# Unifying Explanatory and Constructive Modeling

## Towards Removing the Gulf between Ontologies and Conceptual Models

Thomas Kühne
Victoria University of Wellington
P. O. Box 600, Wellington 6140, New Zealand
Thomas.Kuehne@ecs.victoria.ac.nz

## ABSTRACT

The universal agreement regarding modeling as a useful endeavor can hide the large divide that runs through the modeling community. The differences between explanatory and constructive modeling give rise to two almost disjoint modeling universes, each based on different, mutually incompatible assumptions, rules, and tools. This division is undesirable as it prevents modelers from fluently transitioning between these worlds and denies them the benefits afforded by the underpinnings of the opposite camp. In this paper I characterize the typing disciplines underlying these different schools of thought, identify their respective trade-offs, and propose a unified approach which treats the different world views as modes of modeling that one may transition into in either direction. I present a unifying typing framework that can form the basis for a mutual fertilization between the hitherto rather separated worlds of explanatory versus constructive modeling.

## CCS Concepts

•**Software and its engineering** → **System modeling languages;**

## Keywords

descriptive modeling; prescriptive modeling; unification

## 1. INTRODUCTION

Models in software engineering are predominantly used as a means for planning. They are hence used prescriptively, i.e., as blueprints for solutions. I refer to the respective modeling mode as *constructive* because it aims at building a product. However, there are also many examples of descriptive models, typically used during requirements elicitation and the analysis phase. Such models do not target solutions but rather attempt to describe a problem domain. I refer to the respective modeling mode as *explanatory* because it aims at understanding a subject matter.



*MODELS '16, October 02 - 07, 2016, Saint-Malo, France*



ISBN 978-1-4503-4321-3/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2976767.2976770

I follow Atkinson et al. in referring to "modeling modes" as opposed to ascribing characteristics such as "descriptive" vs "prescriptive" to models themselves [8]. One and the same model may be used in different modes, e.g., as an analysis model and then as a preliminary design model. Moreover, the terms "explanatory" and "constructive" are meant to have deeper implications compared to "descriptive" and "prescriptive" respectively (cf. Tab. 1). The latter terms are already in use in software engineering and refer more to the function of a model, rather than to the features of the supporting technology. The ability of a model to allow many useful interpretations strongly suggests that there is room for incorporating some of the features found in explanatory technologies, such as ontologies [11], into technology that supports software engineering models.

Indeed, prior research has observed that descriptive and prescriptive techniques complement each other [2], attempted to explain similarities and differences [9], showed that ontological principles can inform conceptual modeling [13], aimed at creating synergy between the technologies [15], and suggested ways of reducing the gulf between explanatory and constructive technologies [1, 3, 16].

However, to the best of my knowledge no prior research exists that aims at supporting a fluid transition between explanatory and constructive modeling modes by identifying and addressing a fundamental difference in the typing disciplines traditionally associated with the respective technologies. In Section 2 I elaborate on this difference and perform a trade-off analysis that results in a list of requirements for a unified framework. I then present my unifying approach (Sect. 3) before concluding with ideas for future work (Sect. 4 & Sect. 5).

## 2. BACKGROUND

Table 1 lists a number of differences between the explanatory and constructive modes of modeling in terms of properties (top half of Tab. 1) and supporting technologies (bottom half of Tab. 1). Table 1 aims at an extreme characterization and there indeed are situations, approaches, and tools that alleviate many of the dichotomies to the point where a transition between the modes sometimes becomes feasible. No complete general solution exists, however, that bridges the gap between the typing disciplines typically used in the underlying technologies that support the modeling modes. In the following trade-off analysis between explanatory and constructive modeling I therefore focus on these hitherto unaddressed different typing disciplines.

| | Explanatory | Constructive |
|---|---|---|
| **Function** | descriptive | prescriptive |
| **Goal** | understanding | building |
| **World View** | open world | closed world |
| **Growth** | organic, bottom-up (instances first) | normative, top-down (types first) |
| **Typing** | structural | nominal |
| **Levels** | two-level | multi-level |
| **Models** | ontologies (e.g. using OWL) | conceptual models (e.g., using UML) |
| **Logic** | description logic | first-order logic |

**Table 1: Two Schools of Thought**

For each school of thought I address the questions of when

- an object is an instance of a concept, and
- a type is a subtype of a supertype.

I regard a type as a concept $T$ with an intension and an extension [7]. In the interest of a more compact notation and better readability, I use $T.\varepsilon$, rather than $\varepsilon(T)$ (cf. [17]) to denote the extension of $T$ (i.e., all elements falling under the concept $T$). Likewise, I use $T.\iota$ to denote $T$'s intension (a conjunction of predicates characterizing whether an element falls under the concept or not).

