Designing Domain Specific Languages – A Craftsman's Approach for the Railway Domain Using CASL

Phillip James[1], Alexander Knapp[2], Till Mossakowski[3], and Markus Roggenbach[1]

[1] Swansea University, UK
[2] Universität Augsburg, Germany
[3] DFKI GmbH Bremen, Germany

Abstract. Domain modelling based on UML Class Diagrams is an established industrial practice. In the context of the Railway industry, we show how to utilize such diagrams for verification. This involves the translation of UML Class Diagrams into the algebraic specification language CASL. To this end, we define new Class Diagram institutions and provide suitable institution comorphisms.

1 Introduction

UML Class Diagrams [24] are industrially accepted for modelling a variety of systems across numerous domains. Often they are used to describe all elements and relationships occurring within a domain. As such, a UML Class Diagram (CD) can be thought of as describing a domain specific language (DSL). A typical example of such an endeavour is given by the Data Model [13] of our research partner Invensys Rail which aims to describe all elements within the railway domain.

Classically, DSLs [19,9] allow even non-experts to create programs or specifications. In the context of programming, additional motivation for DSLs is improved tool support along with ease of use, better readability, and increased productivity. James and Roggenbach [15] have demonstrated an approach where formal DSLs within the railway domain aid verification in the context of algebraic specification using CASL [23].

Here, we describe how to utilize industrial DSLs from the Railway domain, formulated as UML CDs, for verification. As UML CDs only capture the static system aspects, we make the realistic assumption that the CD is accompanied with some natural language specification describing the dynamic system aspects. For Railways this situation is given thanks to generally accepted standard literature, e.g., [16].

On the technical side, our construction is based on institution theory: to capture realistic class diagrams, we extend the UML CD institution by Cengarle and Knapp [6] with numerous concepts typically appearing in applications. UML CDs describe invariants that hold during all system runs, however, do not offer the possibility of capturing a system's dynamics. To this end, we employ MODALCASL [21] as a general framework for specifying Kripke-structures. We give an institution comorphism from UML CD to MODALCASL. Part of this mapping is a general construction, namely that of a Pointed Powerset Institution, which factors out a general construction principle necessary for connecting UML CDs with an arbitrary institution capturing system dynamics. In MODALCASL, we then model system dynamics according to the natural language specification describing the dynamic system aspects. Finally, for the sake of better proof support, we use an already established comorphism from ModalCasl to Casl [23]. Overall, this approach allows us to directly import DSL specifications from industry into the verification framework proposed by James and Roggenbach [15].

Related work. Our work closely relates to the various approaches that utilize UML as a graphical frontend for formal methods. In this respect, we follow the approach of Cengarle and Knapp [6]: a UML diagram type can be described in its "natural" semantics; its relations to other UML diagram types is expressed by appropriate translations. Overall, this results in a compositional semantics for UML: each diagram type is treated individually. It also separates concerns: first a semantics is given, then a translation is defined. This differs from monolithic constructions, which first select a fixed, usually small number of UML diagrams and then give a semantics by translation into an established formalism. A first such attempt with Casl as the underlying formalism was given in [12]. Yet another example of this second approach is the UML-RSDS method [17]. UML-RSDS translates specifications consisting of CDs, state machines and OCL into B. CD annotations in the form of stereotypes steer the translation. Lano et al. [17] give a railway example, however, not to the extent of defining a DSL. Another example of this kind is the approach taken within the Iness project [7]. The Iness project defines a DSL whose components are mapped into xUML. These are then translated to Promela in order to verify railway systems with the SPIN model checker. Besides the different approach to semantics, our construction allows for theorem proving technology rather than for model checking. A third approach is given by Meng and Aichernig [18], who define various semantics for CDs depending on their use in software development. Their overall approach is of a co-algebraic nature. The most abstract level, defined as the "object type" semantics, is close to the view we take on CDs. The more concrete levels equip objects with a (hidden) state and a transition structure, visibility tags for attributes and methods, or OCL constraints.

Concerning DSL design, Bjørner [4] has studied the endeavour of domain engineering, where he takes the specification languages RSL as the semantic basis. Here, we develop a methodology that designs formal DSLs based upon the outcome of domain engineering undertaken by industry. Therefore, as the domain analysis is undertaken by the domain experts themselves, the resulting DSL captures concepts and their relations "correctly" for people working within the domain. Using the ASF-SDF environment, Andova et al. [1] develop their "Simple Language of Communicating Objects" into a formal DSL by giving it an operational semantics. Our semantics is axiomatic in style, where concepts have a loose semantics.

Organisation. We first review the notion of a DSL, present an established example from academia as a running example, and briefly comment on the Data Model from our industrial research partner Invensys Rail. In Section 3, we recall notions from institution theory and present the Casl and ModalCasl institutions as well as a comorphism between them. Our new institution for UML CDs is given in Section 4. There, we also embed the UML CD institution into the ModalCasl institution via a comorphism. Finally, using the above constructions, we demonstrate that these techniques allow for automated verification in the Railway Domain.

2 Domain Specific Languages

Domain Specific Languages are languages designed and tailored for a specific application area. Examples include Risla [2] for financial products, Hancock [5] for processing large scale customer data concerning telephone calls, and HTML for web-page design. The defining feature of DSLs is exactly the fact that they provide notations and constructs tailored to a particular domain [9,19]. Usually, DSLs are languages designed for programming. Here we consider their use for algebraic specification. Their advantages over general purpose languages concern *readability* (notations as in the application domain), *productivity* (in design, implementation and maintenance of the software life cycle, thanks to ease of notation), and *re-use* (via domain specific components).

The railway community has developed several informal DSLs (usually company specific) for describing elements within the domain. For example, Invensys Rail have created a detailed DSL for the railway domain using UML and natural language [13]. This data model describes a DSL using UML and accompanying explanations in plain English. It is split into various layers, each describing certain aspects such as signalling or geographical information. These layers are mutually dependent. The document gives details of around 150 classes with a textual explanation of each. This paper describes the foundations of how to turn such a rich, informal DSL into a formal one. However, for ease of illustration we consider a simpler DSL from the academic context.

2.1 Bjørner's Railway DSL

The process of identifying, classifying and precisely defining the elements of a domain has been coined as "Domain Engineering" by Dines Bjørner [4]. Bjørner's classification, i.e., DSL [3], for the Railway domain is illustrated using a UML CD in Fig. 1.It contains the following elements: *Classes*, represented by a box, e.g. Net, Unit, Station etc. These represent concepts in the railway domain. *Properties* are listed inside a class, e.g. id : UID in the class Net expresses that all Nets have an identifier of type UID. *Generalisations*, represented by a unfilled arrow head, e.g. Point and Linear are generalisations of Unit. *Associations*, represented by a line/arrow between two classes, e.g. the has link between Unit and Connector. Associations can have direction, and also multiplicities associated with them. The multiplicities on the has association between Unit and Connector can be read as: "One Unit has two or more connectors". *Compositions*, represented by a filled diamond, e.g. the hasLine composition for Net and Line, tell us that one class "is made up of" another class. Compositions can also have multiplicities. Finally, *operations* are also represented inside a class, for example the isOpen operation of type Boolean inside the Route class. Here, we note that the query operations we consider can be seen as parametrised properties. We do not treat operations that can modify the state of an object. Primitive types such as Boolean and structured types such as List, Set, and Pair are considered to be predefined.

To this "pure" UML class diagram a so-called *stereotype*, «dynamic», has been added. In Fig. 1 this appears for the association stateAt and the two operations isClosedAt and isOpen. All other elements are considered to be «rigid». These domain-specific stereotypes are intended to make clear which parts of an object structure complying to the

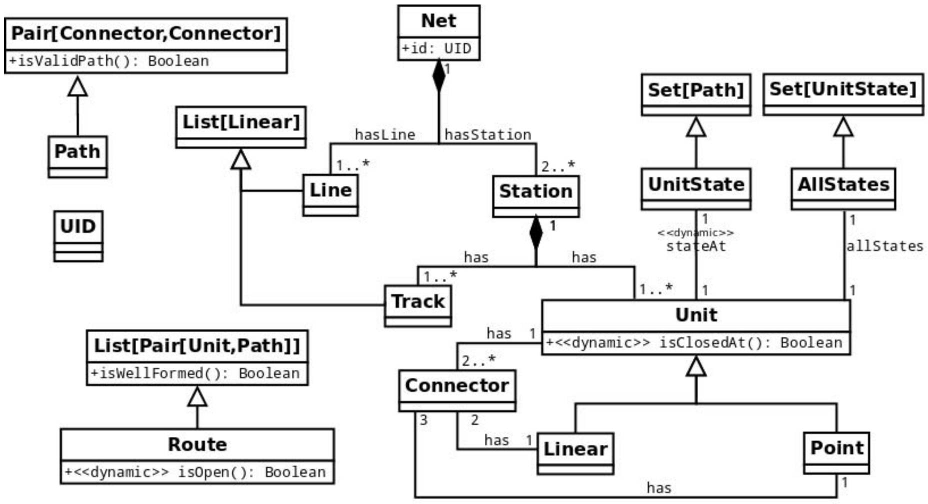


Fig. 1. A UML class diagram for Bjørner's DSL

class diagram can change over time, i.e., are «dynamic» like stateAt, and which parts have to be kept fixed over time, i.e., are «rigid», e.g., the objects of Net, Station, Line, etc.

