Snapshots Generation via Constructive Logic

Mario Ornaghi, Camillo Fiorentini and Alberto Momigliano

Università degli Studi di Milano
{ornaghi,fiorenti,momiglia}@dsi.unimi.it

1 Introduction

A software information system S allows users to store, retrieve and process information about the external world, typically in the form of a data base. We can differentiate two separate aspects in the data elaborated by S: the first concerns *data types*, while the second is related to the *information on the external "real world"* carried by the data. More precisely, a data type is a set of data together with the associated manipulations where the focus is on *operations*. In contrast, the other information carried by the data stored in S is strongly related to their *meaning in the real world*. The need for properly treating data according to their meaning is becoming increasingly important, due to the wide quantity of information exchanged on the Internet [5].

In this context, we are developing COOML [4] (Constructive Object Oriented Modeling Language), an OO specification language where the focus is on the information carried by data. The main features of COOML are:

– a *data model* based on a predicative extension of the constructive intermediate logic E^* [3], following the Brower-Heyting-Kolmogorov interpretation of logical connectives;
– a multi-layered structure, starting with the COOML logic layer, on top of which lies the problem domain logic, and finally the computation layer;
– a data model based on the notion of "pieces of information" $i : P$, where i is a structured information ("information value") giving constructive evidence for the truth of a specification P.

We believe that COOML may have a role as an OO specification framework, and we are developing different prototypical tools – this work is in progress and more details are available at the COOML web page `http://homepages.inf.ed.ac.uk/amomigl1/cooml`. In this abstract, we focus on automatic snapshots generation [1]: in modeling languages (we concentrate on the UML [2] as the de-fact standard) a snapshot represents a system state (object diagram) and snapshots generation in the presence of OCL constraints has proved to be useful both for understanding a specification and verifying its consistency in the problem domain. In our approach, snapshots are represented by the pieces of information and our snapshots generation algorithm is driven by the constructive content of COOML specifications.

2 COOML Data Model

A piece of information $i : P$ for a specification P represents a justification of the truth of P; we can consider the information values i for P as the *structured data* that are needed to justify the truth of P in terms of the truth of its sub-formulas. Let us consider, for example, the specification $P_{item} \equiv [\forall i : Item(i)](\neg inCatalog(i) \vee \exists c\, cost(c,i))$, where $Item(i)$ means that i is an object of class *Item*, *cost* and *inCatalog* have the obvious meaning and $[\forall i : Item(i)]$ is a bounded quantification. To justify P_{item}, we have to produce the set of *Item* objects and, for each of them, we have to justify the corresponding sub-formula. This kind of information is codified by information values such as i_{item} : `((it1 (1 true))(it2 (2 (15.50 true))))`.

The piece of information $i_{item} : P_{item}$ shows that the *Item* objects are `it1` and `it2`, that for `it1` the first disjunct $\neg inCatalog(\texttt{it1})$ holds, and for `it2` the second disjunct $\exists c\ cost(c, \texttt{it2})$ holds with $c = 15.50$. This piece of information may be true or false in a *world-state*, namely an interpretation of the problem domain formulas $Item(\ldots)$, $inCatalog(\ldots)$, and $cost(\ldots)$. Problem domain formulas may be complex: the COOML logical layer takes them as *atoms*, and reasoning on them is delegated to the *problem domain logic PDL*. In this abstract, we assume that *PDL* is an axiomatisation based on classical logic, and world-states correspond to classical interpretations. We use a separate syntax to distinguish the (constructive) COOML logical layer from the underlying *PDL*. The syntax of a COOML specification P is inspired by Java, in particular $\underline{\tau}\ \underline{x}$ denotes a list of distinct variables of appropriate type $\underline{\tau}$.

$$\begin{array}{lcl} AT & ::= & PF \mid \Box P \\ P & ::= & AT \mid \text{AND}\{P \ldots P\} \mid \text{OR}\{P \ldots P\} \mid \text{EXI}\{\underline{\tau}\ \underline{x} : P\} \mid \text{FOR}\{\underline{\tau}\ \underline{x} \mid G(\underline{x}) : P\} \end{array}$$

COOML's *atoms (AT)* consist of arbitrary problem domain formulas PF and $\Box$-formulas. The latter serve the purpose of embedding classical truth in our constructive setting [3]. In our language, as in JML and OCL, universal quantification is bounded. The *generator* $G(\underline{x})$ is a particular *problem formula*, true for finitely many ground instances $\underline{c}_1, \ldots, \underline{c}_m$ of closed terms; we call them the *terms generated by* $G(\underline{x})$.