## 2.1 Explanatory Modeling

Technologies supporting explanatory modeling, such as ontologies, typically assume "structural typing", i.e., make the membership of an instance only dependent on whether or not it has the required properties:

$$T.\varepsilon_d = \{x \mid T.\iota(x)\} \tag{1}$$

I use the subscript "$d$" to indicate "descriptive typing". While "structural typing" is the standard technical term, I am replacing "structural" with "descriptive" in order to avoid the impression of an exclusive reliance on an object's structure. The intension $T.\iota$ is free to require any properties, including behavioral traits.

Definition (1) gives rise to subsumption "$<_d$" –

$$\begin{aligned} T_2 <_d T_1 &\rightleftharpoons (T_2.\iota \rightarrow T_1.\iota), \text{or} \\ &\rightleftharpoons T_2.\varepsilon_d \subseteq T_1.\varepsilon_d \end{aligned} \tag{2}$$

– aka the "is-a" relationship between types [6]. For so-called rigid types [12], "$<_d$" will hold for every instance at every possible time in each possible world.

### 2.1.1 Advantages

Descriptive typing establishes a simple relationship between intensions and extensions. For instance, from (1) we immediately get

$$(T_1.\iota \sim T_2.\iota) \rightarrow (T_1.\varepsilon_d = T_2.\varepsilon_d) \tag{3}$$

i.e., equivalent intensions result in identical extensions. Equation (3) is formulated as an implication rather than an equivalence since explanatory modeling typically implies an open world assumption. The latter means that even though two different intensions $T_1.\iota$ & $T_2.\iota$ may result in identical extensions with respect to a known universe at a particular point in time $t$, they may result in different extensions at $t'$. However, we only regard intensions as being equivalent, if they imply the same extensions in all possible worlds.

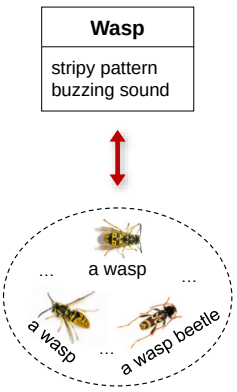


**Figure 1: Mimicry**

The straightforwardness of descriptive typing entails a desirable quality of a typing discipline, leading us to

REQUIREMENT R1. *Objects that fit a concept, should be classified accordingly without requiring manual effort.*

### 2.1.2 Disadvantages

The "*automatic capture*" quality of descriptive typing is a blessing (cf. R1) but also a curse: Consider Fig. 1 depicting the extension of type Wasp. According to the principle "*if it walks like a duck and quacks like a duck, it is a duck*", a wasp beetle instance is classified as a proper Wasp due to its mimicry, which in this case extends to outer appearance and sound effects.

For some modelers such casual classification may not present a problem, but it is easy to imagine that some modeling applications require much more stringent membership approval. While the scenario of Fig. 1 may seem easy to fix, e.g., by requiring the presence of a sting, notice that unless it is possible to identify mutually exclusive properties – e.g., for wasps versus bees – due to (2), type Bee will always be considered to be a subtype of Wasp, or vice versa. If a respective taxonomy is regarded as inadequate for whatever reason then descriptive typing implies considerable effort in maintaining adequate intensions. I chose the term "*maintaining*" because even perfectly discriminating intensions at time $t$, may not be adequate at time $t'$ anymore. Just consider a known universe without wasp beetles for which type Wasp would be perfectly suited, only to be spoiled by the discovery of the first wasp beetle instance. Therefore, we should request

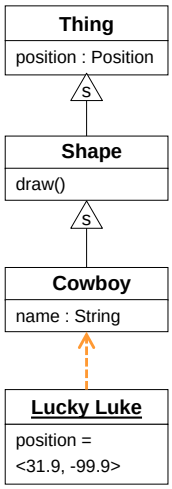


**Figure 2: Pre-Refactoring**

REQUIREMENT R2. *Taxonomy maintenance should be cost-effective.*

Figure 2 depicts a scenario – the arrival of Lucky Luke – which incurs even higher cost in the form of required refactoring. Note that Fig. 2 does not repeatedly show features for subtypes so for example Cowboy also has a draw method.

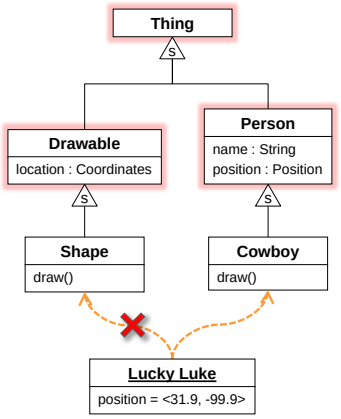


**Figure 3: Post-Refactoring**

In this scenario Lucky Luke is therefore classified as a Shape and fixing this undesirable capturing of an instance to a type requires considerable refactoring of the types involved (cf. shaded types in Fig. 3). Again, avoiding such inadvertent instance captures through very discriminative intensions comes at a cost and, in general, cannot be achieved for all time.

