Innovations Syst Softw Eng (2010) 6:91–98
DOI 10.1007/s11334-009-0118-z

ORIGINAL PAPER

Bringing class diagrams to life

Luis S. Barbosa · Sun Meng

Received: 27 November 2009 / Accepted: 7 December 2009 / Published online: 24 December 2009

Abstract Research in formal methods emphasizes a fundamental interconnection between modeling, calculation and prototyping, made possible by a common unambiguous, mathematical semantics. This paper, building on a broader research agenda on coalgebraic semantics for Unified Modeling Language diagrams, concentrates on class diagrams and discusses how such a coalgebraic perspective can be of use not only for formalizing their specification, but also as a basis for prototyping.

Keywords UML class diagrams · Coalgebraic modeling

1 Introduction

The Unified Modeling Language (UML) is defined, by [9], as *'a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system'*. In practice, it encompasses a collection of inter-related, semi-formal design notations for software development, providing a unified, expressive and widely adopted framework—a de facto standard. It lacks, however, a rigorous and consensual semantic definition leading, therefore, to weak effective support to the design of complex systems and, often, to partially conflicting support tools.

On the other hand, most research in Formal Methods emphasizes a fundamental interconnection between *modeling*, *calculation* and *prototyping*, made possible by a common unambiguous, mathematical semantics. In this context *modeling* refers to the ability to choose the right abstractions for a problem domain, and *calculation* stresses the need for expressing such abstractions in a framework whose mathematical structure is sufficiently rich to enable rigorous reasoning. On the other hand, *prototyping* is related to the execution of abstract models to simulate systems' behavior and gather empirical evidence about their properties.

Such an interconnection seems particularly appropriate with respect to the formalization of UML. However, the number and diversity of diagrams expressing a UML model makes it difficult to base its semantics on a single framework, which must be expressive enough to interpret different languages capturing both static and dynamic aspects of systems. On the other hand, some of the formalizations proposed in the literature are essentially descriptive and difficult to use for either verification or prototyping.

As a contribution to face this challenge, the authors introduced, in a series of previous publications, a generic coalgebraic semantic framework for different models in UML, including class diagrams, use cases, statecharts and sequence diagrams [3,11,13,14]. In such a framework, the semantics of different kinds of models are given as coalgebras [7,10] which encapsulate a state space, regarded as a black box with limited access via specific observers. Notions of bisimulation and refinement capture observational equivalence and simulation preorders, respectively. Such standard tools in coalgebra theory can form the basis of a whole discipline of reasoning and transforming UML designs. Actually, if the semantics of different UML models can be presented as coalgebras for suitable functors, we will end up with a *uniform setting* for tackling the diversity of such models, their properties and inter-relations.

This paper is part of this broad research agenda. It concentrates on *class diagrams* and discusses how a

L. S. Barbosa
DI-CCTC, Minho University, Braga, Portugal
e-mail: lsb@di.uminho.pt

S. Meng (✉)
CWI, Science Park 123, 1090 GB Amsterdam, The Netherlands
e-mail: M.Sun@cwi.nl

coalgebraic perspective can be of use not only for formalizing their specification, but also as a basis for prototyping.

Recall that a UML *class diagram* captures the static structure of a system, as a set of *classes* and relationships, called *associations*, between them. Classes may be further annotated with *constraints*, i.e., properties that must hold for every class instance (or object) along its lifetime. In our approach the dynamics of individual classes, their aggregations and the whole diagram (populated with class instances) are expressed coalgebraically. Each coalgebra corresponds to some sort of transition machine, therefore enabling an interface for prototyping by directly invoking the coalgebra operations. As a proof of concept the prototyping setting described in the paper has been implemented in HASKELL.

The remaining of the paper is organized as follows: Section 2 discusses coalgebraic models for classes, their composition and prototyping. A few elementary concepts in coalgebra theory are briefly recalled. Sections 3 and 4 deal with constraints and associations in a class diagram, respectively. Section 4 also introduces the *diagram engine*, a coalgebraic structure to cope with the global diagram dynamics. Finally, Sect. 5 concludes and points out some issues for future work.

2 Classes

2.1 A coalgebraic perspective

A class declaration in UML *class diagram* introduces a signature of attributes and methods, acting as a type for its possible instances. Such a definition, as discussed below, has a lot in common with the specification of the type of a coalgebra; technically a functor which captures the interface through which its state space can be observed and modified.

Recall that, given such a functor T, understood as a specification of a signature of *observers*, a T-coalgebra is simply a function $p : \mathsf{T}U \longleftarrow U$ mapping elements of a state space U into their observations through T. A useful metaphor identifies functor T with a 'lens' , providing the unique, limited way through which the state of a system is observed. Similarly, a coalgebra $p : \mathsf{T}U \longleftarrow U$ is regarded as a formal description of the observation process.

