Data and Activities: Exploiting Hierarchies of Classes

Alexander T. Borgida
Sol Greenspan

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

1. Introduction

We wish to briefly discuss recent work in conceptual modelling from a slightly different point of view in order to highlight the parallels between data and transactions,and then mention some benefits of this view.

A time-honoured way of describing a system (portion of the world) is by positing a domain of objects and then inter-relating them through function and predicate symbols. The resulting description is a set of axioms in a FOPC. If the world is dynamic, one usually augments the description with the notion of time or state, in which case axioms can be divided naturally into "general laws" (heretofore constraints) holding in all states, and state-specific "facts". Given states, one then also has the ability to describe state transitions (events) as predicates on pairs of states or, as shown below, as objects in their own right.

In practice, problems arise when describing large, detailed systems which need not be logically complex but have a multitude of functions and constraints (e.g., data bases). The first problem is that many functions do not and should not have values for most objects in the domain. In order to avoid the extensive use of *undefined* as a value, it seems advantageous to group objects into classes (sorts, types) - denoted here in capital letters - to which we attach the general constraints on functions defined over the elements of the class. This is one example of a useful abstraction technique, namely classification/instantiation. Note that a class can also be viewed as an object and hence can belong to the range and domain of functions and predicates, and can itself be classified.

In the case that there is significant overlap between the descriptions of many classes, these can be organized in a partial order (IS-A hierarchy) where each class is described by adding details to the description of its immediate super-classes. Equivalently, one can require that if $\mathcal{D}$ IS-A $\mathcal{B}$ then the axiom $(\forall x)(x\in\mathcal{D} \Rightarrow x\in\mathcal{B})$ be present; in this way the class "inherits" all the constraints on the higher classes and we can only add new constraints. This is a second abstraction technique: generalization/specialization.

Such ideas have recently gained wide acceptance in the description of data semantics ([1]) and, as we shall see, are useful in describing events as well.

2. Data and event classes

For each data class, the membership predicate $\in$ describes its instances in a state and, especially in modelling data bases, one is only interested in the objects currently in the class. In order to make description easier, one provides primitive state transitions such as *INSERT* and *DELETE*, plus necessary axioms, in order to specify "changes" to $\in$. The functions describing the elements of a data class are often known as attributes (properties), and the constraints express restrictions on the range of functions, such as its type, or invariants (integrity constraints). For convenience again, *UPDATE* is provided to specify changes to attribute values.

For example, objects in the class *VEHICLE* have properties *type*, *reg#*,... and constraints like $(\forall x\in VEHICLE)(type(x) \in \{bike,car,truck,...\})$, $(\forall x,y\in VEHICLE)(reg\#(x)=reg\#(y) \Rightarrow x=y)$,... .

On the other hand, one can think of a transaction (procedure, process) description as an event class, with particular invocations of the transaction being the instances of the class. (This is the sense in which an event is an object.) Furthermore, the functions applicable to the elements of an event class present the participants in the event (parameters and local variables for procedures). Again, constraints express restrictions on the range of participant functions, such as the type of the range, invariants or pre- and post-requisites.

For example, in describing *TO-LEASE*, the act of leasing a vehicle to a person, we may have participants *leasee* and *object*, with constraints $(\forall z\in TO\text{-}LEASE)(lease(z)\in PERSON\wedge object(z)\in VEHICLE)$, prerequisite $(\forall z\in TO\text{-}LEASE)(age(leasee(z))>16)$, and a post-requisite stating that there must be a new RENTAL-AGREEMENT form when z terminates.

Alternatively, the methodology of step-wise decomposition can be accommodated by describing the "components" of an event. In this view, some functions applicable to instances of an event class have events as value, and the constraints specify the type (event class) of each of these, constraints on the order of invocation of components, and relationships between the participants of the event and its components.

For example, *TO-LEASE* could have parts *get-vehicle* and *fill-in-agreement*, and for z in *TO-LEASE* we may want *get-vehicle*(z) to remove *object*(z) from the class *FREE-VEHICLE*; if *RETRIEVE* was the class of events which removed their participant *what* from the set *from-where*, this could be stated as *get-object*(z)=y∈*RETRIEVE*, *what*(y)=*object*(z) and *from-where*(y)=*FREE-VEHICLE*.