As a result, I postulate

REQUIREMENT R3. *Undesired classification should be addressable with adequate cost.*

## 2.2 Constructive Modeling

Constructive modeling technologies predominantly use nominal typing, i.e., employ explicit typing relationships between objects and their types. Types still have (implicit) intensions but these are satisfied "by construction", i.e., whenever an object is incarnated from a type it automatically complies to it.

$$\forall x : (x.type = T) \rightarrow T.\iota(x) \tag{4}$$

We could consider explicit incarnation histories in order to formally model object instantiation but for the sake of simplicity we will just assume a "*type*" property for an object $o$ to hold the value of a nominal type in case the latter classifies $o$.

Unlike in descriptive typing (cf. (1)), extensions of nominal types are not implied by type intensions but are formed by explicit type membership relationships. One may thus view extensions as explicit enumerations of type incarnations.

$$T.\varepsilon_n = \{o \mid o \in \Upsilon \wedge o.type \leq_n T\} \tag{5}$$

Note that both universe $\Upsilon$ and nominal extension $\varepsilon_n$ should be parameterized with respect to time to capture the notion of a *dynamic extension* [19]. However, for the sake of a simpler presentation, I leave the explicit treatment of time to future work.

### 2.2.1 Advantages

The closed world assumption in constructive modeling implies that any object always has a known classifier as the only way objects may enter the known universe is through incarnation from a known type. It is therefore never necessary to use intensions for working out which objects are classified by a type. As a consequence, intensions play a rather different role in nominal typing.

Figure 4 demonstrates that the extensions of nominal types may be different even though the respective intensions are equivalent. This allows modelers to sharply define a type (essentially through extensional enumeration) without explicitly using a highly discriminative intension. Such high level of control obviously also avoids unintentional captures

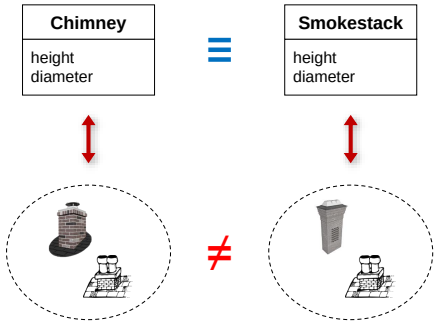


**Figure 4: Nominal Typing**

as it may occur with descriptive types (cf. Sect. 2.1.2) and furthermore allows separating types even if their instances may not be distinguishable (at a certain desirable level of abstraction). Therefore we ask for

REQUIREMENT R4. *Types should be distinguishable even in the absence of phenomenological differences between their instances.*

The same high level of control is afforded by nominal typing with respect to subtyping. Again, subtyping relationships between types are not implied by intensions but explicitly controlled by modelers. As in descriptive typing, nominal subtyping typically implies that the subtype's intension is stronger than the suberconcept's intension, but unlike in descriptive typing, no typing relationship follows from such intension implication (cf. (2)):

$$T_2 <_n T_1 \nleftarrow (T_2.\iota \rightarrow T_1.\iota)$$

For instance, even if Bee's intension builds on Wasp's intension by adding a *nectar collection* trait, Bee will only be considered a subtype of Wasp if the modeler explicitly introduces such a subtyping relationship. Consequently, it becomes much easier to honor the Liskov Substituation Principle (LSP) [18] as one only needs to establish subtyping relationships when subtype behaviour is known to be compatible with supertype behavior.

Technically, subsumption (2) guarantees the LSP automatically but only if all relevant behavior is explicitly specified through intensions. However, the latter is very hard to achieve in practice because of the challenges involved in capturing all relevant behavior and the respective effort and modeler skills required. It seems advisable to always require modeler authorization for subtyping even in the presence of behavior capturing intensions in order to avoid granting substitutability before sufficient evidence for true substitutability has been gathered. It is thus appropriate to request

REQUIREMENT R5. *Substitutability should be granted by modelers, as opposed to being assumed on the basis of typically incomplete intensions.*

### 2.2.2 Disadvantages

The ease with which nominal types can be sharply defined through their extensions invites negligence regarding the maintenance of accurate intensions and hence provokes the existence of distinct types that do not self-document their respective differentiae. We thus call for

REQUIREMENT R6. *Types should be defined as explicitly as tenable in terms of creation and maintenance cost.*

# 3. UNIFYING FRAMEWORK

The trade-off analysis of Sect. 2 suggests that descriptive and nominal typing have weaknesses and strengths that complement each other. The high cost associated with the creation and maintenance of highly discriminating types in declarative typing (cf. R2–R5) is avoided in nominal typing. Conversely, the inadequacy of nominal typing to support an open world assumption due to the implied need to individually assign every discovered object manually (cf. R1) and its potential for over-reliance on type names rather than explicit type intensions (cf. R5), can be addressed by incorporating aspects of descriptive typing.

