Toward a Framework Constraint Language

Shui Ming Ho

Department of Computer Science
University of Manchester
Manchester M13 9PL, U.K.
sho@cs.man.ac.uk

Abstract. In this paper we are concerned with expressing precise semantics for one particular kind of component: the object-oriented design framework as described in the component-based development methodology Catalysis. Currently, the Unified Modelling Language is used to model such frameworks and their semantics are expressed precisely in the Object Constraint Language. Although both languages are used, they fail to fully address the nature of Catalysis frameworks. The Object Constraint Language, in particular, is unable to express some of the behavioural aspects of these frameworks. This paper describes a framework modelling language that addresses the inadequacies of these two languages.

1 Introduction

Frameworks are increasingly recognised as better units of reuse and design than single objects, helping to shift the the base of object-oriented technology from objects to components larger than single objects. In this paper we are concerned with expressing semantics for such components: frameworks as defined in the component-based development methodology Catalysis [4]. Traditional object-oriented frameworks describe the relationships between (abstract) classes of objects, their interactions and division of responsibility. These classes and their relationships may be modelled using class diagrams and interaction diagrams of the Unified Modelling Language (UML) [11]. Precision may be added to these diagrams by constraining the way in which objects behave, expressing invariants on an object's state and preconditions and postconditions on their methods using the Object Constraint Language (OCL) [13].

Generic designs, however, are obtained by focusing on the roles objects play in different frameworks rather than assigning objects to specific classes. Role modelling techniques (e.g., [8]) partition different views of objects across several models. Similarly, in Catalysis, it is assumed that no one framework describes an object in its entirety. Frameworks describe partial objects: complete objects emerge when frameworks are composed with one another. At the heart of Catalysis frameworks is the assumption that the state of an object may be altered as a consequence of its participation in other frameworks. Although the exact nature of these *external* actions may not be known beforehand, a framework may impose constraints on these actions.

As an example of a Catalysis framework, Fig. 1 shows the structural and behavioural relationships between objects within a trade-and-supply framework. The TradeSupply framework in Fig. 1 is a template with placeholders Retailer, Distributor, Order, and Product, representing the roles objects may play in the framework. The framework also contains the fixed class Date. Just as objects may have methods, frameworks may have *joint* actions describing collaborative behaviour between objects, e.g., *makeOrder* in Fig. 1. Such actions are indicated using an extended UML notation.

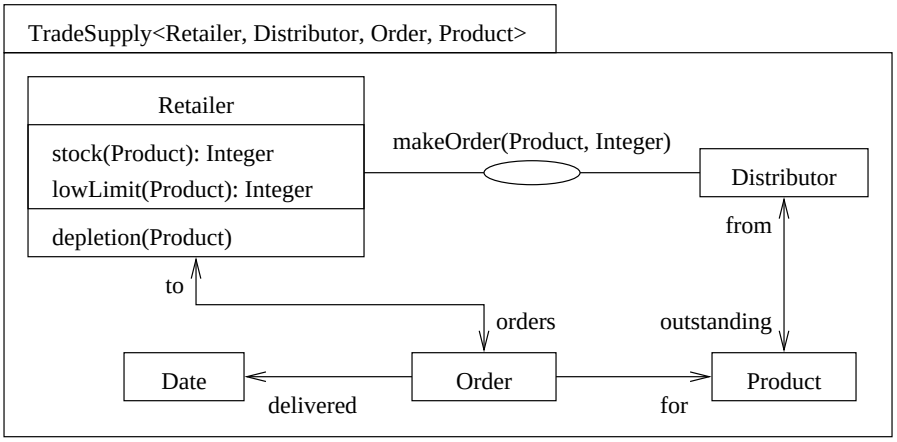

Fig. 1. The TradeSupply framework.