We now address the semantics of atoms. The only information on a world-state w carried by a ground problem formula $A\sigma$ is the elementary information value *true*. It simply means that $A\sigma$ *holds in* w. The truth of a $\Box$-formula is defined interpreting the COOML connectives as ordinary logical ones. For a specification P, the *information type* $\text{IT}(P)$ of P is defined as follows, where information values are lists built starting from the primitive data types of the problem domain:

$$\begin{array}{rcl} \text{IT}(A) & = & \{true\}, \text{ where } A \text{ is an AT} \\ \text{IT}(\text{AND}\{P_1 \ldots P_n\}) & = & \{(i_1, \ldots, i_n) \mid i_j \in \text{IT}(P_j), 1 \leq j \leq n\} \\ \text{IT}(\text{OR}\{P_1 \ldots P_n\}) & = & \{(k, i) \mid 1 \leq k \leq n \text{ and } i \in \text{IT}(P_k)\} \\ \text{IT}(\text{EXI}\{\underline{\tau}\ \underline{x} : P\}) & = & \{(\underline{c}, i) \mid \underline{c} : \underline{\tau} \text{ and } i \in \text{IT}(P)\} \\ \text{IT}(\text{FOR}\{\underline{\tau}\ \underline{x} \mid G : P\}) & = & \{((\underline{c}_1, i_1), \ldots, (\underline{c}_m, i_m)) \mid \\ & & m \geq 0 \text{ and, for } 1 \leq j \leq m, \underline{c}_j : \underline{\tau} \text{ and } i_j \in \text{IT}(P)\} \end{array}$$

A specification P gives meaning to the information values belonging to $\text{IT}(P)$. A *piece of information* is a pair $i : P$, with $i \in \text{IT}(P)$. For every ground substitution σ, the *meaning of* $i : P\sigma$ in a world-state w is given by the relation $w \mid\models i : P\sigma$ ($i : P\sigma$ *is true in* w):

$$\begin{array}{rcl} w \mid\models true : A\sigma & \text{IFF} & A\sigma \text{ holds in } w, \text{ where } A \text{ is an AT} \\ w \mid\models (i_1, \ldots, i_n) : \text{AND}\{P_1 \ldots P_n\}\sigma & \text{IFF} & w \mid\models i_j : P_j\sigma, \text{ for all } j = 1, \ldots, n \\ w \mid\models (k, i) : \text{OR}\{P_1 \ldots P_n\}\sigma & \text{IFF} & w \mid\models i : P_k\sigma \\ w \mid\models (\underline{c}, i) : \text{EXI}\{\underline{\tau}\ \underline{x} : P(\underline{x})\}\sigma & \text{IFF} & w \mid\models i : P(\underline{c})\sigma \\ w \mid\models L : \text{FOR}\{\underline{\tau}\ \underline{x} \mid G(\underline{x}) : P(\underline{x})\}\sigma & \text{IFF} & (\underline{c} \in dom(L) \text{ iff } G(\underline{c})\sigma \text{ holds in } w) \\ & & \text{and } ((\underline{c}, i) \in L \text{ entails } w \mid\models i : P(\underline{c})\sigma) \end{array}$$

3 Class Specifications in COOML

In COOML we introduce classes via class predicates for the problem domain. For example, we can specify a simple "cash-register" OO system (more details in [4]) by the following COOML classes:

Class CashRegister {
CashRegisterPty: OR{ EXI{ Obj !*receipt* : *receipt*.*Receipt*(*this*); }
The receipt is empty; }
}