The aim of the unified framework presented here is to allow the unimpeded application of explanatory and constructive modeling modes while providing consolidating abstractions that support fluid transitioning between them.

The first key to a successful unification of the two typing disciplines is the observation that any constructive type $T$, i.e., an instance generator, also has the potential to be used in a descriptive capacity by referencing its intension. In constructive settings one will typically not encounter explicit intensions but the set of features specified for instances forms an implicit intension and the thus accepted set of instances ($\varepsilon_d$) will likely to be larger than the actual nominal extension ($\varepsilon_n$) that is created through constructors and modified by invoking operations. For example, a type representing a collection may always produce and maintain instances where the value of a count property matches the number of elements in a container feature. Unless a respective constraint is added to the type as an invariant, the implicit intension will allow – i.e., be considered to describe – any collection instance where the count value and the number of elements in the container are unrelated. We can thus use the difference between the nominal extension $T.\varepsilon_n$ and the descriptive extension $T.\varepsilon_d$ of a type as a basis for using a single type in two roles; a descriptive role and a characterizing role.

## 3.1 Conformance

A type describes an object, if and only if the object satisfies the type's intension:

$$o \lhd_d T \rightleftharpoons T.\iota(o) \tag{6}$$

We may say the object conforms to the type and subsumption ($<_d$) follows as in (2):

$$T_2 <_d T_1 \rightleftharpoons (\forall o : o \lhd_d T_2 \rightarrow o \lhd_d T_1) \tag{7}$$

## 3.2 Explicit Type Declarations

Before I can fully define the characterizing role of a type, I need to introduce the second key to a successful unification; the modification of the notion of a nominal extension so that nominal typing can accommodate object discovery as occurring in explanatory modeling. I propose to generalize the idea of the set of *direct instances* of a type – i.e.,

$$T.\varepsilon_{di} = \{x \mid x \in \Upsilon \wedge x.type = T\} \tag{8}$$

– to not only include objects that have been generated using $T$, but also objects have been *adopted* by $T$. Consider the golden retriever gary in Fig. 5. It is described by types Mongrel, Dog, and Animal. All these types are *candidates* for becoming gary's nominal type. If Dog is chosen, for example, all other candidate types will still continue to describe gary, but gary then maintains an "adopted" relationship to Dog (cf. fido and Poodle) and Dog becomes gary's characterizing type (cf. Sect. 3.3). With $T.\varepsilon_o$ only containing the *offspring* of $T$, i.e., the instances generated using $T$, we have

$$o \lhd_a T \rightleftharpoons (o \lhd_d T) \wedge (o.type = T) \wedge (o \notin T.\varepsilon_o) \tag{9}$$

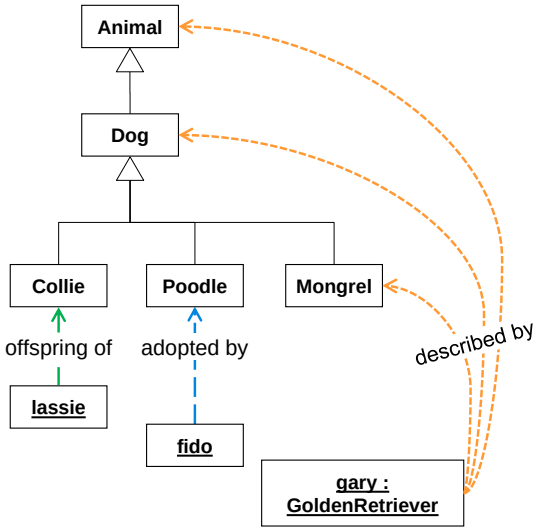


**Figure 5: Explanatory Modeling Choices**

Note that while none of the types in Fig. 5 ideally characterizes gary – no GoldenRetriever type is available – this does not present a problem in nominal typing. A modeler has the option of adding another suitable dog breed type but may also just adopt gary to any of the available types it conforms to. A respective type assignment will not enable access to the full set of gary's features, unless some kind of runtime interrogation of features is supported, but it is the prerogative of the modeler to forgo the respective opportunities.

## 3.3 Characterization

Object adoption gives the modeler a means to nominate a type for an object that not only describes but *characterizes* the object, i.e., provides the most adequate classification amongst a given set of types.

$$o \lhd_n T \rightleftharpoons \begin{array}{l} o \lhd_d T \wedge \\ o.type \leq_n T \end{array} \tag{10}$$

An object that has been assigned a characterizing type is said to be *bound* –

$$\beta(o) = \begin{cases} false, & o.type = \bot \\ true, & \text{otherwise} \end{cases} \tag{11}$$

– e.g. in Fig. 5 both lassie and fido are bound. Note that they are still described by Animal, Dog, etc. but these relationships have been omitted from Fig. 5 for clarity.