Not shown in Fig. 1 are the constraints of the framework: specifications for *makeOrder* and constraints on the external action *depletion*. The *goal* of TradeSupply is to ensure that a retailer's stock of products is not exhausted. This may be accomplished by constraining the effects of *depletion* (currently using OCL) such that any action that corresponds to *depletion* invokes *makeOrder* if the stock is below some given limit. OCL, however, is limited in the types of constraint it can express and where they may be applied. In Catalysis, informal extensions to OCL are used to express the semantics of joint and external actions. Much work exists on extending OCL, e.g., to express temporal properties of systems and the flow of messages between objects. In [6, 7], for example, OCL is extended with an action clause and [10] extends OCL with temporal operators. These extensions require that their semantics are consistent with UML: OCL is not a stand-alone language and must always accompany a UML diagram [12]. Both UML and OCL, however, have informally defined semantics. The problem of giving formal semantics for OCL has been addressed in [3], [5], and [9], and work on semantics for UML includes [1]. Expressing Catalysis frameworks and their composition, however, requires extensions to both UML and OCL. The work presented in this paper is based on an alternative approach to tackling the

problem of framework modelling and expressing their precise semantics: defining a small modelling language that is able to capture the core structural and behavioural aspects of frameworks while also being able to deal with the framework composition mechanisms of Catalysis not present in UML.

2 The $\mathcal{FCL}$ Modelling Language

Figure 2 shows make-up of frameworks and classes in $\mathcal{FCL}$. A framework introduces a number of class definitions, associations and operation symbols. These three components of a framework are illustrated in Fig. 1. A class definition

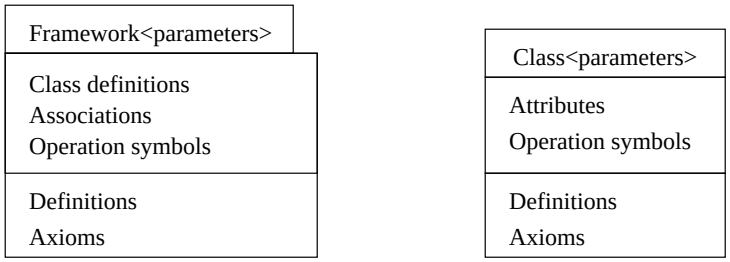

Fig. 2. Frameworks and classes.

introduces a number of attributes and operations. In $\mathcal{FCL}$ both attributes and associations are relations and operations do not have return types. The usual object-based dot notation is used in $\mathcal{FCL}$: for an object o with an attribute a the expression $o.a(x)$ is true if the pair (o, x) is in the extension of the predicate a. In the case where a is intended to be a function the existential quantifier **one** ($\exists!$) may be used to constrain a, e.g.,

$$\textbf{all}\ o \mid (\textbf{one}\ x \mid o.a(x))\,.$$

Operations and definitions. $\mathcal{FCL}$ allows the definitions of named formulae both within classes and at the framework level. One of the ways they can be used is to define separately the preconditions and postconditions of operations. In this respect $\mathcal{FCL}$ differs from OCL but is similar to UML, The semantics of the joint action *makeOrder* in Fig. 1, for example, may be described by the formula

$$\begin{aligned}&\textbf{all}\ r : \text{Retailer},\ d : \text{Distributor},\ p : \text{Product},\ n : \text{Integer} \mid \\ &\quad \text{makeOrder}(r, d, p, n)\ \textbf{implies} \\ &\qquad (\text{makeOrder-pre}(r, d, p, n)\ \textbf{implies}\ \text{makeOrder-post}(r, d, p, n)).\end{aligned}$$

External actions. In $\mathcal{FCL}$ trigger rules describing how a framework reacts to a change of state from any source, whether it is internal to the framework or external, may be documented. In the TradeSupply framework we wish to state that if a retailer's stock of some product is lowered then this state change corresponds to the *depletion* operation. The formula

$$\begin{aligned}&\textbf{all } r : \text{Retailer},\ p : \text{Product},\ x, y : \text{Integer} \mid \\ &\quad r.\text{stock}'(p, x)\ \&\ r.\text{stock}(p, y)\ \&\ x > y\ \textbf{implies}\ r.\text{depletion}(p)),\end{aligned}$$

where $'$ denotes an attribute/association in the next state, expresses this. The above $\mathcal{FCL}$ formula represents the temporal formula below.