From a modelling point of view, Bjørner's DSL prescribes that a railway is a Net, built from Stations that are connected via Lines. A station can have a complex, though fixed, structure, including Tracks, switch Points and LinearUnits. Tracks and Lines can contain LinearUnits only. All Units are attached together via Connectors. Such railway Nets gain dynamics by attaching a UnitState to each Unit, see [3]. Each unit can be in one of several states denoted using a Path. Such a Path indicates in which direction a train can pass. However, trains are not an explicit part of the DSL. Instead, Bjørner describes the concept of a Route, as a list of connected units. Finally, Bjørner gives a series of properties using a narrative description. An example of such a narrative is: "*A route is considered to be open if all units within the* Route *are open for the correct connected* Path*s across them.*" Such properties are not expressible purely using CD's. Along with these concepts, Bjørner stipulates further various well-formedness conditions on such a collection of Nets also using natural language.

3 Institutions and Institution Comorphisms

Following [20], we briefly recall the definition of institutions [11] and institution comorphisms: An institution $\mathscr{I} = (\mathrm{Sig}^{\mathscr{I}}, Sen^{\mathscr{I}}, \mathrm{Mod}^{\mathscr{I}}, \models^{\mathscr{I}})$ consists of (i) a category of *signatures* $\mathrm{Sig}^{\mathscr{I}}$; (ii) a *sentence functor* $Sen^{\mathscr{I}} : \mathrm{Sig}^{\mathscr{I}} \to \mathrm{Set}$, where Set is the category of sets; (iii) a contra-variant *model functor* $\mathrm{Mod}^{\mathscr{I}} : (\mathrm{Sig}^{\mathscr{I}})^{\mathrm{op}} \to \mathrm{Cat}$, where Cat is the category of categories; and (iv) a family of *satisfaction relations* $\models^{\mathscr{I}}_{\Sigma} \subseteq |\mathrm{Mod}^{\mathscr{I}}(\Sigma)| \times Sen^{\mathscr{I}}(\Sigma)$ indexed over $\Sigma \in |\mathrm{Sig}^{\mathscr{I}}|$, such that the following *satisfaction condition* holds for every signature morphism $\sigma : \Sigma \to \Sigma'$ in $\mathrm{Sig}^{\mathscr{I}}$, every sentence $\varphi \in Sen^{\mathscr{I}}(\Sigma)$ and for every Σ'-model $M' \in |\mathrm{Mod}^{\mathscr{I}}(\Sigma')|$:

$$\mathrm{Mod}^{\mathscr{I}}(\sigma)(M') \models^{\mathscr{I}}_{\Sigma} \varphi \Leftrightarrow M' \models^{\mathscr{I}}_{\Sigma'} \mathit{Sen}^{\mathscr{I}}(\sigma)(\varphi) .$$

$\mathrm{Mod}^{\mathscr{I}}(\sigma)$ is called the *reduct* functor, $\mathit{Sen}^{\mathscr{I}}(\sigma)$ the *translation* function.

To an institution $\mathscr{I}$ there is associated a category $\mathrm{Pres}^{\mathscr{I}}$ of *theory presentations*, where presentations are pairs (Σ, A) with $A \subseteq \mathit{Sen}^{\mathscr{I}}(\Sigma)$, and presentation morphisms $\theta : (\Sigma, A) \to (\Sigma', A')$ are signature morphisms $\theta : \Sigma \to \Sigma'$ such that $A' \models^{\mathscr{I}}_{\Sigma'} \mathit{Sen}^{\mathscr{I}}(\theta)(A)$, where the satisfaction relation is extended to a semantic consequence relation between sets of sentences in the usual way. There is an obvious functor $\mathrm{Sig} : \mathrm{Pres}^{\mathscr{I}} \to \mathrm{Sig}^{\mathscr{I}}$ defined on objects by the equation $\mathrm{Sig}(\Sigma, A) = \Sigma$. By abuse of notation we extend $\mathit{Sen}^{\mathscr{I}}$ and $\mathrm{Mod}^{\mathscr{I}}$ to start from $\mathrm{Pres}^{\mathscr{I}}$ setting $\mathit{Sen}^{\mathscr{I}}(\Sigma, A) = \mathit{Sen}^{\mathscr{I}}(\Sigma)$ and letting $\mathrm{Mod}^{\mathscr{I}}(\Sigma, A)$ be the full sub-category of $\mathrm{Mod}^{\mathscr{I}}(\Sigma)$ induced by the class of models M satisfying A.

For relating institutions in a semantics preserving way, we consider simple institution comorphisms. Given institutions $\mathscr{I}$ and $\mathscr{J}$, a *simple institution comorphism* $\mu = (\Phi, \alpha, \beta) : \mathscr{I} \to \mathscr{J}$ consists of (i) a functor $\Phi : \mathrm{Sig}^{\mathscr{I}} \to \mathrm{Pres}^{\mathscr{J}}$; (ii) a natural transformation $\alpha : \mathit{Sen}^{\mathscr{I}} \dot{\to} \mathit{Sen}^{\mathscr{J}} \circ \Phi$; and (iii) a natural transformation $\beta : \mathrm{Mod}^{\mathscr{J}} \circ \Phi^{\mathrm{op}} \dot{\to} \mathrm{Mod}^{\mathscr{I}}$, such that the following *representation condition* is satisfied for all $\Sigma \in |\mathrm{Sig}^{\mathscr{I}}|$, $M' \in |\mathrm{Mod}^{\mathscr{J}}(\Phi(\Sigma))|$, and $\varphi \in \mathit{Sen}^{\mathscr{I}}(\Sigma)$:

$$M' \models^{\mathscr{J}}_{\mathrm{Sig}(\Phi(\Sigma))} \alpha_{\Sigma}(\varphi) \Leftrightarrow \beta_{\Sigma}(M') \models^{\mathscr{I}}_{\Sigma} \varphi .$$

3.1 CASL

The algebraic specification language CASL [23] has many-sorted first order logic with equality as its underlying institution. A CASL *signature* $\Sigma = (S, TF, PF, P)$ consists of a set S of sort symbols, a family of sets TF of total function symbols, a family of sets PF of partial function symbols, and a family of sets P of predicate symbols. Index sets of these families are the possible arities of the function symbols and predicate symbols. CASL *sentences* are the standard first order formulae, including strong equality $t = t'$, existential equality $t \overset{e}{=} t'$, and definedness of terms $\mathit{def}(t)$, where t, t' are terms. CASL *models* M interpret sort symbols s with a non empty carrier set s_M, function symbols $f \in PF \cup TF$ with a function f_M of suitable arity, f_M being total for $f \in TF$, and predicate symbols $p \in P$ with a relation p_M of suitable type. CASL *satisfaction* is as is standard for first order logic, where CASL semantics is strict, i.e., an undefined term in a formula yields that the formula does not hold. For details see [23].

On top of the many sorted CASL institution, CASL subsorting is defined: *Subsorted* CASL-*Signatures* $\Sigma = (S, TF, PF, P, \leq)$ are CASL signatures with an additional subsort relation $\leq \subseteq S \times S$. This subsort relation is required to be reflexive and transitive. There is overloading of function symbols and predicate symbols. Each subsorted signature $\Sigma = (S, TF, PF, P, \leq)$ is associated with a many-sorted signature $\Sigma^{\sharp}$ extending (S, TF, PF, P) for each pair of sorts $s \leq s'$ by a total embedding operation $\mathit{inj}_{s,s'}$ (from s into s'), a partial projection $\mathit{inj}_{s',s}$ operation (from s' onto s), and a membership predicate (testing whether values in s' are embeddings of values in s). *Sentences* of subsorted CASL are the ordinary many-sorted sentences over the associated signature.

Models of subsorted CASL are the ordinary many-sorted models over the associated signature that satisfy a list of properties w.r.t. the embeddings, projections, membership, and overloading. *Satisfaction* is as for many sorted CASL. For details see [23].