Of course we have

$$o \lhd_n T \rightarrow \beta(o) \tag{12}$$

Any object characterized by a type is in the latter's set of direct instances $\varepsilon_{di}$ so with this abstraction of both offspring and adoptees (cf. (8) & (9)) in place, we can define the nominal extension of a type:

$$
\begin{aligned}
T.\varepsilon_n &= T.\varepsilon_{di} \cup T.\varepsilon_s, \text{ where} \\
T.\varepsilon_s &= \bigcup_{S_i \in T_s} S_i.\varepsilon_n \text{ and} \\
T_s &= \{s \mid s <_n T\}
\end{aligned} \tag{13}
$$

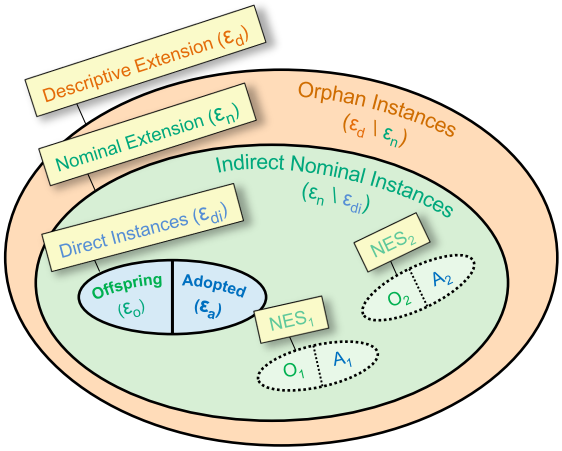


**Figure 6: Unified Framework**

Figure 6 visualizes all sets defined so far and illustrates (13), i.e., that the nominal extension of a type consists of its direct instances ($\varepsilon_{di}$) and all (indirect) nominal instances ($\varepsilon_s$) contributed by its nominal subtypes. Figure 6 also depicts that the nominal extension ($\varepsilon_n$) is a subset of the descriptive extension ($\varepsilon_d$) and that the complement of the former with respect to the latter ($\varepsilon_d \setminus \varepsilon_d$) can be referred to as the set of *orphans* of a type. I suggest the term "orphans" because these instances may or may not become adoptees of the type in the future.

## 3.4 Explicit Subtype Declarations

From requirements R2 and R3 we know that subsumption on its own would not be sufficient to maintain taxonomies in a cost-effective manner. The unified framework therefore retains the idea of explicit subtype relationships from nominal typing. A type $S$ can be declared to be subtype of another type $T$, if $T$ subsumes $S$:

$$
S <_e T \rightarrow S <_d T \tag{14}
$$

The explicitly declared subtyping relationships "$<_e$" give rise to a transitive nominal subtyping relationship:

$$
\begin{aligned}
R <_n T \;\rightleftharpoons\; & R <_e T \vee \\
& \exists S : (R <_e S <_n T)
\end{aligned} \tag{15}
$$

Nominal subtyping should be free of cycles and shortcuts:

$$
\begin{aligned}
\textbf{acyclic} \quad & \forall T_1, T_2, i \geq 2 : T_1 <_n{}^i T_2 \rightarrow T_1 \neq T_2 \\
\textbf{shortcut-free} \quad & \forall i \geq 2 : <_n{}^i \cap <_e = \emptyset.
\end{aligned}
$$

## 3.5 Consolidated Classification

With description ($\lhd_c$) and characterization ($\lhd_n$) the framework features two separate typing relationships. In order to integrate them into a unified framework, I define a consolidated typing relationship "$\lhd$": An object o is an instance of a type $T$, if and only if, o conforms to $T$, and o being bound implies that its characterizing type is (a subtype of) $T$.

$$
\begin{aligned}
o \lhd T \;\rightleftharpoons\; & o \lhd_d T \wedge \\
& \beta(o) \rightarrow o \lhd_n T
\end{aligned} \tag{16}
$$

Conformance to $T$ is a necessary but not a sufficient condition for an object $o$ to be regarded as being an instance of $T$. This design introduces the third key idea underlying the proposed framework, i.e., the notion that the descriptive or characterizing role of a type is selected by objects. Unbound objects, i.e. orphan objects that have no explicit characterizing type (yet), enjoy the automatic classification and implicit taxonomy generation known from explanatory modeling. The framework thus addresses requirement R1. Bound objects, however, are considered to require a more stringent treatment. If an object has a characterizing type, it is assumed to be exacted to discerning classification requirements that go beyond simple descriptive typing. Binding an object can be regarded as selecting a subset of the descriptive types as certified types that appropriately characterize a now fully acknowledged citizen of the known universe. For bound objects, the $\lhd$ relationship essentially discards the complement of the $\lhd_n$ relationships with respect to the $\lhd_d$ relationships. Hence object binding represents a very cost effective approach to avoid unintended instance captures (cf. Sect. 2.1). The framework thus satisfies requirements R2 & R3. Retaining the ability for characterizing types to maintain distinct extensions despite equivalent intensions furthermore satisfies R4.