3. Specialization hierarchies

Turning now to specialization, if D IS-A C then we want all instances of D to be instances of C. Therefore, in order to get consistent descriptions, the properties and constraints applicable to C must also apply to D; thus the strict view of specialization requires that in defining D we can only state additional constraints or describe new properties and related constraints.

For example, a *TRUCK* is a *VEHICLE* and will have additional constraint (∀x∈*TRUCK*)(*type*(x)= truck) and possibly new property *tonage* which has numeric value.

The same pattern applies when specializing an event class. On the one hand we can place additional constraints on existing participants (e.g., more prerequisites) and/or add new participants and related constraints. On the other hand, we can strengthen constraints on existing component events and/or add new components and relate them to the others.

For example, *LEASE-A-TRUCK* is a *TO-LEASE* and for any instance z, *object*(z)∈*TRUCK*, *age*(*leasee*(z))>*21* and the postrequisite would state that the rental form would need to have an insurance rider. Also, z would have an additional component, *buy-insurance*(z), which added the rider to the agreement.

Note that since instances of a class must also be instances of all its superclasses, consistency in description requires that a specialized event cause at least the same "changes" to any state as its more general counterpart. This requirement is explicitly presented above for pre- and postrequisites, but is harder to describe (and implement) for components. It is this strong sense of specialization that is present in TAXIS [2], but is missing from SIMULA for example, and which we intend to exploit.

Finally, we remark that in the presence of a multitude of constraints, it would seem sensible to group these into classes of their own, and then organize them into an IS-A hierarchy. A class of assertions would then have free variables and one of its instances would replace such free variables by constants (e.g. (∀x,y∈Z)(*name*(x)=*name*(y) => x=y) could be instantiated by replacing Z by some class such as *MY-NEIGHBORS*, for which *name* is likely to be one-to-one. The IS-A hierarchy for constraints would be based on implication. Note in passing that by going to a second-order system, the functions used in descriptions could also be structured along the lines drawn above.

To summarize, we have argued that data, events and constraints can all be structured using aggregation (the functions), classification and specialization, and in addition they can be inter-related, as illustrated by the participants of an event or the assertions of a data class.

4. Some applications

To begin with, among the important gains from the preceding discussion is a reasonably small set of uniform tools that can be used to describe both the static and dynamic aspects of a system. In particular, requirements specification (r.s.) is one area where it would be useful to have facilities for describing data, activities and assertions [6]. This view is supported by at least one prominent r.s. technique, Structured Analysis [3], where data and activities co-exist. Clearly, constraints also form an important and central part of specification. The presence of all 3 types of objects allows us, among other things, to increase the redundancy in our specifications, which is acknowledged to be a positive feature. For example, the integrity constraint for a data class could also be attached to those events which have participants in that data class.

Inherent in the description of section 3 is the methodology of description by specialization [4], where one starts from a general class, about which little is said, and then introduces details incrementally by creating more and more specialized subclasses. This methodology complements refinement by decomposition while allowing, in this framework, focus to shift between the 3 kinds of objects. By forcing the user to consider the superclass and its constraints before describing the subclass, our notion of strict specialization reduces the chances of introducing inconsistencies into the specification, which is one of the important issues in r.s.

Avoiding overspecification is another desirable goal of a specification technique. As an example, within our framework the relative ordering of event activations, specified by assertions, need not be completely given in any particular case. Specialization of an event can introduce new constraints that result in stricter orderings, but this need not reach the stage of a linear ordering.

As a second application, if we wish to prove theorems about transactions in our design, then the IS-A hierarchy provides a natural way of decomposing the verification proofs into smaller, more manageable parts, allowing for sharing of work. The decomposition is natural because the underlying classes are defined and organized according to our conceptual world model. To illustrate this sketchily, consider a language which provides a few structuring primitives, such as loops and conditionals, in addition to the machinery described above, and for which we have a Hoare-style logic or, preferably, a weakest precondition predicate

transformer wp (i.e., $wp(A,P)$ is the weakest condition whose truth guarantees that condition P will hold after the execution of the program A).

