A Unifying Approach to Connections for Multi-Level Modeling

Colin Atkinson, Ralph Gerbig
University of Mannheim
Software Engineering Group
Mannheim, Germany
Email: {atkinson, gerbig}@informatik.uni-mannheim.de

Thomas Kühne
Victoria University of Wellington
School of Engineering and Computer Science
Wellington, New Zealand
Email: tk@ecs.vuw.ac.nz

***Abstract*—Capturing relationships between concepts in a domain is as important as capturing the concepts themselves. Modeling languages reflect this by providing connections with rich semantics, such as associations and links, thus providing a key advantage over approaches that support relationships with simple references only. While connections for two-level modeling (e.g. in the UML) have enjoyed a stable design for a considerable time, the same cannot be said for connections in multi-level modeling languages. As interest in multi-level modeling grows, it is important to provide a comprehensive design for connections that not only adheres to multi-level principles such as level-agnosticism and explicit level organization, but also supports deep characterization, i.e., the ability to specify level content beyond one level boundary. In this paper we propose a unifying conceptual model for connections whose expressiveness and scalability does not come at the cost of concept proliferation.**

I. Introduction

As the principles and features of multi-level modeling [1] crystallized over the years from papers published by the modeling community, attention naturally focused on elaborating the core features of the approach first. Thus, ideas such as using an explicit level organization principle, integrating instance and type facets, and deep characterization, were naturally first elaborated around entities (clabjects) that are typically at the center of discussions in modeling. The relationships between these entities, however, have generally received much less attention and are therefore comparatively under-explored. This lack of attention to the structure and semantics of relationships in multi-level modeling has not posed a significant problem until now because the vast majority of multi-level models published to date have been small examples devised to illustrate features of a particular language or tool. While these examples made use of relationships, the latter were not expanded upon, with modelers simply applying the concepts and conventions associated with UML relationships in the way that seemed the most appropriate for their particular modelling situation. Most approaches have opted to treat relationships as clabjects [2], [3], [4] but only a few authors have given relationships explicit consideration [5], [6], [7].

This historical "second class" treatment of relationships in multi-level modeling has led to a divergence of relationship modeling approaches, with differences regarding both the conceptual structure of relationships and their presentation. To provide a solid foundation for the continued evolution and adoption of multi-level modeling it is therefore desirable to define a unified approach for conceptualizing and representing relationships in multi-level modeling which is compatible to the greatest extent possible with UML concepts and conventions, and clarifies the semantics of relationship features that have to date not be defined in a multi-level context.

Pursuing these goals requires a number of challenges to be tackled: First, any successful design for relationships in multi-level modeling should be able to support the creation of large scale models. This means devising a relationship referencing scheme that goes beyond traditional simple naming conventions which only work for small models. Second, opportunities for the consolidation and unification of current UML relationship concepts within an overall simpler level-agnostic conceptualization should be exploited. Just like the single notion of a multi-level clabject subsumed UML classes, stereotypes, powertypes, and tagged values in the past [8], a well-designed multi-level relationship approach has the potential of providing conceptual simplification while retaining, or even increasing, expressiveness. Third, features from multi-level modeling, such as deep characterization [2], should be adapted for relationships in a manner that is intuitive and expressive at the same time. Fourth, new terminology is required that acknowledges the different requirements of a multi-level modeling context compared to traditional two-level approaches. This will enable future discussions to be held with more precision and clarity.

The goal of this paper is to discuss the issues involved in meeting these challenges and identify the key features of a unified conceptualization of relationships in multi-level modeling, which include –

- natural terminology,
- a clean and simple conceptual underpinning,
- sound and expressive semantics, and
- an intuitive notation.

In the rest of the paper we refer to explicit relationships between entities at any level as *connections*. This serves to distinguish them from mere object-owned references/pointers and avoids any traditional absolute type/instance connotations that are attached to terms like "association" and "link".

We regard any such UML relationship as a connection, including dependencies, specializations, higher-arity associations, etc. and capture their different nature, i.e., intra-level significance only vs impact on levels below, through the notion of connection potency, analogous to entities [2].

We refer to the ends of connections as *monikers* because they are essentially origin-specific aliases to entities that may be known through other names to other entities. We prefer this to the widely used term "role name" since the latter implies a "role" / "player" distinction with "roles" having much deeper semantics compared to aliases [9]. We are not arguing against an explicit recognition of proper "roles", though, and our proposal is designed to allow their incorporation in the future.

In the following we identify challenges associated with connections in multi-level modeling (Sect. II), present a unifying connections approach (Sect. III), evaluate the proposed approach (Sect. IV), discuss related work (Sect. V), and finally conclude (Sect. VI).

II. Issues with Existing Connection Approaches

In this section we identify open challenges in how connections are supported in multi-level models to date with a focus on scalability and expressiveness.

A. Existing Connection Usage Styles

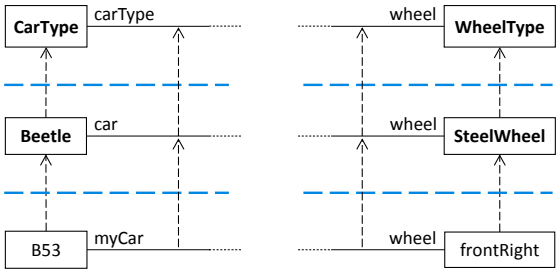

Fig. 1. Two Common Connection Identification Styles

The UML provides two basic ways to adorn connections with names. The first is to label the association itself with its association name, which is often used in combination with a triangular reading-direction indicator [10]. The second is to label the ends of the connection with their so called "association end names" or "role names" in order to define the names by which connectees refer to each other [10]. As discussed earlier, we will use the terms *connection name* and *connection monikers* for connection names and their end names respectively.

The UML places no constraints on how these labeling mechanisms can be used, so for a binary connection any combination is possible, ranging from a name with two monikers to no label at all. In the latter case the names of the connectees need to be sufficient to distinguish one connection from others in a model. The UML furthermore specifies no constraints on label names for instance specifications, e.g. links that are supposed to be instances of connection types such as associations. Even the use of underlining of labels for such instances is optional when the context already indicates that an instance is being labeled [10, p. 84]

As a result, two main styles of using connection names and/or monikers can be observed in modeling practice. The right hand side of Fig. 1 show the widely used approach of using the same moniker (here "wheel") at all levels. The advantage of this approach is that it naturally communicates which connections are meant to be in a type-instance relationship. Its disadvantage is that the required name constancy prevents natural monikers for intra-level navigation. For instance, in Fig. 1 every navigation at any level must use the same wheel moniker whether this is suitable for this particular level or not.