The datatypes considered to be predefined for class diagrams have been captured in the CASL library of standard datatypes [23].

3.2 MODALCASL

MODALCASL [21] extends CASL by modal logic (see e.g. [10]), providing a multi-modal first-order logic with partiality and subsorting. *A* MODALCASL *signature* consists of a CASL signature, a predicate on the operation and predicate symbols of the CASL signature, marking some of them as *rigid* (the others are called flexible), a set of *modalities*, including the special modality ϵ, and a subset of the sort set of the CASL signature, called the *set of modality sorts*. *A* MODALCASL *model* M consists of a set of worlds W, one of which is marked as the initial one[1], and for each world $w \in W$ there is a CASL model M_w, such that carrier sets and the interpretation of rigid operation and predicate symbols are the same for all M_w. A model M has, for each modality, and for each carrier element of each modality sort, a binary accessibility relation on W, which is preserved under the embedding of a carrier element along a subsort injection. MODALCASL *sentences* are built like CASL sentences, with the following modalities as new sentence building (unary prefix) connectives: $\langle m \rangle$, $[m]$ where m is a modality or a term of a modality sort. If $m = \epsilon$, m is omitted, and we obtain the usual (mono-modal) notation $\langle\rangle\varphi$ and $[]\varphi$. *Satisfaction* is based on the definition of satisfaction in CASL, while modalities are interpreted with Kripke semantics in the following sense: $\langle m \rangle\varphi$ means that φ holds in some 1-step m-reachable world, $[m]\varphi$ means that φ holds in all 1-step m-reachable worlds.

A simple institution comorphism from MODALCASL *to* CASL. *Signature translation:* Sorts are unchanged, but a sort w (for "worlds") is added, as well as a constant $\mathit{init} : w$ and a binary relation $R : w \times w$. Rigid operation and predicate symbols are kept unchanged. For flexible operation and predicate symbols, w is added as an extra argument. *Sentence translation:* Sentences are translated according to the standard translation. *Model translation:* A CASL model is turned into a MODALCASL by keeping the carrier sets (note that MODALCASL enforces constant domains) as well as the rigid operations and predicates. The set of worlds and the accessibility relation are obtained by the interpretation of w and R respectively. The flexible operations and predicates are obtained by using the current world as the extra argument of the CASL operation respectively predicate.

4 Institutions for (Stereotyped) UML Class Diagrams

We first present an institution for proper UML class diagrams as used in Fig. 1, based on work by Cengarle and Knapp [6]. We then capture the meaning of the stereotype

[1] This differs from the original MODALCASL design which refrains from having an initial world.

«dynamic» by a simple general construction on institutions. Finally, we represent this institution in ModalCasl. Technical details and proofs can be found in the accompanying technical report [14].

4.1 An Institution for UML Class Diagrams

The UML class diagram institution is constructed as follows: The signatures capture all classes, relationships, and features of a UML class diagram. The models comprise all objects and links between objects that comply to the UML class diagram. The sentences describe the multiplicities of the associations and compositions. In order to provide generic access to primitive types, like Boolean, and type formers like List, we treat these as built-ins with a standard meaning; all other classes are assumed to be inhabited, i.e., to contain at least one object, like $null$.

Class nets — Signatures. The signatures of the UML class diagram institution are given by *class nets* $\Sigma = ((C, \leq_C), K, P, M, A)$ where $(C, \leq_C)$ describes the *class hierarchy* as a partial order which is closed w.r.t. to the built-in type Boolean (i.e., Boolean $\in C$), and "downwards" closed w.r.t. the unary built-in type formers List and Set, and the binary built-in type former Pair (i.e., if $\mathsf{List}[c] \in C$ or $\mathsf{Set}[c] \in C$, then $c \in C$; and if $\mathsf{Pair}[c_1, c_2] \in C$, then $c_1, c_2 \in C$); K fixes the *instance specifications declarations* of the form $k : c$ with $c \in C$; P contains the *property declarations* of the form $c.p(x_1 : c_1, \ldots, x_n : c_n) : c'$ (where we drop the parentheses if the property has no arguments); M gives the *composition declarations* of the form $c{\blacklozenge}r : c'$; and A comprises the *association declarations* of the form $a(r_1 : c_1, \ldots, r_n : c_n)$. We require several conditions in order to avoid super-types of built-in types and type formers (e.g., forbidding Boolean $\leq c$ for all $c \neq$ Boolean), overriding of declarations, and also that the composition declarations are cycle-free: if $c_1{\blacklozenge}r_1 : c_2, \ldots, c_n{\blacklozenge}r_n : c_{n+1} \in M$, then $c_{n+1} \neq c_1$.

Example 1. The UML class diagram in Fig. 1 leads to the following class net:

Classes: $\{\mathsf{Net}, \mathsf{Station}, \mathsf{Line}, \mathsf{Unit}, \ldots, \mathsf{Pair}[\mathsf{Unit}, \mathsf{Path}], \mathsf{List}[\mathsf{Pair}[\mathsf{Unit}, \mathsf{Path}]]\}$

Generalisations: $\mathsf{Point} \leq \mathsf{Unit}, \mathsf{Linear} \leq \mathsf{Unit}, \ldots, \mathsf{Route} \leq \mathsf{List}[\mathsf{Pair}[\mathsf{Unit}, \mathsf{Path}]]$

Properties: $\{\mathsf{Net.ID} : \mathsf{UID}, \ldots, \mathsf{Route.isOpen}(\mathsf{r} : \mathsf{Route}) : \mathsf{Boolean}\}$

Compositions: $\{\mathsf{Station}{\blacklozenge}\mathsf{has} : \mathsf{Unit}, \mathsf{Station}{\blacklozenge}\mathsf{has} : \mathsf{Track}\}$

Associations: $\{\mathsf{stateAt}(\mathsf{unit} : \mathsf{Unit}, \mathsf{state} : \mathsf{UnitState}), \ldots\}$ $\square$

A class net morphism $\sigma = (\gamma, \kappa, \pi, \mu, \alpha) : \Sigma = ((C, \leq_C), K, P, M, A) \to T = ((D, \leq_D), L, Q, N, B)$ from Σ to T consists of a monotone map γ from $(C, \leq_C)$ to $(D, \leq_D)$ which is homomorphic w.r.t. the type Boolean, and the type formers List, Set, and Pair (i.e., $\gamma(\mathsf{Boolean}) = \mathsf{Boolean}$, $\gamma(\mathsf{List}[c]) = \mathsf{List}[\gamma(c)]$, $\gamma(\mathsf{Set}[c]) = \mathsf{Set}[\gamma(c)]$, and $\gamma(\mathsf{Pair}[c_1, c_2]) = \mathsf{Pair}[\gamma(c_1), \gamma(c_2)]$ for all $c, c_1, c_2 \in C$); maps κ, π, and μ between the instance specification, property, and composition declarations, such that classes are mapped along γ; and a map α from A to B for the association declarations such that n-ary association declarations are mapped to n-ary association declarations along γ, i.e., if $\alpha(a\{r_1 : c_1, \ldots, r_n : c_n\}) = b(s_1 : d_1, \ldots, s_m : d_m) \in B$, then $m = n$ and $d_i = \gamma(c_i)$ for all $1 \leq i \leq n$.

Class nets and class net morphisms form the category of *class nets*, denoted by Cl.