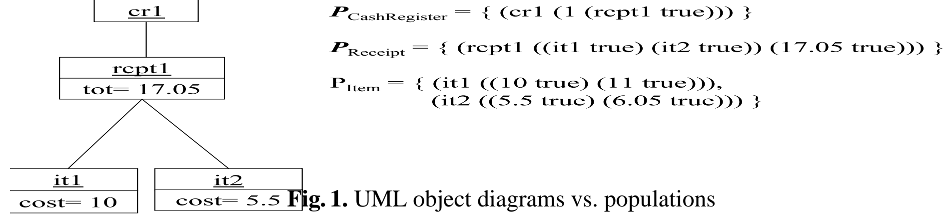

Fig. 1. UML object diagrams vs. populations

```
Class Receipt {
ENV{ Obj cash | this.Receipt(cash) : true;}
ReceiptPty: AND{ FOR{ Obj item | item.Item(this) : item.inCatalog(); }
                 EXI{ float total : total = grandtotal(this) } }
}
Class Item {
ENV{ Obj receipt | this.Item(receipt) : this.inCatalog();}
ItemPty: AND{ EXI{ float cost : cost = cost(this) }
              EXI{ float price : price = price(this); } }
 /*@ ensures \result = cost + cost*VAT/100 @*/
float price();
}
```

For the sake of conciseness, the underlying *PDL* is omitted (the problem domain formulas have the obvious meaning or can even be informal). In COOML, each class definition is interpreted by a *class axiom* **ClassAx**(C), which specifies the information content of C, and a *constraint axiom* **ConstrAx**(C), which is an atom specifying the constraints on the environment variables declared in the ENV-section. For example, the above classes are interpreted according to the following axioms, where **CPty**($\underline{x}$) abbreviates the corresponding formula of the class definition:

ClassAx(CashRegister)	FOR{ Obj *this* | *this.CashRegister*() : **CashRegisterPty**(*this*)}
ConstrAx(Receipt)	□(FOR{Obj *this cash* | *this.Receipt*(*cash*) : *true*; })
ClassAx(Receipt)	FOR{ Obj *this cash* | *this.Receipt*(*cash*) : **ReceiptPty**(*this*, *cash*)}
ConstrAx(Item)	□(FOR{Obj *this receipt* | *this.Item*(*receipt*) : *this.inCatalog*(); })
ClassAx(Item)	FOR{ Obj *this receipt* | *this.Item*(*receipt*) : **ItemPty**(*this*, *receipt*)}

Assume that $\mathcal{P}_C$: **ClassAx**(C) is the piece of information of a class axiom **ClassAx**(C). In fact, $\mathcal{P}_C$ is a (possibly empty) list of information values of the form $((o\ \underline{t})\ i)$, where o instantiates *this* and $\underline{t}$ is a tuple of terms instantiating the environment variables. We call $\mathcal{P}_C$ a *population of class C* and treat it as a set. The population $\mathcal{P}$ of an OO system is the union of the populations of its classes. We say that an object o belongs to the population $\mathcal{P}$ iff there is an information value $((o\ \underline{t})\ i)$ in $\mathcal{P}$. A population $\mathcal{P}$ is finite (an OO system has a finite set of objects) and each object o of $\mathcal{P}$ occurs in a unique information value $((o\ \underline{t})\ i)$ in $\mathcal{P}$ (an object belongs to an OO system in a unique copy). We identify system states with populations, and we define the semantics of system states as follows:

Definition 1. *Let $\mathcal{P}$ be a population for $\mathcal{S}$ and w a world-state. Then $w \mid\models \mathcal{P} : \mathcal{S}$ iff:*

(i) $w \mid\models \mathcal{P}_C$: **ClassAx**(C) *for every class C of $\mathcal{S}$, where $\mathcal{P}_C$ is the population of class C;*
(ii) *Every constraint axiom A of* **Ax**($\mathcal{S}$) *holds in w.*

In our approach, an OO system $\mathcal{S}$ is consistent iff it has a consistent population $\mathcal{P}$, and $\mathcal{P}$ is consistent iff there is at least a world-state w such that $w \mid\models \mathcal{P} : \mathcal{S}$.