The left hand side of Fig. 1 show the alternative of allowing moniker flexibility, i.e., supporting level-dedicated moniker choices. This allows new monikers to be introduced for every individual connection, and thus intra-level navigation to be expressed using natural names. The downside, however, is that instance-of relationships between connections now have to be indicated in a different manner as it is no longer clear whether there is an intended classification relationship, and if so, between which connections.

These aforementioned "name constancy" vs "flexibilty" styles have been used frequently for both connection names and monikers to date because the models were so small that it was possible to use explicit *instance-of* dependencies as in Fig. 1 or address a lack of name constancy for monikers by choosing name constancy for connection names, or vice versa. However, these approaches do not scale up to larger models with a high proliferation of connections.

B. Connection Type Identification

A model does not need to get much larger than the previous example for it to become a challenge to achieve an instance/type correspondence between connections without resorting to explicit *instance-of* arrows which would in turn lead to an opaque criss-crossing of lines, making this approach unscaleable. Fig. 2 shows a slightly elaborated version of the previous example and demonstrates that resorting to name constancy for connection names in order to establish instance/type connections for connections that use a flexible moniker scheme is not viable in general. In the example, the connection name "has" cannot be used to uniquely identify the type of a connection between instances of Beetle and SteelWheel because there are two connections with the name "has" between Beetle and SteelWheel and while the use of "front" and "rear" prefixes in the monikers at the instance level provide a clue to a human reader, they do not to a tool.

C. Connection Diversification

Figure 2 highlights a further challenge which we refer to as *connection diversification*. Note how the name constancy employed in the right hand side of Fig. 1 is highly suggestive of the modeler's intent to specify at the level of WheelType that navigation to wheels such as frontRight at the bottom level should be supported through the wheel moniker. In other words, the modeler's intent is guaranteeing intra-level

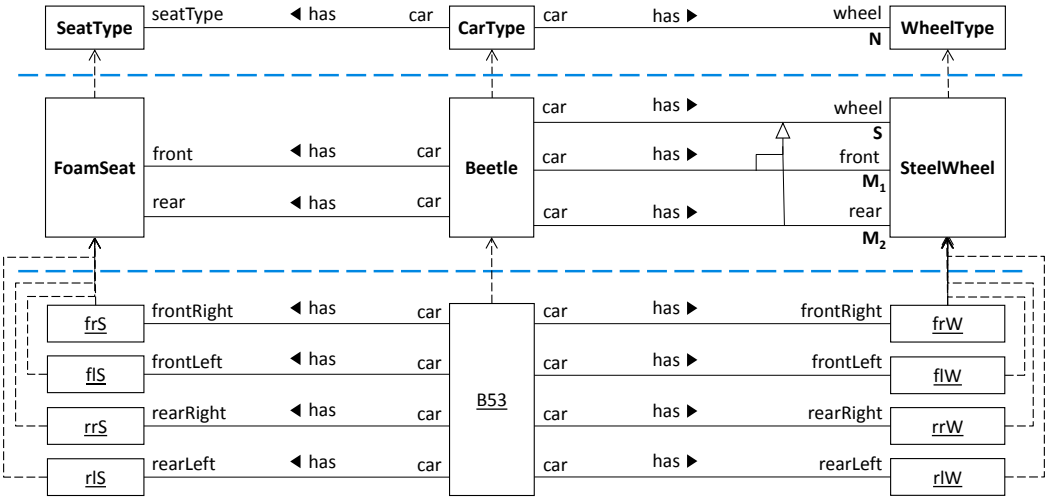

Fig. 2. Connection Proliferation

navigability at the bottom level from two levels above, i.e., achieve *deep characterization* [2].

It is not immediately clear how the same generic bottom-level navigation can be supported in the model shown in Fig. 2. Here, bottom-level navigation to instances of SteelWheel from instances of Beetle is naturally supported through both the front and rear monikers, each referencing a set of two steel wheels respectively when navigating from B53. Yet, due to the diversification of the car–has–wheel connection between CarType and WheelType into both car–has–front and car–has–rear between Beetle and SteelWheel, it is not clear how a modeler can specify that $B53.wheel$ should reference all four instances of SteelWheel unless they add a super-connection car–has–wheel between Beetle and SteelWheel. Note that here we are using "$B53.wheel$" as a shortcut for "evaluating an OCL expression $self.wheel$ in the context of $Beetle$ for the particular $B53$ instance" but also believe that a multi-level version of OCL should support the use of "$B53.wheel$", i.e., allow a specific navigation origin to be nominated.

We refer to a modeling scenario as featuring *connection diversification* when the intent of introducing more specialized connections is not to replace a more general connection but to essentially partition the general connection so that the same instances can be navigated to via a general navigation path (here wheel) or (to subsets of them) through specialized navigation paths (here front & rear).

Clearly, modelers should be able to express whether they intend connection diversification versus other cases like letting general and specialised connection instances coexist, or letting the latter completely replace the former, i.e., disallow general navigation paths altogether. For two-level technology such as the UML, association inheritance (c.f. Fig. 2) and/or association end "{subsets ...}" & "{redefines ...}" constraints have been used to express one of the above modeling alternatives. We aim to allow modelers to make the same choices in a more intuitive manner (c.f. Sect. III).

D. Deep Multiplicities

Another feature of modeling languages that needs to be conservatively embedded in a multi-level classification hierarchies is multiplicity constraints. The simplest approach is to interpret them purely in the traditional, shallow way as only affecting the level immediately below. Under this interpretation, an appropriate multiplicity value for N in Fig. 2 would be two, because the respective connection is intended to be the type for both car–has–front and car–has–rear.