Instance nets — Models. A Σ-model of the UML class diagram institution for a class net $\Sigma = ((C, \leq_C), K, P, M, A)$ is given by a *Σ-instance net* $\mathcal{I} = (C^{\mathcal{I}}, K^{\mathcal{I}}, P^{\mathcal{I}}, M^{\mathcal{I}}, A^{\mathcal{I}})$ consisting of interpretations for all declarations where

- $C^{\mathcal{I}}$ maps each $c \in C$ to a non-empty set such that $C^{\mathcal{I}}(c_1) \subseteq C^{\mathcal{I}}(c_2)$ if $c_1 \leq_C c_2$ with $C^{\mathcal{I}}(\mathsf{Boolean}) = \{\mathit{ff}, \mathit{tt}\}$, $C^{\mathcal{I}}(\mathsf{List}[c]) = (C^{\mathcal{I}}(c))^*$ (where V^* denotes the finite sequences over V), $C^{\mathcal{I}}(\mathsf{Set}[c]) = \wp(C^{\mathcal{I}}(c))$ (where $\wp(V)$ denotes the powerset of V), and $C^{\mathcal{I}}(\mathsf{Pair}[c_1, c_2]) = C^{\mathcal{I}}(c_1) \times C^{\mathcal{I}}(c_2)$;
- $K^{\mathcal{I}}$ maps each $k : c \in K$ to an element of $C^{\mathcal{I}}(c)$;
- $P^{\mathcal{I}}$ maps each $c.p(x_1 : c_1, \ldots, x_n : c_n) : c' \in P$ to a partial map $C^{\mathcal{I}}(c) \times C^{\mathcal{I}}(c_1) \times \cdots \times C^{\mathcal{I}}(c_n) \rightharpoonup C^{\mathcal{I}}(c')$;
- $M^{\mathcal{I}}$ maps each $c{\blacklozenge}r : c' \in M$ to a map $C^{\mathcal{I}}(c) \to \wp(C^{\mathcal{I}}(c'))$;
- $A^{\mathcal{I}}$ maps each $a(r_1 : c_1, \ldots, r_n : c_n) \in A$ to a subset of $\prod_{1 \leq i \leq n} C^{\mathcal{I}}(c_i)$,

such that

- instance specifications $k_1 : c$ and $k_2 : c$ with $k_1 \neq k_2$ are interpreted differently: $K^{\mathcal{I}}(k_1 : c) \neq K^{\mathcal{I}}(k_2 : c)$;
- each instance has at most one owner: if $o' \in M^{\mathcal{I}}(c_1{\blacklozenge}r_1 : c'_1)(o_1) \cap M^{\mathcal{I}}(c_2{\blacklozenge}r_2 : c'_2)(o_2)$, then $o_1 = o_2$.

Example 2. An excerpt of an instance net for the class net for Fig. 1 is:

$$
\begin{aligned}
&C^{\mathcal{I}}(\mathsf{Net}) = \{\mathtt{LondonUnderground}\}\\
&C^{\mathcal{I}}(\mathsf{Station}) = \{\mathtt{CharingCross}, \mathtt{OxfordStreet}, \ldots, \mathtt{PicadillyCircus}\}\\
&C^{\mathcal{I}}(\mathsf{Connector}) = \{\mathtt{c1}, \mathtt{c2}, \mathtt{c3}, \ldots\}\\
&C^{\mathcal{I}}(\mathsf{Pair}[\mathsf{Connector}, \mathsf{Connector}]) = \{(\mathtt{c1}, \mathtt{c1}), (\mathtt{c1}, \mathtt{c2}), (\mathtt{c2}, \mathtt{c1}), (\mathtt{c2}, \mathtt{c2}), \ldots\}\\
&P^{\mathcal{I}}(\mathsf{Net.ID} : \mathsf{UID}) : C^{\mathcal{I}}(\mathsf{Net}) \rightharpoonup C^{\mathcal{I}}(\mathsf{UID}) \quad \text{where}\\
&\qquad \mathtt{LondonUnderground} \mapsto \mathtt{"LUG"} \quad \text{etc.}\\
&P^{\mathcal{I}}(\mathsf{Route.isOpen} : \mathsf{Boolean}) : C^{\mathcal{I}}(\mathsf{Route}) \rightharpoonup C^{\mathcal{I}}(\mathsf{Boolean}) \quad \text{where}\\
&\qquad \mathsf{R1} \mapsto \mathit{tt}\,\text{etc.}\\
&M^{\mathcal{I}}(\mathsf{Net}{\blacklozenge}\mathsf{has} : \mathsf{Station}) : C^{\mathcal{I}}(\mathsf{Net}) \to \wp(C^{\mathcal{I}}(\mathsf{Station})) \quad \text{where}\\
&\qquad \mathsf{LondonUnderground} \mapsto \{\mathtt{CharingCross}, \ldots, \mathtt{PicadillyCircus}\}\\
&A^{\mathcal{I}}(\mathsf{stateAt}(\mathsf{unit} : \mathsf{Unit}, \mathsf{state} : \mathsf{UnitState})) =\\
&\qquad \{(\mathtt{LU1}, \{(\mathtt{c1}, \mathtt{c2})\}), (\mathtt{LU2}, \{(\mathtt{c2}, \mathtt{c3})\}), \ldots\}
\end{aligned}
$$
□

A *Σ-instance net morphism* $\zeta : \mathcal{I} = (C^{\mathcal{I}}, K^{\mathcal{I}}, P^{\mathcal{I}}, M^{\mathcal{I}}, A^{\mathcal{I}}) \to \mathcal{J} = (C^{\mathcal{J}}, K^{\mathcal{J}}, P^{\mathcal{J}}, M^{\mathcal{J}}, A^{\mathcal{J}})$ over a class net $\Sigma = ((C, \leq_C), K, P, M, A)$ is given by a map $\zeta \in \prod c \in C\,.\,C^{\mathcal{I}}(c) \to C^{\mathcal{J}}(c)$ such that all interpretations of the declarations are mapped homomorphically, e.g., $\zeta(\mathsf{Pair}[c_1, c_2])(o_1, o_2) = (\zeta(c_1)(o_1), \zeta(c_2)(o_2))$ for all $c_1, c_2 \in C$, $o_1 \in C^{\mathcal{I}}(c_1)$, $o_2 \in C^{\mathcal{I}}(c_2)$, and $K^{\mathcal{J}}(k : c) = \zeta(c)(K^{\mathcal{I}}(k : c))$ for each $k : c \in K$. Σ-instance nets and Σ-instance net morphisms as morphisms form the category of *Σ-instance nets*, denoted by $\mathrm{Inst}(\Sigma)$.

For reducts of instance nets and instance net morphisms, let $\sigma = (\gamma, \kappa, \pi, \mu, \alpha) : \Sigma = ((C, \leq_C), K, P, M, A) \to T = ((D, \leq_D), L, Q, N, B)$ be a class net morphism. The *reduct* of a T-instance net $\mathcal{J} = (D^{\mathcal{J}}, L^{\mathcal{J}}, Q^{\mathcal{J}}, N^{\mathcal{J}}, B^{\mathcal{J}})$ along σ is the

Σ-instance net $\mathcal{J}|\sigma = (C^{\mathcal{J}|\sigma}, K^{\mathcal{J}|\sigma}, P^{\mathcal{J}|\sigma}, M^{\mathcal{J}|\sigma}, A^{\mathcal{J}|\sigma})$ formed by component-wise reducts, i.e., $C^{\mathcal{J}|\sigma}(c) = D^{\mathcal{J}}(\gamma(c))$ and similar for the other components. The *reduct* of a T-instance net morphism $\zeta : \mathcal{J}_1 \to \mathcal{J}_2$ along σ is the Σ-instance net morphism $\zeta|\sigma : \mathcal{J}_1|\sigma \to \mathcal{J}_2|\sigma$ with $\zeta|\sigma(c) = \zeta(\gamma(c))$. Setting $\mathrm{Inst}(\sigma)(\mathcal{J}) = \mathcal{J}|\sigma$ and $\mathrm{Inst}(\sigma)(\zeta) = \zeta|\sigma$, $\mathrm{Inst} : \mathrm{Cl}^{\mathrm{op}} \to \mathrm{Cat}$ is a contra-variant functor.

Multiplicity formulae — Sentences. For the sentences of the UML class diagram institution we use *multiplicity formulae* defined by the following grammar:

$$\begin{aligned}
\mathit{Frm} &::= \mathit{NumLit} \leq \mathit{FunExpr} \mid \mathit{FunExpr} \leq \mathit{NumLit} \mid \mathit{Composition}\ ! \\
\mathit{FunExpr} &::= \#\ \mathit{Composition} \mid \#\ \mathit{Association}\ [\ \mathit{Role}(,\ \mathit{Role})^*\ ] \\
\mathit{Composition} &::= \mathit{Class}{\blacklozenge}\mathit{Role} : \mathit{Class} \\
\mathit{Association} &::= \mathit{Name}\{\mathit{Role} : \mathit{Class}(,\ \mathit{Role} : \mathit{Class})^*\} \\
\mathit{Class} &::= \mathit{Name} \\
\mathit{Role} &::= \mathit{Name} \\
\mathit{NumLit} &::= 0 \mid 1 \mid \cdots
\end{aligned}$$

where Name is a set of strings. The $\leq$-formulae express constraints on the cardinalities of composition and association declarations, i.e., how many instances are allowed to be in a certain relation with others. The $\#$-expressions return the number of links in an association when some roles are fixed. The !-formulae for compositions express that the owning end must not be empty. The set of sentences or *Σ-multiplicity constraints* $\mathit{Mult}(\Sigma)$ for a class net Σ is given by the multiplicity formulae in Frm such that all mentioned elements of $\mathit{Composition}$ and $\mathit{Association}$ correspond to composition declarations and association declarations of Σ respectively, and the Role names mentioned in the last clause of $\mathit{FunExpr}$ occur in the mentioned association.