Alternatively, a T-coalgebra p can be thought of as a generalized *transition system* ${}_p\!\longleftarrow$, the shape of transitions being determined by T according to

$$ {}_p\!\longleftarrow \; = \; \in_{\mathsf{T}} \cdot p \tag{1}$$

or, introducing variables,

$$u' \;{}_p\!\longleftarrow\; u \equiv u' \in_{\mathsf{T}} p\ u$$

where relation $\in_{\mathsf{T}}$ denotes structural membership. Relation $\in_{\mathsf{T}}$ coincides with datatype membership defined in [6] by a

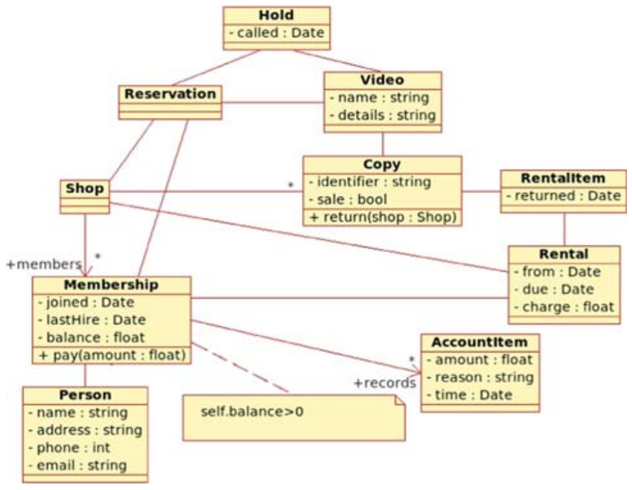

Fig. 1 An example

Galois connection. For the powerset functor, $\in_{\mathsf{T}}$ amounts to standard set membership, while for polynomial functors the following inductive definition applies (see [12] for details):

$$\begin{aligned}
&\in_{\mathsf{Id}} = id\\
&\in_{\mathsf{K}} = \bot\\
&\in_{T_1 \times T_2} = (\in_{T_1} \cdot \pi_1) \cup (\in_{T_2} \cdot \pi_2)\\
&\in_{T_1 + T_2} = [\in_{T_1}, \in_{T_2}]\\
&\in_{T_1 \cdot T_2} = \in_{T_2} \cdot \in_{T_1}\\
&\in_{T^K} = \bigcup_{k \in K} \in_T \cdot \beta_k \quad (\text{where } \beta_k f \; = f \; k)
\end{aligned}$$

Redirecting our attention to UML, consider now the example class diagram depicted in Fig. 1, which is a simplified model of a video renting e-business. Consider, in particular, class **Membership**, to which an OCL constraint is attached, stating that every client's account balance is larger than 0. This class declares three attributes and a method over a state space, identified by variable U below, which is made observable exactly (and uniquely) by the attributes and methods it declares. Concretely,

$$\begin{aligned}
&\mathsf{joined} : Date \longleftarrow U\\
&\mathsf{lastHire} : Date \longleftarrow U\\
&\mathsf{balance} : \mathbb{R} \longleftarrow U\\
&\mathsf{pay} : U \longleftarrow U \times \mathbb{R}
\end{aligned}$$

These four declarations can be grouped in one through a *split* construction

$$\langle \mathsf{joined}, \mathsf{lastHire}, \mathsf{balance}, \overline{\mathsf{pay}} \rangle : Date \times Date \times \mathbb{R} \times U^{\mathbb{R}} \longleftarrow U$$

which is a *coalgebra* for functor

$$\mathsf{T}\, X = Date \times Date \times \mathbb{R} \times X^{\mathbb{R}}$$

Therefore, we write,

$$[\![\mathbf{Membership}]\!] = \langle \mathsf{joined}, \mathsf{lastHire}, \mathsf{balance}, \overline{\mathsf{pay}} \rangle$$

In general, the semantics $[\![c]\!]$ of a class c is given by a specification of a coalgebra

$$\langle \mathsf{at}, \overline{\mathsf{md}} \rangle : A \times (O \times U)^I \longleftarrow U$$

where A is the attribute domain, and each method accepts a parameter, of type I, and delivers both a state change and an output value, of type O. I.e., a coalgebra for functor

$$\mathsf{T}\, X = A \times (O \times X)^I \tag{2}$$

Typically, I and O are *sum* types, aggregating the input-output parameters of each declared method. On its turn, A is usually a *product* type joining all attribute outputs in a way which emphasizes that each of them is available independent of the others, and therefore always able to be accessed in parallel.

More generally, as methods are typically implemented by *partial functions* or even by arbitrary *relations*, a more accurate definition of functor T would be

$$\mathsf{T}\, X = A \times ((O \times X) + 1)^I \tag{3}$$
$$\mathsf{T}\, X = A \times \mathscr{P}(O \times X)^I \tag{4}$$

respectively. Both cases (3) and (4), and many more (for example involving methods whose outcome follows a probabilistic distribution), are subsumed by the following definition

$$\langle \mathsf{at}, \overline{\mathsf{md}} \rangle : A \times \mathsf{B}(O \times U)^I \longleftarrow U \tag{5}$$

where functor T is parametric in a strong monad B[1] capturing some sort of behavioral effect. For example, *partiality* (making $\mathsf{B}X = X + 1$) or *non determinism* (B standing for the finite powerset functor). Additionally, a class may specify some initial conditions, typically as a predicate $\gamma : 2 \longleftarrow U$ which is supposed to hold in the coalgebra initial states.