However, it is desirable to allow modelers to specify multiplicity constraints that target levels beyond the immediate level below them (c.f. Nivel [7]). For example, it would be useful at the top level of Fig. 2 to specify that any instance of an instance of CarType needs to be connected to four wheels. In this case, N should be interpreted as a deep multiplicity constraint and should receive the value four. A complete design for connections should address both of these conflicting goals.

E. Connection Inheritance

If N is interpreted as a deep multiplicity constraint (i.e. a constraint on instances of instances of WheelType) more questions about the relationship of N to S, M_1, and M_2 in Fig. 2 arise. In case the modeler intends connection diversification, i.e., a partitioning of the car–has–wheel connection at the top level by the two specialized connections then it would seem appropriate to require that $S = M_1 + M_2$ and $N = S$. It would be possible to enforce such constraints between multiplicity values if it were possible for modelers to unambiguously specify that connection diversification is indeed intended instead of one of the alternatives mentioned in Sect. II-C.

In the following two sections we propose ways of enhancing connections in multi-level modeling to allow users to be more expressive about their exact intent, i.e., specify more precise constraints about when connection instances can be considered well-formed.

III. Unifying Connections Design

In order to attain the best possible modeling pragmatics we considered simplicity, parsimony, expressiveness, stability, and level-agnosticism to be the main forces in the design space. We also aimed at *conservatism*, i.e. achieving compatibility to the UML and other established languages. Without loss of generality, in the following we will consider binary relationships only.

A. Unifying Conceptual Model

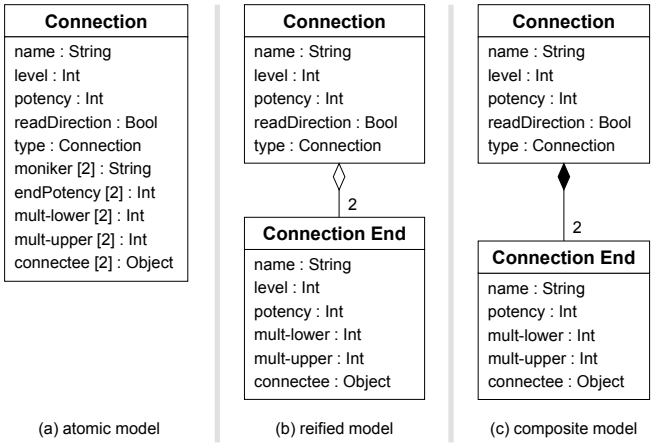

Fig. 3. Possible Conceptual Models for Connections

In order for a notation to be used by modelers to correctly express their modeling intents and allow them to correctly read multi-level models produced by others, modelers need to have an adequate mental model of the conceptual underpinning of the notation. In the following discussion, we are expressly not concerned with representation choices at the implementation level, and only focus on how modelers may best understand and apply the mechanics of connections.

A key question that arises when defining the conceptual model for multi-level connections is whether to regard connection ends as intrinsic features of the connection (*atomic* approach), or whether to treat them as first-class entities in their own right (*reified* approach). These two alternatives are depicted schematically in Figs. 3 (a) & (b).

1) Connection Ends as Attributes: Gutheil et. al. discussed the merits of these approaches in the context of designing a multi-level modeling language and came to the conclusion that the atomic model was the more suitable [5]. According to Gutheil et. al. its advantages include –

a) more natural subsumption of atomic dependencies and association classes.
b) no need to manage the consistency between a connection and its separate ends.
c) more natural error reporting as errors will always be associated with the connection rather than potentially being focused on a connection end only.

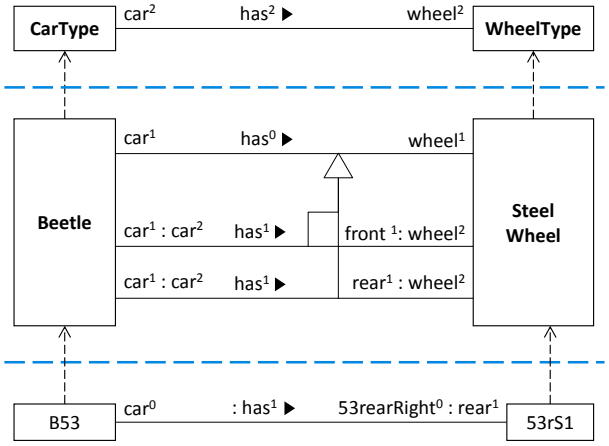

Fig. 4. Connection Specialisation

2) Connection Ends as First-Class Citizens: One of the main disadvantages of the atomic model described above is that it does not lend itself to addressing the connection type specification challenge raised in Sect. II-B by using an instantiation notation "*name*:*type*" in conjunction with monikers. De Lara et al. use this natural style of adding a type specification to the name of a moniker [3, Fig. 6] and thus essentially open up the way for achieving moniker naming flexibility while supporting a means for connection type identification. Fig. 4 shows how B53 can refer to 53rS1 through 53rearRight while the type specification ": rear^1" contributes to identifying car^1–has^1–rear^1 as the connection type at the level above.

The typing fragments ": has^1" and ": rear^1 " for the bottom-most connection in Fig. 4 can be regarded as partial type specifications that in combination identify the intended connection type at the level above. This view is difficult to reconcile with the atomic model that assumes connections to be opaque entities and as such would always require a connection to be fully identified, e.g., as ": car^1–has^1–rear^1", instead of supporting separate references to name and monikers.

Simply adopting the view that connection ends should be regarded as first-class entities, however, would not only imply losing the advantages of the atomic approach (c.f. Sect. III-A1) but would also have undesirable consequences for the meaning of connection end potencies. We argue that in order to be of optimal value in the context of connections, deep monikers should make guarantees about supported navigation paths at levels further down. Just like an association end name "rearWheel" in the UML supports navigation to rear wheel instances at the instance level, a moniker with potency two (e.g., "wheel^2") should define the same navigation support at two levels further down. Yet, this existential semantics for potency values is only associated with fields, i.e., dependent features. In contrast, the semantics for potency values for first-class entities (e.g., clabjects) is just an instantiation depth limit, with no existential guarantees involved.

As a result, an approach that attempts to be parsimonious and unifying with respect to entities and connections must assume connection ends to be dependent features of connec-

tions if monikers are intended to have potency values on their own. Overall, this poses the dilemma of whether to opt for the atomic or the reified model, both having obvious advantages, but being apparently incompatible with each other.