Example 3. Some of the cardinality constraints of the running example in Fig. 1 can be expressed with multiplicity formulae as follows:

$\mathsf{Station}{\blacklozenge}\mathsf{has} : \mathsf{Track}\,!$
— each Track is owned by a Station via has,

$2 \leq \#\mathsf{has}(\mathsf{connector} : \mathsf{Connector}, \mathsf{unit} : \mathsf{Unit})[\mathsf{unit}]$
— association has links each Unit to at least two Connectors,

$\#\mathsf{state}(\mathsf{unitState} : \mathsf{UnitState}, \mathsf{unit} : \mathsf{Unit})[\mathsf{unit}] = 1$
— association state links each Unit to exactly one UnitState.

Note we have used formulae with $=$ instead of two formulae with $\leq$ where the left hand side and the right hand side are switched. □

The *translation* of a formula $\varphi \in \mathit{Mult}(\Sigma)$ along a class net morphism $\sigma = (\gamma, \kappa, \pi, \mu, \alpha) : \Sigma \to T$, written as $\sigma(\varphi)$, is given by applying σ to compositions, associations, and role names.

Satisfaction relation. The *Σ-satisfaction relation* $- \models_\Sigma - \subseteq |\mathrm{Inst}(\Sigma)| \times \mathit{Sen}(\Sigma)$ of the UML class diagram institution is defined for each Σ-instance net $\mathcal{I} = (C^{\mathcal{I}}, K^{\mathcal{I}}, P^{\mathcal{I}}, M^{\mathcal{I}}, A^{\mathcal{I}})$ as

$$\begin{aligned}
&\mathcal{I} \models_{\Sigma} \ell \leq \#c{\blacklozenge}r : c' \quad\Leftrightarrow\quad \forall o \in C^{\mathcal{I}}(c) \,.\, [\![\ell]\!] \leq M^{\mathcal{I}}(c{\blacklozenge}r : c')(o)\\
&\mathcal{I} \models_{\Sigma} \#c{\blacklozenge}r : c' \leq \ell \quad\Leftrightarrow\quad \forall o \in C^{\mathcal{I}}(c) \,.\, M^{\mathcal{I}}(c{\blacklozenge}r : c')(o) \leq [\![\ell]\!]\\
&\mathcal{I} \models_{\Sigma} c{\blacklozenge}r : c'! \quad\Leftrightarrow\quad \forall o' \in C^{\mathcal{I}}(c') \,.\, \exists o \in C^{\mathcal{I}}(c) \,.\, M^{\mathcal{I}}(c{\blacklozenge}r : c')(o) = o'\\
&\mathcal{I} \models_{\Sigma} \ell \leq \#a(r_1 : c_1, \ldots, r_n : c_n)[r_{i_1}, \ldots, r_{i_m}] \quad\Leftrightarrow\\
&\quad \forall o_{i_1} \in C^{\mathcal{I}}(c_{i_1}), \ldots, o_{i_m} \in C^{\mathcal{I}}(c_{i_m}) \,.\\
&\qquad [\![\ell]\!] \leq |\{t \in A^{\mathcal{I}}(a(r_1 : c_1, \ldots, r_n : c_n)) \mid t_{i_1} = o_{i_1}, \ldots, t_{i_m} = o_{i_m}\}|\\
&\mathcal{I} \models_{\Sigma} \#a(r_1 : c_1, \ldots, r_n : c_n)[r_{i_1}, \ldots, r_{i_m}] \leq \ell \quad\Leftrightarrow\\
&\quad \forall o_{i_1} \in C^{\mathcal{I}}(c_{i_1}), \ldots, o_{i_m} \in C^{\mathcal{I}}(c_{i_m}) \,.\\
&\qquad |\{t \in A^{\mathcal{I}}(a(r_1 : c_1, \ldots, r_n : c_n)) \mid t_{i_1} = o_{i_1}, \ldots, t_{i_m} = o_{i_m}\}| \leq [\![\ell]\!]
\end{aligned}$$

where $[\![-]\!] : \mathit{NumLit} \to \mathbb{Z}$ maps a numerical literal to an integer.

Theorem 1. $\mathrm{ClDiag} = (\mathrm{Cl}, \mathit{Mult}, \mathrm{Inst}, \models)$ *forms an institution.*

Proof. It remains to prove the satisfaction condition $\mathcal{J} \models_T \sigma(\varphi) \Leftrightarrow \mathcal{J}|\sigma \models_{\Sigma} \varphi$ for each $\sigma : \Sigma \to T$ in Cl, each $\mathcal{J} \in |\mathrm{Inst}(T)|$, and each $\varphi \in \mathit{Mult}(\Sigma)$, which follows by induction on the structure of formulae. □

4.2 An Institution for UML Class Diagrams with Rigidity Constraints

Let $\mathscr{I} = (\mathrm{Sig}^{\mathscr{I}}, \mathit{Sen}^{\mathscr{I}}, \mathrm{Mod}^{\mathscr{I}}, \models^{\mathscr{I}})$ be an institution. We define an institution $\wp^{\mathrm{i}}_{*}\mathscr{I}$ over $\mathscr{I}$ that has as models pointed powersets $(M_0, \mathcal{M})$ of models of $\mathscr{I}$; inherits the sentences from $\mathscr{I}$ which now have to hold in all models of $\mathcal{M}$; and also requires that in each model $(M_0, \mathcal{M})$ over a signature Σ all elements of $\mathcal{M}$ "behave" like M_0 for a certain part Γ of the signature Σ.

Define the signature category $\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}$ of $\wp^{\mathrm{i}}_{*}\mathscr{I}$ as follows: The objects of $\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}$ are the morphisms $\sigma : \Gamma \to \Sigma$ in $\mathrm{Sig}^{\mathscr{I}}$. A morphism $(\gamma, \rho) : (\sigma : \Gamma \to \Sigma) \to (\sigma' : \Gamma' \to \Sigma')$ of $\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}$ consists of morphisms $\gamma : \Gamma \to \Gamma'$ and $\rho : \Sigma \to \Sigma'$ in $\mathrm{Sig}^{\mathscr{I}}$ such that $\sigma' \circ \gamma = \rho \circ \sigma$.

Define the sentence functor $\mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}} : \mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}} \to \mathrm{Set}$ of $\wp^{\mathrm{i}}_{*}\mathscr{I}$ by $\mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\sigma : \Gamma \to \Sigma) = \mathit{Sen}^{\mathscr{I}}(\Sigma)$ and $\mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\gamma, \rho) = \mathit{Sen}^{\mathscr{I}}(\rho)$.

Define the contra-variant model functor $\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}} : (\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}})^{\mathrm{op}} \to \mathrm{Cat}$ of $\wp^{\mathrm{i}}_{*}\mathscr{I}$ as follows: Let $(\sigma : \Gamma \to \Sigma) \in |\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}|$. The objects of $\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\sigma)$ are the pairs $(M_0, \mathcal{M})$ with $M_0 \in |\mathrm{Mod}^{\mathscr{I}}(\Sigma)|$ and $\mathcal{M}$ a sub-*set* of $|\mathrm{Mod}^{\mathscr{I}}(\Sigma)|$ such that $M_0 \in \mathcal{M}$ and $\mathrm{Mod}^{\mathscr{I}}(\sigma)(M) = \mathrm{Mod}^{\mathscr{I}}(\sigma)(M_0)$ for each $M \in \mathcal{M}$. A morphism of $\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\sigma)$ from $(M_0, \mathcal{M})$ to $(M'_0, \mathcal{M}')$ is given by a morphism $\chi_0 : M_0 \to M'_0$ in $\mathrm{Mod}^{\mathscr{I}}(\sigma)$ where $\mathcal{M}$ and $\mathcal{M}'$ satisfy the condition that for every $M \in \mathcal{M}$ there is an $M' \in \mathcal{M}'$ and a morphism $\chi : M \to M'$. — Let $(\gamma, \rho) : (\sigma : \Gamma \to \Sigma) \to (\tau : \Delta \to T)$ be a morphism in $\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}$. Then $\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\gamma, \rho)(N_0, \mathcal{N}) = (\mathrm{Mod}^{\mathscr{I}}(\rho)(N_0), \{\mathrm{Mod}^{\mathscr{I}}(\rho)(N) \mid N \in \mathcal{N}\})$ and $\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\gamma, \rho)(\psi_0) = \mathrm{Mod}^{\mathscr{I}}(\rho)(\psi_0)$.

