. The procedure is composed of five steps: creating an initial class description, constrain the number of role fillers, disambiguate between role fillers, traverse the tree model of the class definition using property chains, and introduce asymmetry between the branches of the tree to allow for precise property chains. Central to this procedure is to ensure first the recognition of individuals belonging to a class, and secondly enforce compliance to the class definition when an individual is asserted to be a member. Once the definition is as strict as possible, additional information is *inferred* by the liberal use of subclass axioms and, in particular, role inclusion axioms.

The pattern introduces the same_id_as and different_id_from properties as alternatives to the built-in owl:sameAs and owl:differentFrom properties. Because the latter are part of the OWL semantics, they are subject to the tree model restriction of DL. However, the former are not and can be interpreted in two custom ways: as equivalent to their built-in counterparts, e.g. by specification of the DL-safe rule in Section 7.2.2, or as carrying the *identity* of the individuals it relates. The third pattern, of Section 7.4 illustrates that this second interpretation is particularly useful in the description of different *states* of a single entity. The use of nominals or value restrictions as placeholders in class descriptions is not a general alternative to the pattern described here as its applicability is restricted to controlled environments (see Section 7.2.2).

Section 7.3 presents a *triangular* pattern for the summarisation of reified relations, based on use cases for n-ary relations identified by the SWBP (Noy and Rector, 2006), and common ways to define social concepts. The same cannot be achieved using standard RDF reification. The main contributions of this pattern are the ability to construct and combine complex relations using simple combinations of classes and roles, and a general method for abstracting from expressive ontologies to create a simplified version that is easier to reuse.

The pattern is constructed largely by the same procedure as that of Section 7.2, but extends it with a step to introduce *domain dependence* of the summary property, and an optional *punning* step. It is characterised by the use of *marker properties* to identify classes from within the RBox, and *backfiring* of property restrictions over the relations. The latter mechanism is of particular interest to social and legal domains where role playing and e.g. deeming provisions rely heavily on the inheritance of properties from anti-rigid classes over the counts_as relation, as opposed to standard subsumption. In addition to the same_id_as mechanism of the first pattern, this pattern furthermore demonstrates a mechanism for *wrapping* a simple property inside a role inclusion to simulate a rdfs:subPropertyOf relation. This to circumvent situations where asserting this relation would make the property complex (see Section 7.3.4).

The third design pattern, described in Section 7.4, is an approach to repres-

enting *sequences* inspired both by requirements for causal explanation in law (Hoekstra and Breuker, 2007), and protein sequences in biomedical research (Drummond et al., 2006). The construction of class definitions using this pattern follows a similar methodology as that of Section 7.2 and Section 7.3. However, it differs in that membership of the sequence class depends entirely on the inferred property chain, where the other patterns primarily use property chains to infer new relations. It is very lean and flexible compared to existing approaches as sequences are defined solely by role inclusion axioms. An important benefit is that role inclusions can easily be combined to form more elaborate chains.

The combination of the three patterns in Section 7.5 shows that not only the patterns, but perhaps more importantly the *techniques* central to the patterns can be usefully applied to construct elaborate class definitions. It turns out that extension of the three design patterns can be conservative without compromising significant expressive power. This is because they are self-contained and their synthesis is only defined by means of property inclusion axioms and additional class *definitions*, rather than *restrictions* on existing classes. The principle of *backfiring* is a good example; using the subjective entity summarisation pattern, we can specify conditional restrictions on the instances of a particular class without affecting the original class definition. For instance, the Actor_Role requires every Agent that plays the role to initiate some Process, but does not alter the definition of the Agent class itself. In fact, it does not even mention it. Loose coupling of the three patterns furthermore corroborates the assumption of stratification that underlies the distinction between the different levels of description in LKIF Core (see Section 6.3).

As a general approach to ontology development, the procedures used in the patterns are limited in several ways. We have seen multiple times that property inclusions are sub properties of the property they define (Section 7.4.3). This means that although they may allow us to infer the existence of a property in more cases than before, it does not affect the pre-existing semantics of that property provided that it is already treated as *complex*. The use of such extensible properties must therefore follow the global restrictions defined in Motik et al. (2009). It is advisable that ontology engineers who employ one of the patterns explicitly mark simple properties intended for extensibility using some annotation property, similar to e.g. the meta-properties proposed by Guarino and Welty (2004) for marking the rigidity of a class (see Section 5.5.1).

A related restriction is that the assertion of individuals should take the delicate semantics of role inclusion axioms into account: asserting a value for a property defined by such a chain does not imply the existence of that chain between an individual and the value. In principle, this limitation is amendable by introducing macro-like named property chains that allow assertions and class restrictions to refer to the chain directly, rather than its super property. However, such named chains would be subject to strict global restrictions to prevent undecidability.

The third restriction follows from the custom solution to the limited expressiveness of OWL 2 in describing non-tree like structures. The same_id_as and different_id_from properties carry special semantics that is not part of OWL 2 itself, but can be very useful in the context of a knowledge based system that has to deal with the manipulation of multiple entities in different states. An ontology engineer will have to make a conscious decision to interpret them either

as equivalent to their built-in counterparts, or as identity-carrying properties.

On the other hand, the design patterns presented in this chapter demonstrate a promising avenue for a more liberal reuse of parts of ontologies. Rather than capturing a strict ontological commitment, design patterns express an ontological perspective, that can be applied to basic concepts to construct derivative notions.[26]

[26]The patterns described in this chapter are available as OWL files at `http://ontology.leibnizcenter.org`.