3) Connection Ends as Essential Parts: We posit it is possible to get the "best of both worlds" by adopting a conceptual model where connections are indeed only represented by one clabject, but where this clabject has connection ends within it as internal structure.

Figure 3 (c) shows the respective conceptual model in which the connections are reified but are regarded as strict components of the connection clabject which, according to composition semantics [10] cannot be created or deleted independently. From one perspective such a conceptual model adheres to the atomic model, since there is only one outer clabject per connection, but from another perspective it can be seen as supporting the reified approach and hence adequately support individual referencing of ends.

Note that composite connection ends do not have their own "level" field, as they are always at the same level as the enclosing connection. It is also easy to extend this model to n-ary connections by simply changing the multiplicity on Connection End to "2..*". The details of Fig. 3 are to be considered as indicative only as its purpose is to illustrate major differences between the alternatives.

B. Basic Connection End Semantics

We treat connection ends just like connections and clabjects in that we allow them to have potency values of their own. Hence, a connection has an overall potency, denoted with the potency value associated with the connection name, and its ends have their particular potency values as well.

Fig. 4 shows an example of connection specialization which, like in the UML, allows modelers to introduce special connection cases (here car^1–has^1–front^1 and car^1–has^1–rear^1) while still expressing that they can be understood as flavors of a general connection type (car^1–has^0–wheel^1). Unlike in the UML, however, the use of potency values for car^1–has^0–wheel^1 in Fig. 4, clearly communicates that it is not possible for beetles and steel wheels to participate in any car^1–has^0–wheel^1 connections, as the respective connection type itself has potency zero, expressed by choosing "has^0" as the connector designator.

The potency-one connection ends of car^1–has^0–wheel^1, however, communicate that any concrete specialization of car^1–has^0–wheel^1, such as car^1–has^1–rear^1, will support access to connectees at the level below via names car and front respectively, in addition to the connection names that are introduced by the specialized connection. A modeler using the navigation path car^1–has^0–wheel^1 starting from a Beetle instance, would obtain access to the union of all front and rear wheels connected to the Beetle instance.

Had the "wheel" moniker of car^1–has^0–wheel^1 been specified with potency zero as well, no such general navigability would have been enabled. Steel wheels would only be reachable from beetles through front and rear respectively.

C. Connection Conformance

Both clabjects and connections resolve the traditional type/instance duality by comprising instance as well as type facets [2]. The conformance of a connection instance to a connection type is hence analogous to conformance of entity instances to their entity types [11].

In Fig. 4, connection instance car^0–_–53rearRight^0 conforms to car^1–has^1–rear^1 because –

- the connection instance makes a partial type specification (": has^1") in its "name" compartment that matches the name and potency of the connection type ("has^1"),
- the connection instance end 53rearRight^0 makes a partial type specification (": rear^1") that matches the name and potency of the corresponding connection end of the connection type ("rear^1"),
- the connection instance end car does not make a partial type specification that is incompatible with the name of the corresponding connection end of the connection type ("car^1").

The general principle used to determine conformance is that an instance must satisfy all constraints implied by its type. Hence the absence of a type specification fragment is not an issue, as long as the combination of all partial type specifications still uniquely identifies a connection type (c.f. Sect. III-D).

As to be expected, the incrementation of potency values for connections and their ends works according to standard clabject semantics [2], i.e., analogously to entities and their features respectively. Fig. 4 shows a connection car^2–has^2–wheel^2 at the top level with potency two and a corresponding connection instance car^1–has^1–front^1 with potency one.

Note that the car^1–has^1–front^1 connection is declared both to be an instance of car^2–has^2–wheel^2 at the level above and to be a specialization of connection car^1–has^0–wheel^1 in its own level. Fig. 4 thus demonstrates the occurrence of the powertype pattern [12] for connections rather than for entities. The potency-zero connection car^1–has^0–wheel^1 ensures that car^1–has^1–front^1 and car^1–has^1–rear^1 are partitioning the superset implied by car^1–has^0–wheel^1, thus giving the modeller a means to declare that both "front" and "rear" connection specializations are meant to be a *diversification* of the top-level connection car^2–has^2–wheel^2.

Analogous to clabjects, however, the use of a powertype pattern configuration is not mandatory to support connection diversification. Deep connections such as car^2–has^2–wheel^2 do not need additional supertypes, such as car^1–has^0–wheel^1, in order to characterize connection instances beyond one level boundary. In the example in Fig. 4, a connection instance with the type car^1–has^1–front^1 would support bottom-level navigation using a wheel moniker, even in the absence of the supertype connection, as the former type is declared to be an instance of car^2–has^2–wheel^2. In other words, the use of potency two for car^2–has^2–wheel^2 at the topmost level in the example ensures that the car^0–has^0–wheel^0 access path from beetles to their wheels is available at the bottommost level in addition to the "front" and "right" paths.

D. Connection Identification

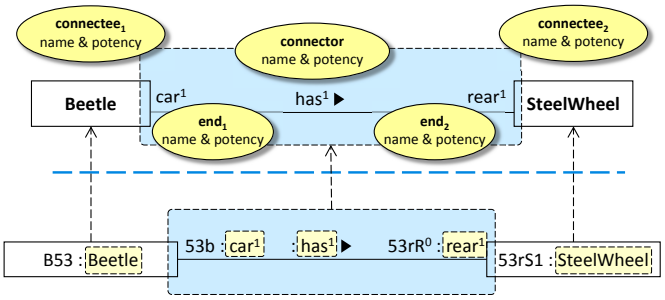

Fig. 5. Connection Designation

As mentioned earlier, using explicit "*instance-of*" dependencies between connection instances and their types is neither practical nor desirable in terms of keeping diagrams uncluttered. Therefore the challenge arises as to how to specify a connection's type without "pointing" at the latter with an explicit "*instance-of*" relationship.

We propose a type specification approach in which the intended connector type is uniquely identified by combining fragments of information from the connector's parts and, if necessary, the names of the connectees. Fig. 5 illustrates the approach by showing a connection instance (big shaded area at the bottom) and its intended connector type (big shaded area at the top). The connection instance uses information from the connection type parts ": Beetle", ": car^1", ": has^1", ": rear^1", and ": SteelWheel" to uniquely specify the connector type car^1–has^1–rear^1 with the connectees Beetle and SteelWheel. In general, a connection designator can include potency values as well as names from these parts.