1 A *strong monad* [8] is a monad $\langle \mathsf{B}, \eta, \mu \rangle$ where B is a strong functor and both η and μ strong natural transformations. B being strong means there exist natural transformations $\mathsf{T}(\mathsf{Id} \times -) : \mathsf{T} \times - \Longleftarrow \mathsf{T} \times -$ and $\mathsf{T}(- \times \mathsf{Id}) : - \times \mathsf{T} \Longleftarrow - \times \mathsf{T}$ called the right and left strength, respectively, subject to certain conditions. Their effect is to *distribute* the free variable values in the context "$-$" along functor B. Strength τ_r, followed by τ_l maps $\mathsf{B}I \times \mathsf{B}J$ to $\mathsf{BB}(I \times J)$, which can, then, be flattened to $\mathsf{B}(I \times J)$ via μ. In most cases, however, the *order* of application is relevant for the outcome. The Kleisli composition of the right with the left strength, gives rise to a natural transformation whose component on objects I and J is given by $\delta r_{I,J} = \tau_{r_{I,J}} \bullet \tau_{l_{\mathsf{B}I,J}}$ Dually, $\delta l_{I,J} = \tau_{l_{I,J}} \bullet \tau_{r_{I,\mathsf{B}J}}$. Such transformations specify how the monad distributes over product and, therefore, represent a sort of sequential composition of B-computations. Whenever δ_r and δ_l coincide, the monad is said to be *commutative* and the unique transformation represented by δ.

Such a coalgebraic setting provides for free a notion of observational equivalence—T-bisimulation, which generalizes to T-shaped transition systems the notion of *bisimulation* found in automata theory or process algebra: a bisimulation is a relation over the state spaces of two coalgebras, p and q, which is *closed* for their dynamics, i.e.

$$(x, y) \in R \;\Rightarrow\; (p\,x, q\,y) \in \mathsf{T}R \tag{6}$$

which, getting rid of variables, becomes the following inequality in the language of the (pointfree) calculus of binary relations [1]:

$$R \subseteq p^\circ \cdot (\mathsf{T}R) \cdot q \tag{7}$$

where p° stands for the relational converse of p. Applying the *shunting* rule of the calculus[2] known as the *shunting rules* [4] on p°, this simplifies to

$$p \cdot R \subseteq (\mathsf{T}R) \cdot q \tag{10}$$

Instantiating definition (6) to functor T, yields two class models being bisimilar iff they provide identical observations through attributes and execution of the method's component not only deliver equal outputs but also makes each of them to evolve to a pair of new states which are also bisimilar.

Formally, given p and q over state spaces U and V, for any $u \in U, v \in V$,

$$\begin{aligned} u \sim v \Leftrightarrow\; & \pi_1 p\, u = \pi_1 q\, v \;\wedge \\ & \forall_{i \in I}\; \mathsf{let}(r, u') = (\pi_2 p\, u) i, (t, v') = (\pi_2 q\, v) i \\ & \quad \mathsf{in}\; r = t \wedge u' \sim v \end{aligned}$$

2.2 Bringing classes to life

Instances of class specifications, being coalgebras for functor T, can be prototyped as transition machines with shape given by the specification of T implicit in definition (5), i.e.,

$$\mathsf{T}X = A \times \mathsf{B}(O \times X)^I \tag{11}$$

Our prototyping framework provides a mechanism for registering and selecting class instances; interaction with each specific instance is done, in a step-by-step mode, through activation of the corresponding $\langle \mathsf{at}, \overline{\mathsf{md}} \rangle$ operation.

But what if one wants to prototype not a single class but a fragment of the whole Class Diagram? In such a case it would be nice to follow a similar mechanism, activating a pair $\langle \mathsf{at}, \overline{\mathsf{md}} \rangle$ combining the dynamics of possible instances of all the classes considered in the fragment of interest. In simpler words, we would like to look at (an instance of) a fragment of a Class Diagram as a coalgebra itself.

2 There are two *shunting rules* [4] as follows

$$f \cdot R \subseteq S \;\equiv\; R \subseteq f^\circ \cdot S \tag{8}$$
$$R \cdot f^\circ \subseteq S \;\equiv\; R \subseteq S \cdot f. \tag{9}$$

This entails the need for specific *composition strategies* for T-coalgebras. In the sequel we discuss three such strategies formalized as combinators over T-coalgebras, borrowing from the authors' previous work on coalgebraic calculi. Reference [5], in particular, introduces a comprehensive calculus of generalized Moore machines framed as coalgebras for a functor similar to (11) which can, to a great extent, be adapted to the present setting to reason about class specifications.

There are two basic ways for combining (instances of) class specifications: *parallel* aggregation, denoted by $\boxtimes$, in which methods in both classes are called simultaneously (as they always act upon disjoint state spaces), and *interleaving*, denoted by $\boxplus$, which offers a choice of which class to call. In both cases, however, attributes are always available to be observed, and therefore are composed in a multiplicative context. Initial conditions are joined by logical conjunction.