## 3.6 Subtyping

In analogy to the approach used for classification ($\lhd$), I finally integrate conformance ($<_d$) and characterization ($<_n$) into a consolidated classification relationship "$<$": A type $S$ is a subtype of another type $T$ with respect to an object o, if and only if $T$ subsumes $S$, and $o$ being bound implies that $S$ is a nominal subtype of $T$.

$$
\begin{aligned}
S <_{(o)} T \;\rightleftharpoons\; & S <_d T \wedge \\
& \beta(o) \rightarrow S <_n T
\end{aligned} \tag{17}
$$

This definition introduces the fourth key idea underlying the proposed framework, i.e., the notion that subtyping is not uniformly defined for all objects but distinguishes between unbound and bound objects. As in the case of consolidated classification ($\lhd$), bound objects are subjected to discerning requirements that go beyond mere conformance checks. For example in the scenario shown in Fig. 2 we do not want Shape to indirectly classify Lucky Luke by virtue of being a supertype of Lucky Luke's type Cowboy, once we recognized Lucky Luke as being a cowboy and only a cowboy. Explicitly characterizing Lucky Luke as a cowboy communicates that Lucky Luke is known to be a person, etc. but not a shape. This implies that only certified subtyping relationships ($<_n$) should be considered for bound objects. That is why no refactoring is necessary in Fig. 2 if Lucky Luke is bound to Cowboy, because (with respect to Lucky Luke) Shape will not be considered a supertype of Cowboy anymore. Shape only subsumes Cowboy and as this relationship has not been certified into a nominal subtype relationship, it is not considered for bound objects with respect to consolidated subtyping.

## 3.7 Discussion

The purpose of the unified framework is to support both explanatory and constructive modeling modes without causing any compromises to either mode. With respect to the typing disciplines it unifies, the framework achieves this by retaining the original descriptive typing ($\lhd_d$, $<_d$) and nominal typing ($\lhd_n$, $<_n$) relationships. Therefore, all respective advantages are retained and the framework thus satisfies R1, R4, and R5.

The abstracted, consolidated relationships, "$\lhd$" & "$<$", add value in two ways: First, they give modelers a common vocabulary and, second, allow shortcomings of one typing discipline to be addressed by the opposing one. A diverse set of modelers may therefore work collaboratively despite different underlying default assumptions of what constitutes a type/supertype. Whether an object has been assigned a characterizing type is public information and hence everyone involved is always fully informed what the types/supertypes of a particular object are, i.e., how "$\lhd$" and "$<$" relationships are interpreted.

It is worth noting that there are only two categories of objects which have respective typing rules associated to them: First, unbound objects are regular objects from a descriptive point of view. Whether they always existed or are discovered at some point, the standard descriptive typing rules apply to them. From a nominal typing point of view these objects are "orphans" (cf. Fig. 6). A constructive modeler regards their descriptive types as candidates for a proper characterizing type, but as long as such objects remain unbound, they can be regarded as *quarantined*. From a constructive modeling point of view the descriptive types associated with an object, in general, are too indiscriminative in order to allow the objects to be used with such types in discerning contexts, such as safety-critical software. Only after using binding to elevate the status of an orphan to that of a certified member of a nominal type, will the object be admitted to critical contexts with the type/supertypes that remain after all $\lhd_d \setminus \lhd_n$ have been eliminated as recognized types. As mentioned before, this scheme prevents cowboys from being used as shapes or vice versa (cf. Sect. 2.1) without requiring refactoring or modification of intensions. As a result, R3 is satisfied.

If a purely descriptive taxonomy already works perfectly, i.e., if there are no unintended instance captures, there are two options: Objects classified by such a taxonomy could be considered as "safe" to use with their respective types. Alternatively, all automatically derived typing relationships could be confirmed manually with respective explicit nominal declarations.

The second category of objects are bound objects. They are regular objects from a constructive point of view. They have a known characterizing type, essentially help to sharpen their types through enumerating the latters' extensions, and only acknowledge certain explicitly certified subtype relationships between their (super-)types. From an explanatory modeling point of view, bound objects are elect objects that opted out of certain typing relationships. To support this capability in a purely explanatory approach, one would have to consider an oracle that could be called upon in intensions in order to explain why such objects do not conform to all descriptive types despite their face value conformance. In the unified approach, a modeler can take the role of such an oracle. While it would be possible to embody the modeler's decisions within type intensions and thus make them repeatable in an automated manner for any new object discoveries, maintaining such taxonomies would come at a high cost (cf. Sect. 2.1). Giving modelers the option to avoid such cost makes the framework satisfy R2.