The modeler should be able to provide only as much of a connection type specification as needed in a particular scenario by supplying a partial type specification only. In some cases it will be sufficient to simply specify the connection type's name (here "has") and potentially its potency in order to uniquely identify it. In general, however, the specification of connection type end names and/or their potencies, and even consideration of the connectee types (here Beetle & SteelWheel) may be necessary to completely disambiguate the intended connection type. Thus, any partial connection type specification involving one or more of the specification fragments highlighted in Fig. 5 is valid as long as in combination they uniquely identify a connection type (c.f. Fig. 6, containing a variety of examples for partial type specifications). There is no need to unconditionally mandate the presence of any of the fragments or assign higher significance to one of them. In practice, however, we expect the quality of error messages by a tool to be improved by introducing a matching heuristic. The goal of such a heuristic would be to increase the likelihood that the fragment an error is reported on is indeed the one that the user would want to change in order to identify an existing connection type.

One approach would be to associate priority values to the type specification fragments in order to achieve matching precedence. For instance, if a connection type's name (here "has") is specified, it could reduce the candidate set of connection types first, before any more fragment matching is undertaken. The tool could thus report non-conformance of a specified connection type end as opposed to matching the end specification with a different connection type first and hence regarding the specified type name to be non-conforming. An alternative matching heuristic would be to select the connection type that yields the largest matching fragment set and report any non-matching fragments as the erroneous ones. Which of these, or another heuristic, is most desirable depends on user expectations and needs to be fine-tuned in future experiments.

Our approach implies that the combination of connection fragments of any connection must be unique within the same level in order to enable textual connection type identification. Strictly speaking, there is no need to enforce such a uniqueness constraint as modelers could still resolve ambiguities by using the explicit *instance-of* dependency connection approach. It may, however, be helpful to modelers to require that connections must be distinguishable by a complete type specification in order to prevent misinterpretation when, for instance, the type level is not shown or *instance-of* dependencies are temporarily filtered out by the tool for clarity of presentation. Future work will show whether uniqueness of complete connection type specifications should be suggested through respective warnings or enforced through respective errors.

E. Connection Navigation

With respect to navigation, connection ends with potency one correspond to association ends in the UML [10]. They enable navigation between instances at the level below. In Fig. 4, for instance, the connection end name rear enables navigation from Beetle instances to their rear wheels. In the example we just have $B53.rear = 53rS1$ but in general the whole set of wheels attached to B53 would be returned. For example the OCL fragment "$self.rear$" in an OCL constraint using Beetle as a context would reference the whole set of rear seats connected to a Beetle instance.

Should the need arise to navigate to a specific rear wheel of the $B53$ instance, it is possible to specify the respective rear wheel moniker, e.g., with $B53.53rearRight$.

Note that an instance such as rrW is accessible to B53 through several aliases, i.e., those connection end names that have potency zero at B53's level. Depending on the potencies of the higher level connection ends, other instances are included as well. For the model in Fig. 6 navigations from B53 using the connection end names "wheel", "rear", and "53rearRight" would respectively yield $\{frW\ flW\ rrW\ rlW\}$, $\{rrW\ rlW\}$, and rrW. The last result is not a set, as the extent of the potency-zero moniker 53rearRight is just an element, as opposed to the sets defined by wheel and rear which both have potency one at the level above.

The approach suggested here naturally supports intra-level navigation at any level. Should the need arise to navigate from Beetle to SteelWheel, e.g., in order to check whether

the supplier information registered at SteelWheel satisfies the requirements specified by the Beetle type, then the expression $Beetle.rear$ can be used.

Navigation expressions as used above may also involve potencies themselves, e.g., as in $Beetle.rear^1$ that references the set of all rear wheels that can be accessed from Beetle, i.e., all rear seats of all Beetle instances. Note that the default value in navigation expressions for connection ends is zero to support the much more frequent case of pure intra-level navigation.

Figure 6 shows a range of examples of using connection potencies. In this context, the expression $B53.(front : wheel^2)$ yields the set containing the two front wheels attached to $B53$, i.e., $\{frW\ flW\}$ (note the disambiguation of "front" to wheels rather than seats by referencing the corresponding moniker of the connection type). These two wheels are recognizable as front wheels since their connection type specifications reference the car^1–hf^1:has^2–front^1 connection from the level above them. The expression $B53.(rear : wheel^2)$ yields $\{rrW\ rlW\}$ because even though the connection end "::wheel^2" of the connection between B53 and rlW does not reference the connection end "rear^1" of its type explicitly, the name of its connection ("hrl^0:hr^1:has^2") ensures that car^1–hr^1:has^2–rear^1 is uniquely identified as the latter's type.

If the car^0–hrl^0:hr^1:has^2–::wheel^2 connection had omitted the "hr^1" component, the identified connection type would have been car^1–has^1–wheel^1 from the level above. The latter is not explicitly visible in the model but exists due to the top level connection car^2–has^2–wheel^2. Due to the fact that this top-level connection has potency two, it is possible for instances two levels down to participate in connections of this kind. Note again that all wheels connected to B53 are reachable via $B53.wheel$. Since both car^1–hf^1–front^1 and car^1–hr^1–rear^1 reference car^2–has^2–wheel^2 as their type, all their respective connection instances are deep instances of car^2–has^2–wheel^2.

Note that it is not necessary for connection car^1–hr^1–rear^1 between Beetle and SteelWheel to declare its "rear^1" moniker to be a specialization of "wheel^1", as it already has a "wheel" alias by virtue of the wheel^2 connection end of its connection type car^2–has^2–wheel^2, just like its connection end sibling front. We only included the specialization as an example of how one could introduce a generalized connection end without utilizing a potency-two connection end at the level above.

We regard the use of "$<$" in "rear$^1 <$ wheel^1" to denote specialization, as a candidate shortcut notation for the visual alternative to explicitly adding a general car^1–hw^1:has^2–wheel^1 connection between Beetle and SteelWheel which car^1–hr^1:has^2–rear^1 would have to specialize from.