Therefore, given coalgebras p and q, over state spaces U and V, respectively, we define their product $p \boxtimes q$ as

$$\langle \gamma_{p\boxtimes q}, \langle \mathsf{at}_{p\boxtimes q}, \overline{\mathsf{md}_{p\boxtimes q}} \rangle\rangle, \tag{12}$$

where

$$\gamma_{p\boxtimes q} = U \times V \xrightarrow{\gamma_p \times \gamma_q} 2 \times 2 \xrightarrow{\wedge} 2$$

$$\mathsf{at}_{p\boxtimes q} = U \times V \xrightarrow{\mathsf{at}_p \times \mathsf{at}_q} A \times A'$$

$$\begin{aligned}\mathsf{md}_{p\boxtimes q} = U \times V \times (I \times I') &\xrightarrow{\mathsf{m}} (U \times I) \times (V \times I') \\ &\xrightarrow{\mathsf{md}_p \times \mathsf{md}_q} \mathsf{B}(O \times U) \times \mathsf{B}(O' \times V) \\ &\xrightarrow{\delta} \mathsf{B}((O \times U) \times (O' \times V)) \\ &\xrightarrow{\mathsf{Bm}} \mathsf{B}((O \times O') \times (U \times V))\end{aligned}$$

where m is an isomorphism (combining Cartesian product commutativity and associativity), and δ is the Kleisli composition of left and right strengths associated to monad B. Interleaving, or *choice*, differs from $\boxtimes$ only in the methods component. Thus,

$$\begin{aligned}\mathsf{md}_{p\boxplus q} = U \times V \times (I + I') &\xrightarrow{\triangle \times \mathsf{id}} (U \times V)^2 \times (I + I') \\ &\xrightarrow{\cong} (U \times I) \times V + (V \times I') \times U \\ &\xrightarrow{f} \mathsf{B}(O \times U) \times V + \mathsf{B}(O' \times V) \times U \\ &\xrightarrow{\tau_r \times \tau_r} \mathsf{B}((O \times U) \times V) + \mathsf{B}((O' \times V) \times U) \\ &\xrightarrow{\cong} \mathsf{B}(O \times (U \times V)) + \mathsf{B}(O' \times (U \times V)) \\ &\xrightarrow{g} \mathsf{B}((O + O') \times U \times V) + \mathsf{B}((O + O') \times U \times V) \\ &\xrightarrow{\nabla} \mathsf{B}((O + O') \times U \times V)\end{aligned}$$

where $f \overset{\text{abv}}{=} \mathsf{md}_p \times \mathsf{id} + \mathsf{md}_q \times \mathsf{id}$ and $g \overset{\text{abv}}{=} \mathsf{B}(\iota_1 \times \mathsf{id}) + \mathsf{B}(\iota_2 \times \mathsf{id})$. On the other hand, $\triangle = \langle \mathsf{id}, \mathsf{id} \rangle$ and $\nabla = [\mathsf{id}, \mathsf{id}]$ denote, respectively, the diagonal and co-diagonal functions (see [4] for details on these definitions and the calculus of functions).

Finally, another tensor, denoted by $\circledast$, corresponds to what may be called *concurrent* composition. It is defined as a combination of the $\boxtimes$ and $\boxplus$, allowing for both parallel or interleaved method execution. Formally, the action of $p \circledast q$ on methods of classes p and q accepts either separated or tupled inputs to deliver the result of applying either $\mathsf{md}_{p\boxplus q}$ or $\mathsf{md}_{p\boxtimes q}$:

$$\mathsf{md}_{p\circledast q} : \mathsf{B}((O + O' + O \times O') \times U \times V) \longleftarrow U \times V \times (I + I' + I \times I')$$

The reader can easily check $\mathsf{md}_{p\circledast q}$ is defined by

$$\mathsf{md}_{p\circledast q} = \mathsf{B}(\mathsf{dl}^\circ) \cdot \delta \cdot (\mathsf{md}_{p\boxplus q} \times \mathsf{md}_{p\boxtimes q}) \cdot \mathsf{dr}$$

where dl and dr stand for product left and right distribution, respectively.

A *wrapping* mechanism is also in order to adapt methods' input/output interfaces. This may be useful, when prototyping a class diagram, for re-arranging arguments or results of the methods declared in the class according to a certain order or to cast or transform such values. Wrapping has no effect on the attribute component. For the methods' component, however, it corresponds to the pre- and post-composition with functions.

Formally, let p be the diagram model whose combined methods' input (respectively, output) interface is I (respectively, O). and consider functions $f : I \longleftarrow I'$ and $g : O' \longleftarrow O$. Model p wrapped by f and g is denoted by $p[f, g]$ and defined by input pre-composition with f and output post-composition with g. Clearly,

$$\begin{aligned}\mathsf{md}_{p[f,g]} = U_p \times I' &\xrightarrow{\mathsf{id} \times f} U_p \times I \xrightarrow{\mathsf{md}_p} \mathsf{B}(U_p \times O) \\ &\xrightarrow{\mathsf{B}(\mathsf{id} \times g)} \mathsf{B}(U_p \times O')\end{aligned}$$