$$\begin{aligned}(\forall r, p, x, y)\ \ &(\text{Retailer}(x) \wedge \text{Product}(p) \wedge \text{Integer}(x) \wedge \text{Integer}(y) \\ &\Rightarrow (\bigcirc\text{stock}(r, p, x) \wedge \text{stock}(r, p, y) \wedge x < y \Rightarrow \text{depletion}(r, p))).\end{aligned}$$

Both Retailer and Product are *class predicates*. Analogous to the "isKindOf" operator in OCL, the term "Retailer(x)" is true if x is an object of the class Retailer.

The following formula expresses the condition that external actions corresponding to *depletion* must observe.

$$\begin{aligned}&\textbf{all } r : \text{Retailer},\ p : \text{Product},\ x, y : \text{Integer} \mid \\ &\quad r.\text{depletion}(p)\ \&\ r.\text{stock}'(p, x)\ \&\ r.\text{lowLimit}'(p, y)\ \&\ x < y \\ &\qquad \textbf{implies}\ (\textbf{exists } d : \text{Distributor},\ n : \text{Integer} \mid \text{makeOrder}(r, d, p, n)).\end{aligned}$$

3 Framework Composition

A VendorSupply framework may be derived by composing the earlier TradeSupply framework and the PublicVending framework in Fig. 3. In Catalysis, frameworks are *template packages*. They are "composed" by being imported into others. Importing under Catalysis differs from UML: in Catalysis, elements from imported packages may be renamed and merged together. The fragment of $\mathcal{FCL}$ below specifies a renaming of the roles of Vendor and Retailer to Shop. The definitions of both these classes are merged together. Thus, Shop is characterised by the attributes, operations, definitions, and axioms of Vendor and Retailer.

imports PublicVending ⟨Vendor/Shop⟩; TradeSupply ⟨Retailer/Shop⟩;

Renaming may also be applied to individual class attributes, operations, and definitions to prevent names from coinciding.

4 Underlying Framework Specification

$\mathcal{FCL}$ provides an object-oriented notation for modelling frameworks. Underlying every $\mathcal{FCL}$ specification, however, is a specification of the framework in temporal

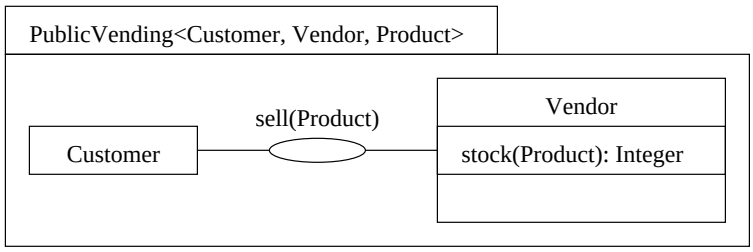

Fig. 3. The PublicVending framework.

logic. Whereas in $\mathcal{FCL}$ frameworks are organised around classes, this is not the case at the temporal logic level. At this underlying level, a framework $\mathcal{F}(\Pi) = \langle \Sigma, X \rangle$ consists of a signature Σ, parameters Π, and axioms X. The signature consists of a single sort $\mathcal{O}$, the constant symbols of which denote object identities. In this section the mapping from $\mathcal{FCL}$ to first order signatures is considered.

Classes. In $\mathcal{FCL}$ associated with each class C is a class predicate and a number of parameters, attributes, operations, and definitions. A class predicate C is mapped to an equivalent unary predicate symbol, C, in Σ. Each attribute α, operation symbol σ, and parameter π of class C is mapped onto unique symbols, here denoted by α_C, σ_C, and π_C — classes have unique attributes, operations, and definitions, the declarations of which implicitly introduce their own axioms. For example, associated with a parameterised attribute a, declared as $a(S_1, \ldots, S_n) : S$, is an axiom

$$(\forall i, x_1, \ldots, x_n, y : \mathcal{O})$$
$$(a_C(i, x_1, \ldots, x_n, y) \Rightarrow C(i) \wedge S_1(x_1) \wedge \ldots \wedge S_n(x_n) \wedge S(y)).$$

For parametric classes, their associated parameter symbols appear in Π. In the case of classes which act as placeholders of the framework, their class predicate, attribute, and operation symbols are all framework parameters.