F. Connection Multiplicities

As in the UML, connection multiplicities impose constraints on the out-degrees and in-degrees of connection instances between clabject instances that engage in the respective connection. Going beyond Nivel [7], we suggest that modelers should be able to specify multiplicity constraints for multiple levels at the same time.

The top right hand multiplicity constraints in Fig. 6 show how a modeler may stipulate that instances of CarType (such as Beetle) must have instances themselves (such as B53) that must always have exactly four wheels attached to them. The modeler uses a potency value of two for the respective "4" multiplicity constraint. The regular (potency-one) multiplicity constraint of "2" stipulates that instances of CarType must have exactly two "has"-connections to WheelType instances. The model in Fig. 6 satisfies this constraint by establishing the two car^1–hf^1–front^1 and car^1–hr^1–rear^1 connections between Beetle and SteelWheel as instances of car^2–has^2–wheel^2.

IV. Design Discussion

After presenting a design for multi-level connections, we now discuss to what extent the design satisfies the goal of being unifying while maintaining expressiveness.

A. Concept Coverage

Our design is unifying in the sense that it covers a number of traditional language constructs with fewer concepts. For instance, we can accommodate association classes and higher-arity associations by virtue of interpreting connections as clabjects with a special purpose in the same vein as Gutheil et al. [5].

We can furthermore subsume the "instance facet only" nature of type-level links such as UML dependency and specialization relationships by modeling them with potency-zero values for the connection and its monikers.

Qualified associations in the UML can be understood as representing intra-level navigation with object-category specific monikers (e.g., "leftWheel" vs "rightWheel) playing the role of keys.

Establishing a specialization relationship between these specific access paths and a general path provides a name for the overall navigation. The criterion used to partition this general access into multiple qualification subsets may even be explicitly named by a corresponding potency-two type at the level above. The notational convenience of qualified associations in the UML may be retained in potential future developments of our approach but at least there is a way to understand the mechanics of a qualified association within the existing framework without needing to introduce it as a primitive construct.

B. Parsimony

Our design is also unifying in the sense that it uses the same principles for entities (clabjects and their features) as it does for connections (connections proper and their monikers). Once a modeler understands the former, the implications for the latter follow without the need to learn a separate set of rules or mechanisms.

Despite this minimalism, the design provides a high degree of expressiveness. For instance, the potency of monikers can be interpreted as controlling over how many levels further down the moniker is available to support navigability between instances. As a result, the potency value zero excludes any

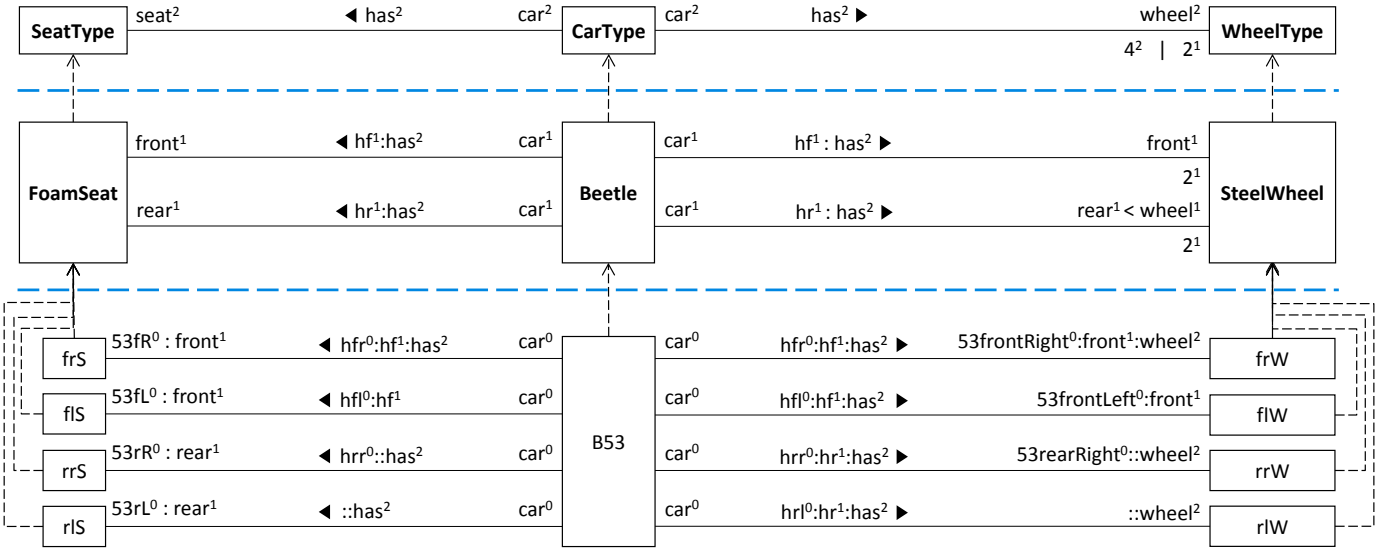

Fig. 6. Connection Navigation

navigability further down and thus amounts to a navigability restriction, analogous to the "cross" notation for association ends in the UML. Intra-level access to the connectee itself (rather than its instances at a certain depth) through a potency-zero moniker is still possible. This is analogous to the way UML navigability restrictions only applies to instances, not to a traversal of the connection at the type level.

C. Expressiveness

Being able to choose the potency values for connections and their monikers independently from each other provides fine-grained but nevertheless intuitive control over model well-formedness. Table I lists the four main distinguishing cases of potency value combinations. Potencies higher than one are not considered, nor cases where monikers have different potency values (c.f. the "M_i" columns in Tab. I). Such mixed cases are obviously allowed and well-defined as the resulting meaning is implied by the semantics of deep instantiation [2], [11].

Note that in the context of connection specialization, potency-zero super-connections still provide an abstract view on connection instances of sub-connections. The potency of a super-connection provides control over whether it only has indirect instances (i.e., is a strict union of the subsets implied by its subconnections, (c.f. rows 1 & 2 in Tab. I) or has direct instances as well (c.f. rows 3 & 4 in Tab. I).