The combinator enjoy the usual properties of a wrapper. For example, for functions $f : I \longleftarrow I'$, $f' : I' \longleftarrow J$, $g : O' \longleftarrow O$ and $g' : R \longleftarrow O'$,

$$(p[f, g])\,[f', g'] \sim p\left[f \cdot f', g' \cdot g\right] \tag{13}$$

because

$$\begin{aligned}&\mathsf{md}_{(p[f,g])[f',g']} \\ \sim\quad &\{\text{wrapping definition}\} \\ &\mathsf{B}(\mathsf{id} \times g') \cdot \mathsf{md}_{p[f,g]} \cdot (\mathsf{id} \times f') \\ \sim\quad &\{\text{wrapping definition}\} \\ &\mathsf{B}(\mathsf{id} \times g') \cdot \mathsf{B}(\mathsf{id} \times g') \cdot \mathsf{md}_p \cdot (\mathsf{id} \times f) \cdot (\mathsf{id} \times f') \\ \sim\quad &\{\times \text{ is a functor}\} \\ &\mathsf{B}(\mathsf{id} \times g' \cdot g) \cdot \mathsf{md}_p \cdot (\mathsf{id} \times f \cdot f') \\ \sim\quad &\{\text{wrapping definition}\} \\ &\mathsf{md}_{p[f \cdot f', g' \cdot g]}\end{aligned}$$

On the other hand, all three forms of composition (corresponding to $\boxtimes$, $\boxplus$ and $\boxast$), combinators are associative as well as commutative, whenever B is a commutative monad. As one would expect, such properties are stated up to bisimilarity.

The proof of commutativity of $\boxtimes$ below illustrates a way to reason, in a calculational style, with coalgebraic definitions. The basic proof technique resorts to the well-known fact that a morphism between coalgebras entails bisimilarity. In this example isomorphism $\mathsf{s} : V \times U \longleftarrow U \times V$ relating the state spaces of classes $p \boxtimes q$ and $q \boxtimes p$, is shown to be a T-coalgebra morphism. The only non trivial part of the proof is the one related to the methods' component, which we detail as follows in a completely pointfree style. Note a swap of the arguments is also necessary, which is achieved by a suitable wrapping:

$$
\begin{aligned}
& a_{q \boxtimes p[\mathsf{s},\mathsf{s}]} \cdot (\mathsf{s} \times \mathsf{id}) \\
= \quad & \{\ \boxtimes \text{ and wrapping definition}\ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_q \times a_p) \cdot \mathsf{m} \cdot (\mathsf{id} \times \mathsf{s}) \cdot (\mathsf{s} \times \mathsf{id}) \\
= \quad & \{\ \mathsf{s} \text{ natural and } \mathsf{s} \cdot \mathsf{m} = \mathsf{m} \cdot (\mathsf{s} \times \mathsf{s})\ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot \mathsf{s} \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \delta_l, \delta_r \text{ interchangeable}\ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \mathsf{Bs} \cdot \delta_r \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \text{routine: } \mathsf{m} \cdot \mathsf{s} = (\mathsf{s} \times \mathsf{s}) \cdot \mathsf{m}\ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{B}(\mathsf{s} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_r \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \mathsf{B} \text{ commutative}\ \} \\
& \mathsf{B}(\mathsf{id} \times \mathsf{s}) \cdot \mathsf{B}(\mathsf{s} \times \mathsf{s}) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \\
= \quad & \{\ \mathsf{s} = \mathsf{s}^\circ, \boxtimes \text{ and wrapping definition}\ \} \\
& \mathsf{B}(\mathsf{s} \times \mathsf{id}) \cdot a_{p \boxtimes q}
\end{aligned}
$$

3 Constraints

Constraints are another essential ingredient of class diagrams. Their semantic effect is to constraint what coalgebras count as valid instances for the class. Consider, for example, constraint

$$\mathsf{balance} > 0$$

attached to class **Membership** in our example.

Constraints are predicates which should be preserved along the life-time of each instance of the corresponding class. Formally, they can be seen as *invariants*. Following the approach recently proposed in [2], such predicates are first encoded as coreflexives, i.e., as fragments of the identity relation, according to

$$y\ \Phi_P\ x \equiv y = x \wedge P\ x$$

Invariance, with respect to coalgebra p, is then expressed as

$$p \cdot \Phi_P \subseteq \mathsf{T}\,\Phi_P \cdot p \tag{14}$$

Applying shunting (8) to (14) leads to

$$\Phi_P \subseteq \underbrace{p^\circ \cdot (\mathsf{T}\Phi_P) \cdot p}_{\bigcirc_p \Phi_P} \tag{15}$$

which brings about a sort of *"next time"* modal operator, holding for those states whose all immediate successors, if any, satisfy Φ_P. Therefore, assertion

$$\Phi_P \subseteq \bigcirc_p \Phi_P \tag{16}$$

is an alternative statement of "Φ_P in an invariant" for coalgebra p.

When reasoning about diagram transformations constraints entail *proof obligations*. For example,

$$
\begin{aligned}
& [\![\mathsf{balance} > 0]\!] = \\
& \quad [\![\textbf{Membership}]\!] \cdot \Phi_{\mathsf{balance}>0} \\
& \qquad \subseteq \mathsf{T}\,\Phi_{\mathsf{balance}>0} \cdot [\![\textbf{Membership}]\!]
\end{aligned}
$$

needs to be discarded whenever justifying a transformation involving class **Membership**.

