TAKEUCHI83

Ikuo Takeuchi, Hiroshi Okuno, and Nobuyasu Ohsato, "TAO---A harmonic mean of Lisp, Prolog and Smalltalk", Musashino Electrical Communication Laboratory Nippon Telegraph and Telephone Public Corporation, 3-9-11, Midoricho, Musashino, Tokyo, Japan 180, SIGPLAN Notices, V18 #7, July 1983 This paper presents a new language, Tao, which attempts to unify procedural, functional, logic and object-oriented programming. The authors say that their language is a "harmonic mean" of Lisp, Prolog and Smalltalk as opposed to an "arithmetic mean." One supposes this translates to: "there's better and worse ways of combining the features of these languages, and we've chosen one of the better ways." Tao is to be the "kernel language" of Nue, a complete programming environment. Tao's intended audience, then, must be the same as Nue's: professional programmers developing large scale programs in a highly interactive manner. Lisp. The authors believe that the most important feature Lisp has to offer is its basic data structure, the list, and its notation, the s-expression. The authors believe that the best features of Lisp are related to what the s-expression, which they refer to as Lisp's pre-language. Programs and data are both represented by s-expressions. The s-expression has both the s-expressions's internal representation and external syntax are simple and standard. The internal representation allows the garbage collector to worry about storage management, instead of the user. The internal representation also means that both the high and low levels of the system can be accessed by the user. The minimal syntax allows the maximum freedom in attaching semantics. Tao includes must more than just Lisp's s-expression: the rest of the language is thrown in as well. Prolog. Prolog variables and Lisp variables are not the same. Tao does not make any attempt to unify them. Instead, they are syntactically marked as separate. Prolog variables can be partially instantiated. Tao has Prolog versions of Lisp's_and_, _or_ and _not_ primitives. In Lisp, _and_ and _or_ have a procedural aspect to them. _and_ takes any number of arguments and returns their logical conjunction. But the arguments might have side effects. What a Lisp-and does is evaluate its arguments from left to right. As soon as it finds an argument that evaluates to nil, lisp-false, it doesn't bother to evaluate the rest of the arguements. Prolog-and works in the same way, but with an added twist. When Prolog-and encounter and argument that it can't instantiate, prolog-false, it backtracks to the last choice it made (within certain limitations) and tries this choice another way. Smalltalk. Perhaps Nue, which doesn't yet exist, will mimic Smalltalk's windows. Currently, what Tao has to offer in the way of object-oriented programming is a limited version of the Zetalisp Flavors package. As might be already apparent from the structure of this reviews, Tao does not present a new style of programming that has good qualities of three old styles. Instead, these three old styles are simply set side by side. To put it another way, a Tao language manual would be as lo g as the sum of the lengths of the original three languages, plus additional pages to describe how these three parts interact. The Lisp and Prolog sub-languages, for example, must use distinct sets of variables. One operator is called a "hinge" because it connects together these two otherwise disjointed parts. The introduction and conclusion are a fun combination of modern technology and Japanese folk-religion as filtered through the three authors. There's illustrations as well.