Framework composition. The "imports" statement in $\mathcal{FCL}$ describes a mapping function from the signature of the imported frameworks to that of the importing framework, extended to map the axioms from one framework to another. The signature of VendorSupply, for example, is derived from the signatures obtained by applying morphisms to the signatures of TradeSupply and PublicVending, mapping Retailer to Shop in the former, and Vendor to Shop in the latter. Similarly, the set of axioms of VendorSupply is the union of the sets of axioms obtained by extending the morphisms to terms.

5 Summary

In this paper an overview of $\mathcal{FCL}$, a framework modelling language, was presented, the motivation of which is to address the shortcomings of UML and OCL

when applied to Catalysis frameworks. Specifically, this text has focused on the use of $\mathcal{FCL}$ to describe the behavioural aspects of frameworks, such as external actions, but also framework composition. The purpose of $\mathcal{FCL}$, however, is to provide an object-oriented notation for framework specification, concealing an underlying non-object-based temporal logic specification of the framework.

References

1. T. Clarke, A. Evans, and S. Kent. The Metamodelling Language Calculus: Foundation Semantics for UML. In *Fourth International Conference on Fundamental Approaches to Software Engineering*, volume 2029, pages 13–21, Genova, Italy, 2001. Springer-Verlag.
2. S. Cook, A. Kleppe, R. Mitchell, J. Warmer, and A. Wills. Defining the Context of OCL Expressions. In *Second International Conference on the Unified Modelling Language, Lecture Notes in Computer Science*, volume 1723, Colorado, USA, 1999. Springer-Verlag.
3. D. Distefano, J. P. Katoen, and A. Rensink. On a Temporal Logic for Object-based Systems. In S. F. Smith and C. L. Talcott, editors, *Formal Methods for Open Object-based Distributed Systems IV*, pages 305–326. Kluwer Academic Publishers, September 2000.
4. D. D'Souza and A. Wills. *Objects, Components, and Frameworks with UML.* Addison-Wesley, 1998.
5. R. Hennicker, H. Hussmann, and M. Bidoit. On the Precise Meaning of OCL Constraints. In *Object Modeling with the OCL, Lecture Notes in Computer Science*, volume 2263, pages 69–84. Springer-Verlag, 2002.
6. A. Kleppe and J. Warmer. Extending OCL to Include Actions. In *UML 2000, Lecture Notes in Computer Science*, volume 1939, pages 440–450. Springer-Verlag, 2000.
7. A. Kleppe and J. Warmer. The Semantics of the OCL Action Clause. In *Object Modeling with the OCL, Lecture Notes in Computer Science*, volume 2263, pages 213–227. Springer-Verlag, 2002.
8. T. Reenskaug et al. *Working with Objects.* Manning/Prentice-Hall, 1995.
9. M. Richters and M. Gogolla. OCL: Syntax, Semantics, and Tools. In *Object Modeling with the OCL, Lecture Notes in Computer Science*, volume 2263, pages 42–68. Springer-Verlag, 2002.
10. S. Sendall and A. Strohmeier. Specifying Concurrent System Behaviour and Timing Constraints Using OCL and UML. In *UML 2001, Lecture Notes in Computer Science*, volume 2185, pages 391–405. Springer-Verlag, 2001.
11. *UML Specification V1.4 draft (ad/01-02-14)*, February 2001.
12. M. Vaziri and D. Jackson. Some Shortcomings of OCL, the Object Constraint Constraint Language of UML. Technical report, Massachusetts Institute of Technology, December 1999.
13. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1999.