Proof obligation (14), once instantiated to functor (2), reads

$$\langle \mathsf{at}, \overline{\mathsf{md}} \rangle \cdot \Phi_P \subseteq A \times \mathsf{B}(O \times \Phi_P)^I \cdot \langle \mathsf{at}, \overline{\mathsf{md}} \rangle \tag{17}$$

which reduces, by split fusion and absorption (cf., laws in [4]), to

$$\langle \mathsf{at} \cdot \Phi_P, \overline{\mathsf{md}} \cdot \Phi_P \rangle \subseteq \langle \mathsf{at}, \mathsf{B}(O \times \Phi_P)^I \cdot \overline{\mathsf{md}} \rangle$$

which, by structural equality, is equivalent to the following two conditions:

$$\mathsf{at} \cdot \Phi_P \subseteq \mathsf{at} \tag{18}$$
$$\overline{\mathsf{md}} \cdot \Phi_P \subseteq \mathsf{B}(O \times \Phi_P)^I \cdot \overline{\mathsf{md}} \tag{19}$$

On the other hand, constraints should also be satisfied by the initial conditions, i.e.,

$$\forall_{u \in U} \ .\ \gamma\ u \Rightarrow P\ u$$

which can be expressed in a point-free format as the inclusion of the corresponding coreflexive relations:

$$\Phi_\gamma \subseteq \Phi_P \tag{20}$$

What needs to be proved, however, is that constraints are preserved by the aggregation combinators, $\boxplus$, $\boxtimes$ and $\boxast$. Clearly, for all cases, (20) holds trivially: the effect of each combinator on the initial conditions is their conjunction. Verification of proof obligation (14) is shown for combinator $\boxtimes$, the other cases being similar. Consider, thus, two T-coalgebras $p = \langle \mathsf{at}_p, \overline{\mathsf{md}}_p \rangle$ and $q = \langle \mathsf{at}_q, \overline{\mathsf{md}}_q \rangle$, over state spaces

U and V, respectively. Suppose they correspond to two classes in a Class Diagram whose constraints are specified by predicates P and P', respectively. Conditions (18) and (19) instantiates to

$$\mathsf{at}_{p\boxtimes q} \cdot \Phi_{(P\times P')} \subseteq \mathsf{at}_{p\boxtimes q}$$
$$\overline{\mathsf{md}}_{p\boxtimes q} \cdot \Phi_{(P\times P')} \subseteq \mathsf{B}((O \times O') \times \Phi_{(P\times P')})^{I\times I'} \cdot \overline{\mathsf{md}}_{p\boxtimes q}$$

The first inequality holds because $\Phi_{(P\times P')}$ is a coreflexive. To prove the second one we reason

$$\begin{array}{ll}
& \mathsf{md}_{p\boxtimes q} \cdot (\Phi_{(P\times P')} \times (\mathsf{id} \times \mathsf{id})) \\
= & \quad \{ \text{ definition of } \boxtimes \text{ and } \Phi_{(P\times P')} = \Phi_P \times \Phi_{P'} \} \\
& \mathsf{Bm} \cdot \delta \cdot (\mathsf{md}_p \times \mathsf{md}_q) \cdot \mathsf{m} \cdot ((\Phi_P \times \Phi_{P'}) \times (\mathsf{id} \times \mathsf{id})) \\
= & \quad \{ \mathsf{m} \text{ is a natural transformation} \} \\
& \mathsf{Bm} \cdot \delta \cdot (\mathsf{md}_p \times \mathsf{md}_q) \cdot ((\Phi_P \times \mathsf{id}) \times (\mathsf{id} \times \Phi_{P'})) \cdot \mathsf{m} \\
= & \quad \{ \times \text{ is a functor} \} \\
& \mathsf{Bm} \cdot \delta \cdot ((\mathsf{md}_p \cdot (\Phi_P \times \mathsf{id})) \times (\mathsf{md}_q \cdot (\mathsf{id} \times \Phi_{P'}))) \cdot \mathsf{m} \\
\subseteq & \quad \{ p \text{ (resp., } q\text{) preserves } P \text{ (resp., } P'\text{)} \} \\
& \mathsf{Bm} \cdot \delta \cdot ((\mathsf{B}(O \times \Phi_P) \cdot \mathsf{md}_p) \times (\mathsf{B}(O' \times \Phi_{P'}) \cdot \mathsf{md}_q)) \cdot \mathsf{m} \\
= & \quad \{ \delta \text{ is a natural transformation and } \times \text{ is a functor} \} \\
& \mathsf{Bm} \cdot \mathsf{B}((O \times \Phi_P) \times (O' \times \Phi_{P'})) \cdot \delta \cdot (\mathsf{md}_p \times \mathsf{md}_q) \cdot \mathsf{m} \\
= & \quad \{ \mathsf{m} \text{ is a natural transformation} \} \\
& \mathsf{B}((O \times O') \times (\Phi_P \times \Phi_{P'})) \cdot \mathsf{Bm} \cdot \delta \cdot (\mathsf{md}_p \times \mathsf{md}_q) \cdot \mathsf{m} \\
= & \quad \{ \text{ definition of } \boxtimes \} \\
& \mathsf{B}((O \times O') \times (\Phi_P \times \Phi_{P'})) \cdot \mathsf{md}_{p\boxtimes q}
\end{array}$$