For each $(\sigma : \Gamma \to \Sigma) \in |\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}|$, define the satisfaction relation $- \models^{\wp^{\mathrm{i}}_{*}\mathscr{I}}_{\sigma:\Gamma\to\Sigma} - \subseteq |\mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\sigma : \Gamma \to \Sigma)| \times \mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}(\sigma : \Gamma \to \Sigma)$ of $\wp^{\mathrm{i}}_{*}\mathscr{I}$ by $(M_0, \mathcal{M}) \models^{\wp^{\mathrm{i}}_{*}\mathscr{I}}_{\sigma:\Gamma\to\Sigma} \varphi \Leftrightarrow \forall M \in \mathcal{M} \,.\, M \models^{\mathscr{I}}_{\Sigma} \varphi$.

Theorem 2. $(\mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}, \mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}, \mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}, \models^{\wp^{\mathrm{i}}_{*}\mathscr{I}})$ *forms an institution.* □

We now apply this construction to the UML class diagram institution ClDiag to represent rigidity. In the following, we assume all signature elements of the class diagram to be consistently and completely annotated with stereotypes «rigid» and «dynamic». Consistency rules include: Boolean is «rigid»; if a classifier c is «dynamic», then List$[c]$ is «dynamic» as well, etc. Classifiers c that are added by the signature closure are «rigid» unless forced to be «dynamic» by consistency rules. We form a class net consisting only of those elements that are stereotyped with «rigid»; additionally we form the full class net. Since the signatures of ClDiag are set-based, there is an inclusion morphism which we take as the signature in $\wp^{\mathrm{i}}_{*}$ClDiag. Thus, the meaning of all elements stereotyped with «rigid» is fixed by their meanings in the distinguished state. In fact, for representing UML class diagrams with rigidity constraints it is enough to work with those signatures in $\wp^{\mathrm{i}}_{*}$ClDiag which are inclusions in the category of signatures Cl of ClDiag. We denote this institution by $\wp^{\hookrightarrow}_{*}$ClDiag.

There is a straightforward simple institution comorphism from $\wp^{\mathrm{i}}_{*}\mathscr{I}$ to $\mathscr{I}$ (which also applies to $\wp^{\hookrightarrow}_{*}$ClDiag): Define the functor $\Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}} : \mathrm{Sig}^{\wp^{\mathrm{i}}_{*}\mathscr{I}} \to \mathrm{Pres}^{\mathscr{I}}$ by $\Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}(\sigma : \Gamma \to \Sigma) = (\Sigma, \emptyset)$ and $\Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}(\gamma, \rho) = \rho$. Define the natural transformation $\alpha^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}} : \mathit{Sen}^{\wp^{\mathrm{i}}_{*}\mathscr{I}} \stackrel{\cdot}{\to} \mathit{Sen}^{\mathscr{I}} \circ \Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}$ by $\alpha^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}_{\sigma:\Gamma\to\Sigma}(\varphi) = \varphi$ for $\varphi \in \mathit{Sen}^{\mathscr{I}}(\Sigma)$. Define the natural transformation $\beta^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}} : \mathrm{Mod}^{\mathscr{I}} \circ (\Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}})^{\mathrm{op}} \stackrel{\cdot}{\to} \mathrm{Mod}^{\wp^{\mathrm{i}}_{*}\mathscr{I}}$ by $\beta^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}_{\sigma:\Gamma\to\Sigma}(M) = (M, \{M\})$ and $\beta^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}_{\sigma:\Gamma\to\Sigma}(\chi) = \chi$. Then we obtain:

Theorem 3. $\mu^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}} = (\Phi^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}, \alpha^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}}, \beta^{\wp^{\mathrm{i}}_{*}\mathscr{I},\mathscr{I}})$ *forms a simple institution comorphism from* $\wp^{\mathrm{i}}_{*}\mathscr{I}$ *to* $\mathscr{I}$. □

4.3 From UML Class Diagrams with Rigidity Constraints to ModalCasl

In the following, we give a sketch of a simple institution comorphism from UML class diagrams with rigidity constraints to ModalCasl: Let $(\Gamma \subseteq \Sigma) \in |\mathrm{Sig}^{\wp^{\hookrightarrow}_{*}\mathrm{ClDiag}}|$ with classifier net $\Sigma = ((C, \leq_C), K, P, M, A)$. We map $(\Gamma \subseteq \Sigma)$ to the following ModalCasl-presentation $\Phi(\Gamma \subseteq \Sigma) = (((S, \mathit{TF}, \mathit{PF}, P, \leq), \mathit{rigid}, \mathit{modalities}, \mathit{modalitySorts}), \mathit{Ax})$:

Let us denote the connected components of $\leq_C$ by $cc(\leq_C)$ and the connected component of a class name[2] $c \in C$ by $[c]_{\leq_C}$, i.e., $cc(\leq_C) = \{[c]_{\leq_C} \mid c \in C\}$. Then,

$$S = C \cup cc(\leq_C), \quad \leq \, = \, \leq_C \cup \{(c, [c]_{\leq_C}) \mid c \in C\} \cup \{([c]_{\leq_C}, [c]_{\leq_C}) \mid c \in C\}$$

For the classifier Boolean and classifiers involving type formers such as Pair$[c_1, c_2]$ we include and (possibly) instantiate specifications from the Casl Basic Datatypes [23]. For example, the sort *PairConnectorConnector* is specified by instantiating the Casl Pair specification and renaming the sort name:

Pair[**sort** *Connector*][**sort** *Connector*] **with**
Pair[**sort** *Connector*][**sort** *Connector*] $\mapsto$ *PairConectorConector*

[2] Here we assume $[c]_{\leq_C} = t$, if t is the top sort of the component of c.

Here, we assume that the semantics of each predefined type and type former is an element of the model class of the respective monomorphic CASL datatype.

Considering the running example, we obtain:

%% Classes:
sorts *Net*, *Station*, *Unit*, ..., *UID*
%% Hierarchy:
sorts *Point*, *Linear* < *Unit*; ...; *Route* < *ListPairUnitPath*

The total function symbols TF just comprise the instance specification declarations, the partial function symbols PF the property declarations:

$$TF = \{k : \ \to c \mid k : c \in K\}$$
$$PF = \{p : c \times c_1 \cdots \times c_n \to ?\ c' \mid c.p(x_1 : c_1, \dots, x_n : c_n) : c' \in P\}$$

Considering the running example, we obtain:

%% Properties:
rigid op *id* : *Net* $\to$? *UID*
flexible ops *isClosedAt* : *Unit* $\to$? *Boolean*; ...

The classification into"rigid" and "flexible" results directly from $(\Gamma \subseteq \Sigma)$.

The predicate symbols P comprise the composition declarations M and the association declarations A, as well as a predicate $isAlive$ that is used to model "flexible" sort interpretations in MODALCASL:

$$\begin{aligned} P = &\{r : c \times c' \mid c \blacklozenge r : c' \in M\} \cup \\ &\{a : c_1 \times \cdots \times c_n \mid a(r_1 : c_1, \dots, r_n : c_n) \in A\} \cup \\ &\{isAlive : c \mid c \in C\} \end{aligned}$$

Considering the running example, we obtain:

%% Compositions:
rigid preds __*has*__ : *Station* × *Unit*; __*has*__ : *Station* × *Track*;
%% Associations:
rigid preds __*has*__ : *Unit* × *Connector*; __*has*__ : *Linear* × *Connector*; ...
%% Is Alive preds:
rigid preds *isAlive* : *Net*; *isAlive* : *Station*; *isAlive* : *Unit*; ...; *isAlive* : *UID*;

Once again, Γ determines which of the predicates in P are rigid. The predicate $isAlive$ is rigid if the corresponding class is rigid. In our institution comorphism, the $modalities$ set is just $\{\epsilon\}$, and the $modalitySorts$ predicate is false for all sorts. That is the models have exactly one accessibility relation defined between worlds. Naturally, representing CDs in MODALCASL imposes no constraints on this relation.