Due to the fact that descriptive typing relationships do not disappear for bound objects – they are just not acknowledged in the consolidated "$\lhd$" and "$<$" relationships – it is possible to compare the nominal classification relationships with the descriptive ones. If desired, a model/ontology may be iteratively refactored – through modifications to the intensions – until all nominal and descriptive relationships exactly coincide. Such a set of descriptive types could be labeled as "*perfectly discriminating*". The fact that the framework supports such endeavors makes it satisfy R6.

Summarizing, modifying the "bound" status of an object by assigning/unassigning a directly characterizing type to it – whether actually or just hypothetically – makes it possible to address shortcomings in one typing discipline by exploiting the advantages of the opposing one. The respective "*per-object-category switch*" for typing rules lies at the heart of the proposed unified framework. Its novel nature can be easily understood by phrasing it in terms of description logic [5]: Essentially, "ABox"-facts (whether an object is bound or not), select one of two available "TBoxes" (i.e., typing rules). While this may seem unsettling at first, the approach should be uncontentious with respect to "$\lhd$" because answering questions about an object's type naturally depends on the object itself (here, including whether it is bound or not).

It is considerably more unconventional, though, to make subtyping relationships between types dependent on object categories, i.e., to use "$<_{(o)}$" rather than static relationships that universally apply to all objects. Yet, I maintain that it is just a matter of perspective to allow this design to be recognizable as entirely adequate: After all, subtyping relationships in themselves are not of particular interest. Their relevance is created by the implications they have on which types constitute alternative, indirect types of an object, e.g., whether Shape is an indirect type of Lucky Luke or not. If one phrases the "subtype/supertype" question from the perspective of an object in the form of "*What are an object's indirect types?*" then it becomes obvious that it is perfectly valid to make the answer to this question dependent on the object (here, including whether it is bound or not).

# 4. CONCLUSION

In this paper I proposed a step towards removing the gulf between explanatory and constructive modeling technologies. There are many commonalities between

- ontologies and conceptual models,
- descriptive and prescriptive models,
- and the technologies used to support them.

Essentially, knowledge engineering and knowledge representation underlie many similar but separated applications of modeling and while this recognition is a prerequisite for increasing the synergy arising between these separated areas, the main challenge is to find ways in which the differences between these separated areas can be addressed.

The work presented here focused on addressing the significant differences between the typing disciplines that underpin explanatory versus constructive approaches. The presented unifying framework can support both typing disciplines while providing consolidating concepts that abstract away from the differences between them. I believe in order for the framework to become accepted as a common basis and thus support further integration of explanatory and constructive technologies, it is necessary that modelers from neither camp should have to adapt to any conventions or even workarounds that are not native to their original approach. I therefore retained the core of each typing discipline and provided an additional abstracting layer of consolidating notions on top of them, plus a mechanism to allow fluid transitioning between the typing disciplines. I maintain that the proposed framework succeeds in allowing modelers with diverse background assumptions to agree on classification and subtyping relationships without compromising the advantages of their familiar typing discipline. On the contrary, judicious use of transitioning objects between typing disciplines can address the shortcomings inherent to either typing discipline.

The framework I propose is based on four key ideas:

1. All types have two roles; a descriptive role (through their intension) and a characterizing role (through explicitly maintaining an extension).

2. Discovered objects can be assimilated into a nominal typing regime through *object adoption*, i.e., by assigning a characterizing type to them. The resulting abstraction of *direct instances* of a characterizing type is a generalization of both generated instances and adoptees.

3. Whether consolidated classification only considers characterizing types or also includes descriptive types depends on the status of the object in question. If an object is *bound* – i.e., has a characterizing type – then all purely descriptive types are not acknowledged as consolidated classifiers.

4. Consolidated subtyping also makes a difference between unbound and bound objects. From a nominal perspective, these object categories represent *orphans* and *certified citizens* respectively. For certified citizens, the set of (indirect super-) types is reduced to those that are established by explicit certified subtyping relationships, as opposed to being based on the typically much more liberal subsumption relationships.

As discussed in Sect. 3.7, this approach satisfies all requirements that were collected in the initial trade-off analysis of the typing disciplines (cf. Sect. 2). In addition, the framework supports a mixed mode environment in which notions like *quarantining* and *adoption* are supported. These notions do not exist in either typing discipline separately but enable a collaborative cooperation between diverse modelers in the unified framework, or alternatively, simply enable a single modeler to choose the most appropriate typing approach for a task at hand.

The trade-off analysis revealed that descriptive typing is strong on automatically recognizing the maximum amount of classification potential and equally excels at explicitly capturing the nature of types, making the latter self-documenting. Its weaknesses – a high cost of maintaining intensions and useful taxonomies – are perfectly addressed by nominal typing with its cost-effective "*by declaration*" approach to classification. Conversely, the weaknesses of nominal typing – the potential of an over-reliance on type names rather than explicit definitions and the inability to deal with discovered objects – are perfectly addressed by descriptive typing and the notion of object adoption.