4 Associations and the diagram engine

4.1 Associations

In a UML class diagram *associations* are recorded as arrows with annotations of multiplicities, representing relationships between sets of instances of the involved classes. Being properties of relationships, they can be regarded as their *types* and as *invariants* over a coalgebra representing the whole set of instances of a class diagram. Such a coalgebra, referred in the sequel as the *diagram engine*, is defined over a state space $\mathsf{Pop} \times \mathsf{Assocs}$ where

$$\mathsf{Pop} = \mathscr{P}(\mathsf{Ref})^{\mathsf{ClassId}}$$
$$\mathsf{Assocs} = \mathscr{P}(\mathsf{Assoc})^{\mathsf{AId}}$$
$$\mathsf{Assoc} = \mathsf{ClassId} \times \mathsf{ClassId} \times \mathscr{P}(\mathsf{Ref} \times \mathsf{Ref})$$

where $\mathscr{P}$ denotes the finite powerset functor. We assume classes, class instances and associations have unique identifiers (of types $\mathsf{ClassId}$, Ref and AId, respectively).

A first property common to all associations in a class diagram is the fact of being *total* with respect to the actual sets of instances of the classes involved. That is: no instance can be left out of an association in which its class participates. How can this be expressed?

Actually, many useful properties of relations have simple algebraic formulations,[3] namely resorting to the *kernel* and *image* operators, given by

$$\mathsf{ker} R = R^\circ \cdot R$$
$$\mathsf{img} R = R \cdot R^\circ$$

where R° denotes the converse of relation R. The totality property we are looking for is specified through the requirement that the identity relation is contained in $\mathsf{ker}\, R$. Therefore, let $S = (\rho, \alpha)$ be the state space of the *diagram engine* coalgebra as indicated above (note the type of this coalgebra is still to be discussed). Thus, for all association identifier a, let $\alpha(a) = (c, d, r)$. The association is total iff

$$\mathsf{id}_{\rho(c)} \subseteq \mathsf{ker}\, r \tag{21}$$

Now, the different types of associations can be characterized similarly:

- *one-to-one*: $\mathsf{ker}\, r \subseteq \mathsf{id}_{\rho(c)}$, which charaterizes injectivity, and $\mathsf{img}\, r \subseteq \mathsf{id}_{\rho(d)}$ which specifies r as a simple (i.e., functional) relation. Combining with (21) yields

$$\mathsf{ker}\, r = \mathsf{id}_{\rho(c)} \wedge \mathsf{img}\, r \subseteq \mathsf{id}_{\rho(d)} \tag{22}$$

- *many-to-one*: $\mathsf{img}\, r \subseteq \mathsf{id}_{\rho(d)}$, which combined with (21) specifies r as a (total) function.
- *one-to-many*: $\mathsf{img}\, r^\circ \subseteq \mathsf{id}_{\rho(c)}$ which is equivalent, by duality, to $\mathsf{ker}\, r \subseteq \mathsf{id}_{\rho(c)}$. Together with (21) yields

$$\mathsf{ker}\, r = \mathsf{id}_{\rho(c)} \tag{23}$$

- *many-to-many*: any relation does the job.

A little more contrived is the specification of a m to n association. Multiplicity *at most m* in the source class is captured by

$$\forall_{p \in \mathsf{dom}\, r} \,.\, \#(r \cdot \{p\}) \leq m \tag{24}$$

Similarly, multiplicity *at most n* in the target class is expressed by

$$\forall_{q \in \mathsf{rng}\, r} \,.\, \#(\{q\} \cdot r) \leq n \tag{25}$$

where dom and rng are coreflexive relations obtained by intersecting with the identity relation ker and img, respectively. Note that, as p is a coreflexive pair, composition $r \cdot \{p\}$ corresponds to relation $\{(y, \pi_1\, p) |\ (y, \pi_1\, p) \in r\}$.

[3] See [1] for a detailed account of the pointfree calculus of binary relations.

4.2 The diagram engine

We have characterized associations in a class diagram as properties of binary relations between class instances. This entailed the need for referring explicitly to the sets of class instances as well as to the actual relations between instances. Such sets, indexed by class and association identifiers, as mentioned above, form the state space of a coalgebra—the *diagram engine*—which represents the dynamics of the whole class diagram. In a sense the properties of associations should be regarded as invariants for such a coalgebra.

But what is its shape? As usual, this is determined by the signature of operations available upon it. In other words, the operations which allow us to prototype the diagram and test its behavior. Let U abbreviate type $\mathsf{Pop} \times \mathsf{Assocs}$. At least the following operations must be considered:

– create new instances:
$\mathsf{new} : U \times \mathsf{Ref} \longleftarrow U \times \mathsf{ClassId}$
– remove instances:
$\mathsf{del} : U \longleftarrow U \times \mathsf{Ref}$
– connect a class instance to another in the context of a declared association:
$\mathsf{connect} : U \longleftarrow U \times \mathsf{AId} \times (\mathsf{ClassId} \times \mathsf{Ref})^2$
– disconnect a class instance from an association:
$\mathsf{disconnect} : U \longleftarrow U \times \mathsf{Ref} \times \mathsf{AId}$

which leads to the definition of the *diagram engine* as a coalgebra

$$(U, \delta : (\mathsf{Ref} + 1) \times U)^{IP} \longleftarrow U) \tag{26}$$

where

$$IP = \mathsf{ClassId} + \mathsf{Ref} + (\mathsf{AId} \times \mathsf{ClassId} \times \mathsf{Ref})^2 \\ + (\mathsf{Ref} \times \mathsf{AId})$$

represents the input parameters for the four operations. Eventually, initial conditions can be specified to characterize δ initial valid states (for example, forcing initially all sets of instances to be empty).