Finally, we need axioms for our comorphism. The first group of axioms stipulates that arguments of operations and predicates need to be "alive". The second group of axioms corresponds to the condition on Σ-instance nets that each instance has at most one owner.

$$\begin{aligned}
Ax = \{&\mathit{isAlive}(k : c) \mid k : c \in K\} \\
&\cup \{\forall x : c, x_1 : c_1, \dots, x_n : c_n . \mathit{def}\ p(x, x_1, \dots, x_n) \implies \\
&\qquad \mathit{isAlive}(x) \wedge \mathit{isAlive}(x_1) \wedge \dots \mathit{isAlive}(x_n) \mid \\
&\qquad\quad c.p(x_1 : c_1, \dots, x_n : c_n) : c' \in P\} \\
&\cup \{\forall x : c, x' : c'.r(x, x') \implies \mathit{isAlive}(x) \wedge \mathit{isAlive}(x') \mid c \blacklozenge r : c' \in M\} \\
&\cup \{\forall x_1 : c_1, \dots, x_n : c_n . a(x_1, \dots, x_n) \implies \\
&\qquad \mathit{isAlive}(x_1) \wedge \dots \mathit{isAlive}(x_n) \mid a(r_1 : c_1, \dots, r_n : c_n) \in A\} \\
&\cup \{\forall x_1 : [c_1]_{\leq_C}, x_2 : [c_2]_{\leq_C}, x : [c'_1]_{\leq_C} . r(x_1, x) \wedge r(x_2, x) \Rightarrow x_1 = x_2 \mid \\
&\qquad c_1 \blacklozenge r_1 : c'_1, c_2 \blacklozenge r_2 : c'_2 \in M,\ [c_1]_{\leq_C} = [c_2]_{\leq_C},\ [c'_1]_{\leq_C} = [c'_2]_{\leq_C}\}
\end{aligned}$$

The sentences of $\wp_*^{\hookrightarrow}$ClDiag are the sentences of ClDiag. These can be systematically encoded in first order logic using "poor man's counting", e.g. [8], by providing the necessary number of variables, e.g. $\#(c \blacklozenge r : c') \geq n$ translates to

$$\forall x : c\ \exists y_1, \dots, y_n : c' : \mathit{pwDifferent}(y_1, \dots, y_n) \wedge r(x, y_1) \wedge \dots \wedge r(x, y_n)$$

i.e., for every value x in c we find at least n different values in c' of which x comprises of. Here, $\mathit{pwDifferent}(y_1, \dots, y_n)$ encodes $y_i \neq y_j$ for $i \neq j$, $1 \leq i, j \leq n$. A typical example is:

$$\begin{aligned}
&2 \leq \#\mathsf{has}(\mathsf{connector} : \mathsf{Connector}, \mathsf{unit} : \mathsf{Unit})[\mathsf{unit}] \mapsto \\
&\qquad \forall u : \mathit{Unit} . \exists c1, c2 : \mathit{Connector} . c1 \neq c2 \wedge \mathit{has}(c1, u) \wedge \mathit{has}(c2, u)
\end{aligned}$$

Let $\mathcal{M}$ be a ModalCasl model in $|\mathrm{Mod}^{\mathrm{ModalCasl}}(\Phi(\Gamma \subseteq \Sigma))|$, let W be the sets of worlds in $\mathcal{M}$, $i \in W$ the initial world, and M_w the Casl models associated with $w \in W$. In the following, we use the abbreviations $\lceil x \rceil = (\mathit{inj}_{c,[c]})_{M_w}(x)$ for $x \in c_{M_w}$; and $\lfloor x \rfloor = (\mathit{pr}_{[c],c})_{M_w}(x)$ for $x \in [c]_{M_w}$. We first define for each w, how to turn its associated Casl model M_w into a *Σ-instance net* $\beta(M_w) = (C^{M_w}, K^{M_w}, P^{M_w}, M^{M_w}, A^{M_w})$:

- If c does not involve a type former: $C^{M_w}(c) = \{\lceil x \rceil \mid x \in c_{M_w}, \mathit{isAlive}(x)\}$.
  If c involves a type former, we take the representation of the corresponding type in the instance net.
- $K^{M_w}(k : c) = \lceil (k_{\langle\rangle,c})_{M_w} \rceil$.
- $P^{M_w}(c.p(x_1 : c_1, \dots, x_n : c_n) : c')(y, y_1, \dots, y_n) = \lceil (p_{\langle c, c_1, \dots, c_n \rangle, c'})_{M_w}(\lfloor y \rfloor, \lfloor y_1 \rfloor, \dots, \lfloor y_n \rfloor) \rceil$ for all $y \in C^{M_w}(c)$, $y_1 \in C^{M_w}(c_1), \dots, y_n \in C^{M_w}(c_n)$.
- $M^{M_w}(c \blacklozenge r : c')(x) = \{\lceil y \rceil \mid (r_{c,c'})_{M_w}(\lfloor x \rfloor, y)\}$ for all $x \in C^{M_w}(c)$.
- $A^{M_w}(a(r_1 : c_1, \dots, r_n : c_n)) = \{(\lceil x_1 \rceil, \dots, \lceil x_n \rceil) \mid (x_1, \dots, x_n) \in (a_{\langle c_1, \dots c_n \rangle})_{M_w}\}$.

By construction, $\beta(\mathcal{M}) = (\beta(M_i), \{\beta(M_w) \mid w \in W\})$ is a model in $|\mathrm{Mod}^{\wp_*^{\hookrightarrow} \mathrm{ClDiag}}(\Gamma \subseteq \Sigma)|$.

5 Crafting a Formal DSL for the Railway Domain

Based on the informal DSL as presented in Section 2, we now demonstrate how to create a formal DSL with the techniques developed so far. This process fits with the "Designing DSLs for verification" methodology presented in [15].

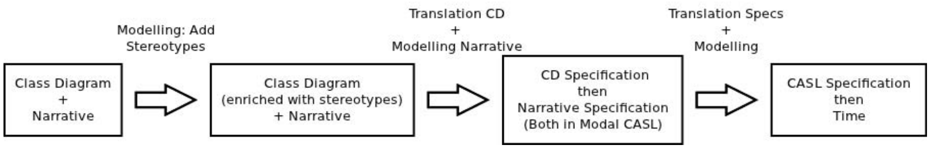

Modelling: Add Stereotypes
Translation CD + Modelling Narrative
Translation Specs + Modelling
Class Diagram + Narrative
Class Diagram (enriched with stereotypes) + Narrative
CD Specification then Narrative Specification (Both in Modal CASL)
CASL Specification then Time

Fig. 2. Process for capturing a DSL

Step 1. Start by creating a UML CD giving concepts and relationships between concepts within the desired domain. Add an accompanying narrative as illustrated in Section 2. Next, in a modelling step, add stereotypes (as discussed in Section 2). These stereotypes are used to describe elements which remain static or are dynamic within the system. *Result:* CD and Narrative – as one can find in industry, see, e.g., [13].

Step 2. Using the comorphism defined in Section 4.3, translate the CD into MODALCASL. Next, extend the resulting MODALCASL specification by a modelling of any narrative elements not captured within the CD. *Result:* MODALCASL specification capturing CD and Narrative.

For example, considering the narrative of what it means for the *Route* to be "open" from Bjørner's DSL – see Section 2 – we can produce the following MODALCASL specification:

$$
\begin{array}{l}
\textbf{flexible op } \mathit{isOpen} : \mathit{Route} \rightarrow ?\ \mathit{Boolean}\\
\forall\ r : \mathit{Route};\ \mathit{upp} : \mathit{UnitPathPair};\ u : \mathit{Unit};\ \mathit{us} : \mathit{UnitState};\ p : \mathit{Path}\\
\bullet\ \mathit{isOpen}(r) = \mathit{tt}\\
\quad \Leftrightarrow r\ \mathit{has}\ \mathit{upp} \wedge \mathit{upp}\ \mathit{getUnit}\ u \wedge \mathit{upp}\ \mathit{getPath}\ p \wedge u\ \mathit{stateAt}\ \mathit{us} \wedge \mathit{us}\ \mathit{has}\ p
\end{array}
$$

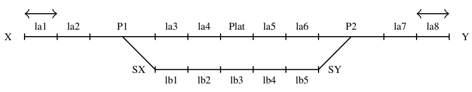
Also, given the methodology presented in [15], at this point any verification conditions required can be modelled and added to the specification. In the case of Bjørner's DSL, one would like to check that two *Route*s which share any *Unit*s (i.e. are "conflicting"):

$$
\begin{array}{l}
\textbf{rigid pred } \_\_\mathit{inConflict}\_\_ : \mathit{Route} \times \mathit{Route}\\
\forall\ r1, r2 : \mathit{Route}\\
\bullet\ r1\ \mathit{inConflict}\ r2\\
\quad \Leftrightarrow \neg\ r1 = r2 \wedge \exists\ \mathit{upp1}, \mathit{upp2} : \mathit{UnitPathPair};\ u : \mathit{Unit}\\
\qquad \bullet\ r1\ \mathit{has}\ \mathit{upp1} \wedge r2\ \mathit{has}\ \mathit{upp2} \wedge \mathit{upp1}\ \mathit{getUnit}\ u \wedge \mathit{upp2}\ \mathit{getUnit}\ u
\end{array}
$$