M_1	C	M_2	**Connection kind**
0	0	0	pure intra-level connection, analogous to a UML link
1	0	1	abstract connection with a "derived union" semantics
0	1	0	connection with no navigability
1	1	1	regular connection

TABLE I
POTENCY VALUE COMBINATIONS AND THEIR MEANING

The previously discussed case of *connection diversification* is represented by row 2 in Tab. I. It supports generic navigability to entities (e.g., using "frontWheel") through potency-one monikers, despite the fact that actual connection instances use diverse monikers such as "frontLeftW" vs "frontRightW". Having such diverse monikers can be important in order to model structure or rule out illegal combinations such as mounting a directional tire to the wrong side.

Row 3 in Tab. I may seem like a pathological case as it defines connection instances that do not support any navigability at all. While actual usage of this case in practice may be rare, the UML infrastructure specification explicitly allows and discusses it [13, Fig. 11.8].

D. Conservatism

We maintained UML backward-compatibility in three ways:

1) A multi-level model using only the two traditional type and instance levels with no use of extra features such as deep characterization, can be read like regular UML notation but with notational simplifications to support level-agnosticism and without UML constructs that have become redundant due to unification.
2) Our support for navigation between entities is richer than that of the UML but is a conservative extension of it. In other words, current conventions, e.g., assumed by the OCL [14], still hold and have only been augmented with additional capabilities.
3) We require no limitations that would reduce the current expressiveness of the UML. While initial investigations are promising, future work will have to demonstrate that certain UML constructs such as navigability restrictions on association ends have been truly made redundant in each conceivable case.

The proposed approach is also compatible with various kinds of connection specification styles currently used in multi-level modeling, namely

a) the connection-only style [2],
b) the connection-only style with type specifications [15],
c) the moniker-only style [6],
d) the moniker-only style with type specifications [3], and
e) any combination of the above.

This is important because it means that the existing multi-level models that use any of the above styles remain valid under the proposed approach. It follows that the majority of multi-level model tools currently support a valid subset of the proposed approach and could be conservatively enhanced to full support.

V. Related Work

In the following we compare our proposal to alternatives that considered connections in multi-level modeling at least to the extent of assigning potencies to connections.

Kennel [6], Melanee [16], MetaDepth [17], and the DPF modeling environment [15] all support potency values for connections, but do not allow monikers to have their own independent potency values. They therefore do not afford modelers with the same expressive power to document specific scenarios, such as *connection diversification*.

Kennel advocated the principle of moniker constancy in order to provide a systematic and efficient basis for exploratory as well as constructive modeling [6]. The approach hence features a simpler labeling scheme than the one advocated in this paper, but can sometimes result in non-intuitive labels as they cannot be level-specific.

MetaDepth is noteworthy in that it assumes reified connection ends and incorporates the "monikerName : Type" notation [3] that we found useful to obtain a connection identification approach that is both intuitive and scales.

The Nivel language [7] does not support moniker potency either, but supports deep multiplicities (i.e., multiplicity constraints with a potency value). In contrast to our proposal, there is only one constraint per connection end, though, so the modeler's control over lower levels is not as comprehensive. On the other hand, Nivel gives explicit consideration to roles, i.e., recognizes that participants in relationships can be regarded as players that assume a certain role in the relationship [18]. We designed our approach to be extensible in this direction but defer any further elaboration to future work.

The UML features association end subsetting, association end redefinition, derived unions, and association end navigability control which, in combination, can also be used to express modeler intent with respect to *connection diversification* [10]. In our approach negative navigability control (cross notation at association ends) corresponds to potency-zero monikers. Positive navigability in the UML (arrow notation at association ends) does not permit access but emphasizes that the latter must be efficient [10]. We do not consider access efficiency as a concern.

We surmise that one of the reasons that the other three of the aforementioned UML mechanisms do not appear to be widely used in common modeling is that it is challenging to understand the subtleties of their semantic overlap and their intended interaction with each other. The latter challenge was reflected by UML specifications themselves as they declared the interaction of these features with association specialization as a "semantic variation point", i.e. as undefined. This semantic variation point has only recently been removed. We posit that our approach using potencies for both connections and their monikers supports the case of connection diversification with a fewer number of concepts that are intuitively understood by modelers familiar with the *deep modeling* (i.e., clabject and potency-based) approach to multi-level modeling [1]. For example, the UML's derived union mechanism is a way of expressing that a super-connection is fully partitioned by a number of sub-connections, i.e., has no direct instances of its own. This can be achieved in our approach by assigning potency zero to the connection. Association end redefinition appears to be a way of declaring that the sub-connections replace super-connections. The equivalent effect can be achieved in our approach by introducing a new moniker name in the sub-connection (which must have potency of at least one to be instantiatable and may introduce further refinements, e.g., regarding multiplicity) and declaring it (either textually or visually) to be a subtype of the redefined moniker. Whether or not the resulting redefinition is exclusive, i.e., prevents the sub-entities to participate in super-connections, is controlled via the potency of the super-connection. Independently of the actual mappings between our potency-based approach and the UML's various constructs, it can be observed that the meaning of any model constructed based on our proposal immediately and unambiguously follows from the semantics of deep instantiation, without the need to clarify and define the interaction of several semantically overlapping features.

Lars and Gogolla propose an endogenous metamodeling approach to formally define the semantics of UML and include UML subsetting and derived union constraints in their examples [19]. We interpret their virtual links as manifestations of alternative navigation paths, i.e., as representing the fact that some specialized connections are also indirect instances of general connection types (as well as being direct instances of their specialized connection types). We maintain that explaining the semantics of the scenarios they cover would have been easier for them if they only had to formalize clabjects and their instantiation semantics, instead of the variety of UML constructs needed to achieve the same expressiveness.

The Deep OCL dialect [20] supports navigation over deep monikers using "*level casts*". For instance, it is possible to access all wheels of B53 in Fig. 6 by explicitly casting the type of B53 to reference its deep type CarType in order to enable the use of the wheel moniker. In our approach the wheel moniker is implicitly available to B53. The advantage of explicit level casts is that they document within navigation expressions which (deep) type introduced the utilized navigation path. The disadvantage are more verbose navigation expressions. The best of both worlds could be achieved by allowing level casts for documentation purposes but not requiring them.