Note that properties of associations, as discussed in the previous section, can be given as invariants for coalgebra δ. In rigor, however, there are a number of *unstable* states in δ which may fail to verify such properties: for example, after the creation of a new instance and before its addition to the relevant associations. From a prototyping point of view, this is achieved by forcing the *diagram engine* to execute from a state which violates such almost invariants, until it comes to a state in which they hold. This basically means that on creating or removing a class instance, the coalgebra is not observed until the associated operations of connecting or disconnecting terminate. Such is the strategy adopted in our HASKELL proof-of-concept implementation.

5 Conclusions

In this paper we define a comprehensive coalgebraic semantic model for UML class diagrams coping with classes, their composition, constraints and associations. A diagram engine to capture the global diagram dynamics is also framed coalgebraically. A number of combinators are defined corresponding to different ways of combining class specifications. Constraints, as well as associations, are formally characterized as invariants over a coalgebra representing the corresponding class or class diagram specifications. Furthermore, such invariants are shown to be preserved by the combinators. The coalgebraic specification of a class diagram is captured by the diagram engine, which has been instantiated in a prototype developed in HASKELL.

Prospects for future work include the investigation of other features present in class diagrams, such as generalization, as well as the planning of significative case studies to assess empirically the merits of the approach proposed here. We are also interested in simulation of system model behavior. In addition, another natural follow up of this paper concerns *refinement* of class diagrams, extending previous work (e.g., in [3,12]) to take into account constraints and associations.

Acknowledgments This research was partially supported by a grant from the GLANCE funding program of NWO, through project CooPer (600.643.000.05N12), as well as by FCT (the Portuguese Foundation for Science and Technology) under contract `PTDC/EIA/73252/2006`.

References

1. Backhouse RC, Hoogendijk PF (1993) Elements of a relational theory of datatypes. In: Möller B, Partsch H, Schuman S (eds) Formal program development, pp 7–42. Springer lecture notes in computer science, vol 755
2. Barbosa LS, Oliveira JN, Silva AM (2008) Calculating invariants as coreflexive bisimulations. In: Meseguer J, Rosu G (eds) Proceedings of the 12th international conference on algebraic methodology and software technology, AMAST 2008, Urbana, IL, USA, July 28–31, 2008, pp 83–99. Springer lecture notes in computer science, vol 5140
3. Barbosa LS, Sun M (2008) UML model refactoring as refinement: a coalgebraic perspective. In: Negru V, Jebelean T, Pectu D, Zaharie D (eds) Proceedings of GlobalComp, at 10th SYNASC, 26–29 September 2008, Timisoara, Romania, pp 340–347. IEEE Computer Society, USA
4. Bird R, Moor O (1997) The algebra of programming. Series in computer science. Prentice-Hall International, Englewood Cliffs
5. Cruz A, Barbosa L, Oliveira J (2005) From algebras to objects: generation and composition. J Univers Comput Sci 11(10):1580–1612
6. Hoogendijk PF (1996) A generic theory of datatypes. PhD thesis, Department of Computing Science, Eindhoven University of Technology
7. Jacobs B, Rutten J (1997) A tutorial on (co)algebras and (co)induction. EATCS Bull 62:222–259
8. Kock A (1972) Strong functors and monoidal monads. Archiv für Mathematik 23:113–120

9. Object Management Group (2007) Unified Modeling Language: Superstructure, version 2.1.1, http://www.uml.org/
10. Rutten J (2000) Universal coalgebra: a theory of systems. Theor Comp Sci 249(1):3–80. (Revised version of CWI Technical report CS-R9652, 1996)
11. Sun M, Aichernig BK, Barbosa LS, Naixiao Z (2005) A coalgebraic semantic framework for component based development in UML. In: Birkedal L (ed) Proceedings of the international conference on category theory and computer science (CTCS'04), vol 122. Electronic notes in theoretical computer science. Elsevier, Amsterdam, pp 229–245
12. Sun M, Barbosa LS (2006) Components as coalgebras: the refinement dimension. Theor Comput Sci 351:276–294
13. Sun M, Barbosa LS (2008) A coalgebraic semantic framework for reasoning about UML sequence diagrams. In: Zhu H (ed) Proceedings of the eighth international conference on quality software, QSIC 2008, 12–13 August 2008, Oxford, UK. IEEE Computer Society, USA, pp 17–26
14. Sun M, Naixiao Z, Barbosa LS (2004) On semantics and refinement of UML statecharts: a coalgebraic view. In: Cuellar J, Liu Z (eds) Proceedings of 2nd IEEE international conference on software engineering and formal methods, Beijing, China, September 2004. IEEE Computer Society Press, USA, pp 164–173