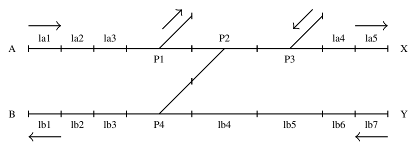
are not "open" at the same time:

%% Verification condition:
$\forall\ r1, r2 : \mathit{Route} \bullet r1\ \mathit{inConflict}\ r2 \Rightarrow \neg\ (\mathit{isOpen}(r1) = \mathit{tt} \wedge \mathit{isOpen}(r2) = \mathit{tt})$

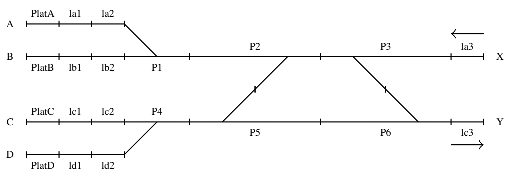
Step 3. The next step of the process is to apply the existing MODALCASL to CASL comorphism [21] to gain a CASL Specification of the DSL. This will allow connection to multiple theorem provers for later use in verification. If appropriate, one can specify in CASL how worlds evolve, e.g., by adding a loose specification of discrete time. *Result:* CASL specification of the formal DSL.



(a) A pass through station (TP-A)



(b) A double junction (TP-B)



(c) A terminal station (TP-C)

Fig. 3. Verified track plans

Using the illustrative examples from Section 4.3, the translation results in the following CASL specification:

sorts *AllStates*, *Boolean*, *Connector*, *Line*, *Linear*,...
sorts *Line*, *Track* $<$ *ListLinear*; *UnitState* $<$ *SetPath*;
op *isClosed* : *g_World* $\times$ *Unit* $\rightarrow?$ *Boolean*; ...
pred *__hasStation__* : *Net* $\times$ *Station*; *__hasLine__* : *Net* $\times$ *Line*;...
pred *isAlive* : *Net*; *isAlive* : *Path*; ...
$\forall$ *g_w1* : *g_World*; *r* : *Route*; *upp* : *UnitPathPair*; *u* : *Unit*; *us* : *UnitState*; *p* : *Path*
• *isOpen*(*g_w1*, *r*) = *tt*
$\Leftrightarrow$ *r has upp* $\wedge$ *upp getUnit u* $\wedge$ *upp getPath p*
...

In this specification, we can see that all elements which were labelled as flexible have now been associated with a *g_World* representing a world in the Kripke structure of MODALCASL. Also, all elements which were labelled as rigid have been translated to elements ignoring *g_World*. The final result of the overall process is a formal DSL that can be used for verification.

5.1 Verification

Once a DSL has been captured in CASL, the techniques described in [15] can be applied. That is, the DSL can be enriched with a series of domain specific lemmas that are proven as a consequence of the DSL. Such lemmas are often specific to the property to be proven. Next, the DSL can be used to describe a particular railway network. For each railway network described, the verification condition can, thanks to the added domain specific lemmas, be automatically discharged using Hets as a broker to various automatic theorem provers [15].

With respect to Bjørner's DSL, this approach has been successfully applied, that is, Bjørner's DSL has been successfully extended for verification. The result of this work is that we have successfully automatically verified a number of track plans of real world complexity against the verification condition that "Conflicting open routes can not be open at the same time". Several of these track plans are illustrated in Figure 3.

Verification times for all track plans are within the region of seconds[3], with TP-A requiring 9.79s, TP-B requiring 2.94s and TP-C requiring 26.35s. The increase in time for track plan TP-C is due to the large number (48 in total) of conflicting routes.

With respect to the Invensys Rail Data Model [13], translation of the CDs required for verification and the modelling of the narrative has resulted in a formal DSL in CASL. The development of domain knowledge for verification is ongoing work. First results on small scale verification problems demonstrate that the full approach is successful.

6 Conclusion

We have suggested a methodology for formal DSL design. Starting with an industrial document in the from a UML class diagram with accompanying natural language descriptions, we have defined a translation based process to obtain a formal specification in CASL that can be used for verification. To this end, we have developed the theoretical concepts required, including an institution for UML CDs and a institution comorphisms to MODALCASL. Finally, we have illustrated the approach works by applying the process to Bjørner's DSL for the railway domain.

Future work. The Heterogeneous Toolset [22] makes extensive use of institutions and and their relations for tool support for various logics. We intend to implement the presented institutions and translations within Hets. Further to this, we hope to complete the development of suitable domain specific lemmas for the Invensys Rail Data Model, and to produce, as outlined in [15], a graphical front end with automatic proof support for use by railway engineers. On the more theoretical side, we want to further investigate the relations of the institutions involved.

Acknowledgements. The authors would like to thank S. Chadwick and D. Taylor from Invensys Rail for their continued support and encouraging feedback. We also thank Erwin R. Catesbeiana for sharing with us his stereotypical views on objects, flies, and the dynamics of the world.

References

1. Andova, S., van den Brand, M., Engelen, L.: Prototyping the Semantics of a DSL using ASF+SDF: Link to Formal Verification of DSL Models. In: AMMSE 2011. Electr. Proc. Theo. Comp. Sci., vol. 56, pp. 65–79 (2011)
2. Arnold, B., van Deursen, A., Res, M.: An Algebraic Specification of a Language for Describing Financial Products. In: Wirsing, M. (ed.) Wsh. Formal Methods Applications in Software Engineering Practice, Seattle, pp. 6–13 (1995)
3. Bjørner, D.: Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In: CTS 2003. Elsevier (2003)

[3] PC with Intel 3.0GHz Quad Core processor with 8GB RAM running Ubuntu 12.10.

4. Bjørner, D.: Domain Engineering – Technology Management, Research and Engineering. JAIST Press (2009)
5. Bonachea, D., Fisher, K., Rogers, A., Smith, F.: Hancock: A Language for Processing Very Large-scale Data. SIGPLAN Notices 35(1) (2000)
6. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A Heterogeneous Approach to UML Semantics. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 383–402. Springer, Heidelberg (2008)
7. dos Santos, O.M., Woodcock, J., Paige, R.: Using Model Transformation to Generate Graphical Counter-Examples for the Formal Analysis of xUML Models. In: 16th Int. Conf. Engineering of Complex Computer Systems (ICECCS 2011), pp. 117–126. IEEE (2011)
8. Ebbinghaus, H., Flum, J., Thomas, W.: Mathematical logic. Springer (1994)
9. Fowler, M.: Domain Specific Languages. Addison-Wesley (2010)
10. Hughes, G.E., Cresswell, M.J.: A new introduction to modal logic. Routledge (1996)
11. Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the ACM 39, 95–146 (1992)
12. Hussmann, H., Cerioli, M., Baumeister, H.: From uml to casl (static part). Technical Report DISI-TR-00-06, DISI-Universit di Genova (2000)
13. Invensys Rail. Data Model – Version 1A (2010)
14. James, P., Knapp, A., Mossakowski, T., Roggenbach, M.: From UML Class Diagrams to Modal CASL. Technical report, Universität Augsburg (to appear, 2013)
15. James, P., Roggenbach, M.: Designing domain specific languages for verification: First steps. In: Höfner, P., McIver, A., Struth, G. (eds.) 1st Wsh. Automated Theory Engineering (ATE 2011). CEUR Wsh. Proc., vol. 760, pp. 40–45. CEUR-WS.org (2011)
16. Kerr, D., Rowbotham, T.: Introduction to Railway Signalling. Institution of Railway Signal Engineers (2001)
17. Lano, K., Clark, D., Androutsopoulos, K.: UML to B: Formal Verification of Object-Oriented Models. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 187–206. Springer, Heidelberg (2004)
18. Meng, S., Aichernig, B.: Towards a Coalgebraic Semantics of UML: Class Diagrams and Use Cases. Technical Report 272, UNU/IIST (2003)
19. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-specific Languages. ACM Computing Surveys 37(4) (2005)
20. Mossakowski, T.: Relating CASL with Other Specification Languages: The Institution Level. Theo. Comp. Sci. 286, 367–475 (2002)
21. Mossakowski, T.: ModalCASL — Specification with Multi-Modal Logics. Language Summary (2004)
22. Mossakowski, T.: HeTS — the Heterogeneous Tool Set, home page (June 2011), `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/`
23. Mosses, P.D. (ed.): CASL Reference Manual. Springer (2004)
24. Object Managment Group. Unified Modeling Language (UML), v2.4.1 (2011), `http://www.omg.org/spec/UML/2.4.1`