One application of this would be to prove that the constraints on data classes are not violated by the transactions defined. Let A be a transaction and suppose that we have proven for some formula that $I \Rightarrow wp(A,I)$. Suppose now that B is a specialization of A and we wish to show that $I \Rightarrow wp(B,I)$. This is already true if B has no components in addition to those of A, by our strict definition of IS-A. If B does have additional components, B' say, and, as is often the case, the relative ordering of A and B' is not significant (i.e., B=A;B' because for example, each alters different "variables") then it is sufficient to prove $I \Rightarrow wp(B',I)$ in order to conclude that $I \Rightarrow wp(A;B',I)$.

Similarly, transaction A could be characterized by its pre- and post-requisite constraints p and q on the parameters of A; we would then want to show that the components of A combine to meet the above specification i.e., $p \Rightarrow wp(A,q)$. Transaction B would be characterized by $p\&p'$ and $q\&q'$, because one can only strengthen constraints in specialization, and arguments like the one above indicate that proving $q \Rightarrow wp(B',q)$ and $p\&p' \Rightarrow wp(A;B',q')$ is sufficient to establish $p\&p' \Rightarrow wp(A;B',q\&q')$.

Finally, suppose we describe the system in two steps: first, in a higher, non-procedural specification language [6] and then in a lower implementation language. In the specification language an event E is specified as $R|E|S$ where R and S are initial and final conditions expressing the effect of E. Each event E is implemented by a transaction T, with its own pre- and post-requisites p and q, as well as components T. In order to show that the specification is correctly implemented, we could follow an ALPHARD-like technique (see [5]). This requires a realization mapping π which defines the value of every symbol in the specification language in terms of the lower one; one must then prove that invariants are maintained, that p and q correctly describe T, that the initial conditions of E ensure the invocation of T, $(\pi(R)\&I \Rightarrow p)$, and that the post-requisites of T establish the final condition of E, $(\pi(R)\&I\&q \Rightarrow \pi(S))$. We have shown already how the first two steps are aided by the IS-A hierarchy. If E' is a specialization of E, then its initial and final conditions R' and S' must be at least as strong as those of E (i.e. $S' \Leftrightarrow S\&S''$ and $R' \Leftrightarrow R\&R''$ for some R'', S''). Similarly, if the implementation T' of E' is a specialization of T then its pre- and post-requisites p' and q' must also be stronger (i.e. $p' \Leftrightarrow p\&p''$ *and* $q' \Leftrightarrow q\&q''$). But then in proving that the description of T' satisfies that of E', it is sufficient to show that the truth of R' implies that of p'' and that $R'\&q'$ ensures S'', because the rest of the proof is the same as proving E to be implemented by T.

The important thing to realize in the above examples is that these savings will be effected repeatedly since one transaction will normally have many specializations. For example, there are numerous subclasses of *VEHICLE* (trucks, boats, buses,...) and of *PERSON* (teenager, adult, over-70, ...) and for each pair *TO-LEASE* may be specialized.

Hopefully the above examples have given the reader an appreciation of some of the potential benefits of organizing both data and events along dimensions such as classification and specialization.

Acknowledgements We are indebted to John Mylopoulos, Harry Wong and Paolo Paolini for a number of thought-provoking discussions.

References

[1] Smith, J.M. and Smith, D.C.P., "Database Abstraction: Aggregation and Generalization", ACM-TODS 2,2 - 1977.

[2] Mylopoulos, J., Bernstein, P. and Wong, H.K.T., "A Language Facility for Designing Database-Intensive Applications", ACM-TODS 5,2 - 1980.

[3] Ross, D.T., "Structured Analysis - A Language for Communicating Ideas", IEEE Trans. Soft. Eng., SE-3,1 - 1977.

[4] Wong, H.K.T., Ph.D. dissertation, University of Toronto (forthcoming)

[5] Wulf, W.A., London, R.L. and Shaw, M., "An Introduction to the Construction and Verification of ALPHARD Programs", IEEE Trans. Soft. Eng. SE-2,4 - 1976.

[6] Greenspan, S., Ph.D. dissertation, University of Toronto (forthcoming)

This work was supported by the Natural Sciences and Engineering Research Council of Canada.