VI. Conclusion

In this paper we identified limitations in terms of scalability and expressiveness in existing multi-level connection approaches. We then presented a range of ideas to address these limitations, achieving unification along several aspects. We first proposed a conceptual model of connections that unifies both the individually useful but apparently incompatible views of "atomic connections" versus "reified connection ends". Achieving an accurate model in which connection ends are recognized as parts that are separate from but unconditionally dependent on connections was critical in order to allow modelers to have correct expectations about connection semantics. To this end, we clarified the different semantics of potency for first-class entities versus dependent features. Our proposed composite connection end model naturally supports a novel connection type identification style that scales in the presence of connection proliferation.

We expect modelers familiar with deep modeling [1] to quickly become competent users of the proposed approach due to the second unifying aspect of our work: We ensured that the semantics of connections and their monikers is consistent with clabjects and their features respectively, i.e., that entities and connections are viewed in a unifying manner. We did not follow this approach dogmatically, however, but judged its merit by evaluating whether the increased complexity compared to earlier connection approaches is matched by a respective increase in expressiveness. One of our key findings was that important modeling intents such as *connection diversification* can be naturally supported by using connection specialization in conjunction with moniker potencies. We showed that the powertype pattern known for entities [12] also occurs in the context of connection diversification and observed that the use of deep characterization for a top-level connection allows the super-connection type in the pattern to become optional.

The aforementioned expressiveness of our design leads to the third unifying aspect of this paper which is the potential for subsuming a number of known diverse modeling constructs with the flexible combination of a few basic ideas already known from *deep modeling* [1]. Building on and improving on earlier success in the reduction of separate modeling construct required [5], we have earmarked constructs like qualified associations, association end subsetting, derived unions, and association end navigability control as candidates for being demonstrated as being redundant within the proposed design. Further work will be necessary to provide a definitive answer to the question of which traditional constructs are truly expendable but the potential of mirroring the success story of subsuming stereotypes, powertypes, and tagged values with the notion of clabjects with potency [8] is clearly present.

Future work is required for determining currently unresolved details such as appropriate default rules that yield the most natural meaning when certain features (e.g. names, potencies, multiplicities, etc.) have not been provided by the modeler. We expect respective choices to have an impact on model readability and even model extensibility. In a similar vein, decisions relevant for tool builders, such as which heuristics to apply when reporting model inconsistencies, are yet to be made.

We furthermore plan to investigate a proper integration of roles [18]. The UML terminology for monikers, "role names" already suggests that connection ends could serve a double purpose of enabling navigation paths as well as referencing role definitions. The latter may specify a contract for player types and may even enhance the latter's protocol and behavior. We hope the large number of interesting questions opened up by our proposal will help researchers to make multi-level modeling a useful tool for mainstream modeling challenges.

References

[1] C. Atkinson, R. Gerbig, and T. Kühne, "Comparing multi-level modeling approaches," *Proceedings of MULTI 2014*, vol. CEUR-WS.org/Vol-1286, pp. 53–61, 2014.

[2] C. Atkinson and T. Kühne, "The essence of multilevel metamodeling," in *The UML. Modeling Languages, Concepts, and Tools*, ser. LNCS, M. Gogolla and C. Kobryn, Eds. Springer, 2001, vol. 2185, pp. 19–33.

[3] J. D. Lara, E. Guerra, and J. S. Cuadrado, "When and how to use multilevel modelling," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 2, pp. 12:1–12:46, Dec. 2014.

[4] T. Kühne and D. Schreiber, "Can programming be liberated from the two-level style: Multi-level programming with deepjava," in *22nd ACM SIGPLAN OOPSLA Proceedings*. ACM, 2007, pp. 229–244.

[5] M. Gutheil, B. Kennel, and C. Atkinson, "A systematic approach to connectors in a multi-level modeling environment," in *MoDELS 2008*, 2008, pp. 843–857.

[6] B. Kennel, "A unified framework for multi-level modeling," Ph.D. dissertation, University of Mannheim, 2012.

[7] T. Asikainen and T. Männistö, "Nivel: A metamodelling language with a formal semantics," *Software & Systems Modeling*, vol. 8, no. 4, pp. 521–549, 2009.

[8] C. Atkinson and T. Kühne, "Reducing accidental complexity in domain models," *Software & Systems Modeling*, vol. 7, no. 3, pp. 345–359, 2008.

[9] F. Steimann, "Formale Modellierung mit Rollen," Habilitationsschrift (Universität Hannover), 2000.

[10] OMG, "OMG Unified Modeling Language™, Superstructure Version 2.4.1," http://www.omg.org/spec/UML/2.4.1, 2011.

[11] T. Kühne and F. Steimann, "Tiefe charakterisierung," in *Modellierung 2004, Proceedings zur Tagung, 23.-26. März 2004, Marburg*, 2004, pp. 109–119.

[12] J. Odell, "Power types," *Journal of Object-Oriented Programming*, vol. 7, no. 2, pp. 8–12, May 1994.

[13] OMG, "Omg unified modeling language™, infrastructure version 2.4.1," http://www.omg.org/spec/UML/2.4.1, 2011.

[14] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.

[15] A. Rossini, "Diagram predicate framework meets model versioning and deep metamodelling," Ph.D. dissertation, University of Bergen, 2011.

[16] C. Atkinson and R. Gerbig, "Melanie: Multi-level modeling and ontology engineering environment," in *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*, ser. MW '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:2.

[17] J. de Lara and E. Guerra, "Deep meta-modelling with metadepth," in *Proceedings of the 48th TOOLS conference*, ser. LNCS. Springer, 2010, pp. 1–20.

[18] F. Steimann, "A radical revision of UML's role concept," in *Proceedings of UML 2000 - Third International Conference, York, UK, October 2000*, ser. LNCS, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 194–209.

[19] L. Hamann and M. Gogolla, "Endogenous metamodeling semantics for structural uml 2 concepts," in *Model-Driven Engineering Languages and Systems*, ser. LNCS, A. Moreira, B. Schtz, J. Gray, A. Vallecillo, and P. Clarke, Eds., vol. 8107. Springer, 2013, pp. 488–504.

[20] D. Kantner, "Specification and implementation of a deep ocl dialect," Master's thesis, University of Mannheim, 2014.