Different Types of Inheritance

Mike Jackson
School of Computing and Information Technology
University of Wolverhampton
35-49 Lichfield Street
Wolverhampton WV1 1EQ
UK
Email: mj@wlv.ac.uk

The primary aim of this short paper is to remind the database community that the issue of inheritance has been debated extensively in the past and that very early papers on semantic data models distinguished between the model of inheritance necessary for semantic data models as opposed to that used by object-oriented programming languages. It is therefore not surprising, that Date and Darwen in their book "Foundation for Future Database Systems" [Date and Darwen 2000] examine the inheritance mechanisms commonly found in object-oriented programming languages and find them to be lacking in the context of database systems. It poses the question, "is a new inheritance mechanism necessary or are the mechanisms proposed in the past for semantic models sufficient if only they could be rediscovered?". The conclusion reached is that the database world may have to resign itself to supporting two models of inheritance, one that permits extensible typing and one that can be used to show associations amongst data.

Inheritance in programming languages

The concept of inheritance was introduced into mainstream Computer Science via the language Simula 67 [Nygaard and Dahl 1978]. Although only single inheritance was implemented in Simula, the basic idea is the one that appears in all modern OO programming languages. It is important, in this context, to understand why new features are added to programming languages. Computer science theory tells us that it is not possible to create new functionality by adding new constructs to a language. Given a bare machine the best we can hope for is that the machine has a capability equal to that of a Turing machine since a Turing machine distinguishes between those algorithms that can be implemented and those that cannot. When a programming language is constructed it does not extend the power of the machine, instead its purpose is to make life easier for those that have to program the machine. Consequently, since the dawn of programming, machine code programmers have criticised high-level languages because they provide a limited subset of the techniques available to a low-level programmer. Despite these criticisms high-level languages have proved popular because they make it possible to construct working software in a shorter timescale than would be possible using machine level coding.

Amongst the many ways in which a high-level language can improve the reliability of a piece of software is the concept of "strong typing". Strong typing is an idea introduced into programming languages in order to reduce the number of errors occurring at run-time. In a programming language with strong typing the compiler can identify potential sources of error and prevent the programmer from including them in the final program. This sounds ideal until one realises that strongly typed languages are very restrictive indeed. Whilst they may help to produce reliable software, the set of programs that may be written within the restrictions of strong typing is much smaller than the set of reliable programs.

An assembler programmer might reason about the problem in this way. A piece of data d_1 is represented by a string of bytes. Some code is written which given an address at which d_1 is located, manipulates the data using byte offsets to address it. A new piece of data d_2 is defined which contains data that has the same structure as d_1 and some additional bytes of data appened

to it. In assembly language programming the code that is used to process d_1 could also be used to process d_2. In a strongly typed language, however, d_1 and d_2 have different types and therefore separate code has to be written in order to manipulate them.

Inheritance (and object-oriented programming) provides a language mechanism whereby a variable may conform to more than one type. Using inheritance a programmer may have the safety of strong typing but retain the flexibility allowed in loosely types languages.

If indeed the motivation for inheritance in programming languages is the one described above it seems unlikely that such a concept is useful in the database arena.

Inheritance in Semantic Models

Prior to the emergence of object-oriented databases, the difference between inheritance in data models and that in object-oriented programming was clearly recognised. Hull and King [Hull and King 1987] distinguish between the concept of inheritance in semantic data models as opposed to the one used in object-oriented programming languages. Interestingly, they remark "semantic models encapsulate structural aspects of object, whereas object-oriented languages encapsulate behavioural aspects of objects". The argument presented here is that Date and Darwen use reasoning appropriate to semantic modelling and try to apply it to object-oriented programming.

In semantic modelling, the idea of class is not the same as the class concept used in object-oriented programming. A class in a semantic model is equivalent to a collection class in an OO schema. In this context, inheritance is something different from the idea found in OO programming languages. In a semantic model if a class Y inherits from class X then class Y contains a subset of the objects in class X. In this context it is perfectly natural to say that Circle inherits from Ellipse since the set of all ellipses is the superset of the set of all circles. This is at variance with Stroustrups's statement "in most programs a circle should not be derived from an ellipse or an ellipse derived from a circle" [Stroustrup 97], however it is the exception that proves the rule as Stroustrup is talking about the construction of programs and not semantic models.

Conclusion

The database community has used the word "inheritance" for a number of years. Unfortunately, it has not used it consistently and this has caused much confusion. Originally the term was used in the context of semantic modelling; latterly it has been use in the context of OO programming. In reality, the community needs both usages. It needs the semantic modelling usage to build better data models and the OO programming usage to manufacture extended data types. Unfortunately, natural language, unlike programming languages has yet to master overloading!

References

Date, C. J. and Darwen (2000), "Foundation for Future Database Systems", Ed. 2, Addison-Wesley, 2000, ISBN: 0-201-70928-7, 2000.

Hull, R. and King, R. (1987), "Semantic Database Modelling: Survey, Applications, and Research Issues", ACM Computing Surveys, Vol. 19, No. 3, September 1997, pp 201-259.

Nygaard, K. and O.-J. Dahl (1978), "The development of the Simula languages."Foundation History of Programming Languages Conference, vol. 13, no. 8 ACM SIGPLAN Notices, 1978.

Stroustrup, B. (1997), "C++ Programming Language", Ed. 3, Addison-Wesley, ISBN: 0-201-88954-4, 1997.