Populations are similar to UML object diagrams [2]. This correspondence is illustrated in Fig. 1. In general, a population contains more information than the corresponding UML

snapshots and, more importantly, the pieces of information contained in it are related in a precise logical way to the problem domain. Formally:

Theorem 1. *Let $P\sigma$ be an instance of a COOML specification and w be a reachable world-state of the problem domain. Then: (a) $P\sigma$ holds in w iff there is an information value $i \in \mathrm{IT}(P)$ such that $w \mid\models i : P\sigma$; (b) for every $j \in \mathrm{IT}(P)$, we can extract a set $\mathcal{A}(j, P\sigma)$ of ground atoms from j such that $w \mid\models j : P\sigma$ iff $\mathcal{A}(j, P\sigma)$ holds in w.*

4 Snapshots Generation and Consistency

Theorem 1 can be used for snapshot generation and consistency checking as follows. Using the recursive definition of $\mathrm{IT}(P)$, we can generate information values for the purpose of automatic snapshots generation in two stages. In the first one, we generate a partially defined information value $p(\underline{x}) \in \mathrm{IT}(P)$, where object names are instantiated and the other data are denoted by suitable variable symbols $\underline{x}$, related to the existential and universal quantifiers of P. In the second stage, the atoms $\mathcal{A}(p(\underline{x}), P\sigma)$ are extracted and their consistency is checked. Of course, in general consistency is not decidable. However we can check certain properties using one of the existing tools or formalising the former (if possible) in a suitable Horn theory. We give an idea of the algorithm for generating the information values with a Prolog-like pseudo-code. We only show the definition of the main predicate *info*, where $info(P^+, Info^-, L^+, NL^-)$ takes a specification P and generates an information value *Info* for P, using L and NL as auxiliary data structures:

$info(A, true, L, L) \;\leftarrow\; isAtom(A).$
$info(and\{F_1 \ldots F_n\}, [I_1, \ldots, I_n], L_0, L_n) \;\leftarrow\; \bigwedge_{j=1}^{n} info(F_j, I_j, L_{j-1}, L_j).$
$info(or\{F_1 \ldots F_n\}, [k, I], L, NL) \;\leftarrow\; info(F_k, I, L, NL). \quad k = 1, \ldots, n$
$info(exi\{T : F\}, [V, I], L, NL) \;\leftarrow\; choose(V, T, L, L1) \wedge info(F, I, L1, NL).$
$info(for\{T\, X \mid G : F\}, FI, L, NL) \;\leftarrow\; generate(X, T, G, Dom, L, L1) \wedge map(F, Dom, FI, L1, NL).$

The predicate $choose(V, T, L, L1)$ chooses a value for V according to the type T. Similarly, the predicate $generate(X, T, G, Dom, L, L1)$ generates a domain *Dom* for the generator formula G. Finally, $map(F, Dom, FI, L1, NL)$ computes $[I_{v_1}, \ldots, I_{v_n}]$, where $v_1, \ldots, v_n$ are the elements of *Dom*.

One of the advantages of using a constructive approach to OO modeling is that the semantics itself, namely the notion of piece of information, directly supports the *declarative* generation and validation of system snapshots. Moreover, in contrast with model based approaches, information values can be shaped at different levels of granularity. For example, we can use $\mathrm{EXI}\{\tau\; x : \Box\mathrm{EXI}\{\tau\; y : r(x, y)\}\}$ to indicate that the information value has to contain a witness for x, but not for y. This opens the door to applying constructive logical methods to express and formally reason about more intensional system properties.

References

1. M. Gogolla, J. Bohling, and M. Richters. Validation of UML and OCL Models by Automatic Snapshot Generation. In P. Stevens, J. Whittle, and G. Booch, editors, *UML'03*, volume 2863 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2003.
2. C. Larman. *Applying UML and Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.
3. P. Miglioli, U. Moscato, M. Ornaghi, and G. Usberti. A constructivism based on classical truth. *Notre Dame Journal of Formal Logic*, 30(1):67–90, 1989.
4. M. Ornaghi, M. Benini, M. Ferrari, C. Fiorentini, and A. Momigliano. A constructive object oriented modeling language for information systems. *Proceedings of CLASE 2005, ENTCS*, 2005. To appear, available at `http://homepages.inf.ed.ac.uk/amomigl1/papers/cooml.pdf`.
5. A. Sheth. DB-IS research for Semantic Web and enterprises. Brief history and agenda. LSDIS Lab, Computer Science, University of Georgia, 2002. `http://lsdis.cs.uga.edu/SemNSF/Sheth-Position.doc`.