The completeness with which descriptive and nominal typing complement each other suggests that their integration into one unified framework should not only be regarded as a means to reduce a gulf between separated worlds, but rather as a desirable typing discipline in its own right.

# 5. FUTURE WORK

In order to increase the clarity of presentation I did not formally capture time, i.e. the possibility for dynamic extensions to grow and shrink over time, and the possibility of known universes to expand due to object discovery. For a similar reason and due to space constraints, I did not discuss "multiple classification" either, i.e., the ability of an object to have multiple direct types. I plan to add both aforementioned aspects in the future.

A less clearcut question is how to define the semantics of "potency" [4] in the unified framework. In its present form the framework only addresses two-level paradigms in the form they are commonly encountered in explanatory technologies. Increasingly, however, both explanatory and constructive environments support multi-level modeling and the presented framework should therefore be extended to cover multiple levels of classification, including "potency" specifications.

Since the framework is intended to improve practical modeling, I plan to implement the presented ideas with a tool. The open source project Melanee [3] is an excellent platform to validate the approach and support an exploration of appropriate interaction patterns regarding quarantining, adoptions, etc. Melanee already supports both explanatory and constructive modes of modeling to some extent [16] and ideas regarding identifying type candidates and assigning characterizing types are currently explored as part of a master thesis that aims at adding better connection support to Melanee [10]. A respective implementation will furthermore facilitate the exploration of ideas such as allowing *abandonment* as an inverse to *adoption*. Such an operation would allow instances to be created from a known type (e.g., Cuckoo) and let them be adopted by other types (e.g. Sparrow), for example to explore how a system would respond to potential future adoptions of discovered objects.

A tool supporting the unified framework could moreover provide means to measure how close a descriptive taxonomy is to being "*perfectly discriminating*" (cf. Sect. 2.1.2). Further taxonomy styles could be identified and checked by the tool, such as a "*minimally concrete*"-style, in which only leaf types in a taxonomy are allowed to be characterizing types [14].


## Acknowledgments

I would like to thank Colin Atkinson for numerous debates on the topics presented here. These have sharpened some of the concepts and significantly influenced their presentation.

## 6. REFERENCES


[1] U. Aßmann, A. Bartho, and C. Wende, editors. *Reasoning Web. Semantic Technologies for Software Engineering*, volume 6325 of *LNCS*. Springer, 2010.

[2] U. Aßmann, S. Zschaler, and G. Wagner. *Ontologies for Software Engineering and Software Technology*, chapter Ontologies, Meta-models, and the Model-Driven Paradigm, pages 249–273. 2006.

[3] C. Atkinson and R. Gerbig. Melanie: Multi-level modeling and ontology engineering environment. In *Proceedings of Modeling Wizards'12*. ACM, 2012.

[4] C. Atkinson and T. Kühne. The essence of multilevel metamodeling. In M. Gogolla and C. Kobryn, editors, *Proceedings of the $4^{th}$ International Conference on the UML 2000, Toronto, Canada*, LNCS 2185, pages 19–33. Springer Verlag, Oct. 2001.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] R. Brachman. What is-a is and isn't: An analysis of taxonomic links in semantic networks. *Computer*, 16(10):30–36, 1983.

[7] R. Carnap. *Meaning and Necessity: A Study in Semantics and Modal Logic*. University of Chicago Press, 1947.

[8] B. G. Colin Atkinson, Bastian Kennel. Supporting constructive and exploratory modes of modeling in multi-level ontologies. In *Procs. 7th Int. Workshop on Semantic Web Enabled Software Engineering*, 2011.

[9] M. G. Colin Atkinson and K. Kiko. On the relationship of ontologies and models. In *Proceedings WoMM'06.*, volume 96 of *LNI*, 2006.

[10] T. K. Colin Atkinson, Ralph Gerbig. A unifying approach to connections for multi-level modeling. In *Proceedings of MODELS'15*, pages 216–225. IEEE Computer Society, 2015.

[11] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer, 1993.

[12] N. Guarino. Concepts, attributes and arbitrary relations. *Data & Knowledge Engineering*, 8:249–261, 1992.

[13] G. Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, Enschede, The Netherlands., 2005.

[14] W. L. Huersch. Should superclasses be abstract? In *ECOOP '94*, LNCS, pages 12–31. Springer, 1994.

[15] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In *Proceedings of MODELS'06*, volume 4199 of *LNCS*, pages 528–542. Springer, 2006.

[16] B. Kennel. *A Unified Framework for Multi-level Modeling*. PhD thesis, University of Mannheim, 2012.

[17] T. Kühne. Matters of (meta-) modeling. *Software and System Modeling*, 5(4):369–385, 2006.

[18] B. H. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, Nov. 1994.

[19] F. Steimann and T. Kühne. A radical reduction of UML's core semantics. In *Proceedings of UML'02*, volume 2460 of *LNCS*, pages 34–48. Springer, 2002.