Of Types and Prototypes:

The Treaty of Orlando

WHEREAS among the purported benefits of object oriented technology are flexibility and dynamicism, the vast majority of existing object oriented languages have type hierarchies which require conformance to strictly hierarchical ideals and impose penalties on innovation;

WHEREAS these issues have a brief but intense history of discussion and debate: [Lieberman 86] was a reaction to these problems of traditional models, proposing an alternative object sharing mechanism based on the idea of delegation — or forwarding of messages — to replace the traditional implicit inheritance mechanisms, and suggesting that modeling shared behavior through objects representing prototypes could replace the traditional model of abstract classes; [Ungar 87] reacted to the complications engendered by traditional class and inheritance mechanisms by proposing a mechanism also based on prototypes, which model did not propose explicit delegation, but which also shared — through "dynamic inheritance" — the essential characteristics of allowing dynamic sharing patterns and attaching idiosyncratic behavior to individual objects, resulting in a drastic simplification of the Smalltalk object model, using the PARENT link to more flexibly serve for the more complex class/subclass/instance protocol; [Stein 87] attempted a rapprochement between the delegation and inheritance views by noticing that the class/subclass relationship already captured delegation, if a shift was made to a class-only representation rather than Lieberman's instance-only representation; and in unifying classes and instances, extended the class-instance relationship to allow idiosyncratic "instances," providing individual object behavior and more dynamic sharing patterns than traditional inheritance models;

WHEREAS these, our proposals, have helped to elucidate the following three independent dimensions along which this sharing mechanism — which some call delegation and others inheritance, but which is in any case fundamental to object oriented systems — can be examined, namely: *FIRST*, whether *STATIC* or *DYNAMIC*: When does the system require that the patterns of sharing be fixed? Static systems require determining the sharing patterns by the time an object is created, while dynamic systems permit determination of sharing patterns during runtime, when an object actually receives a message. *SECOND*, whether *IMPLICIT* or *EXPLICIT*: Does the system have an operation that allows a programmer to explicitly direct the patterns of sharing between objects, or does the system do this automatically and uniformly? Explicit delegation (or inheritance) allows the ability to delegate only a single method, rather than "anything that can't be handled locally." and *THIRD*, whether *PER OBJECT* or *PER GROUP*: Is behavior specified for an entire group of objects at once, as it is with traditional classes or types, or can idiosyncratic behavior be attached to an individual object? Conversely, can behavior be specified/guaranteed for a group?

and *WHEREAS* there exist a spectrum of programming situations ranging from large production programs where efficiency and reliability are the primary criteria, and which change relatively slowly, to exploratory research programming, which demands flexibility and rapid response to change:

RESOLVED, that different programming situations call for different combinations of these features: for more exploratory, experimental programming environments, it may be desirable to allow the flexibility of dynamic, explicit, per object sharing; while for large, relatively routine software production, restriction to the complimentary set of choices — strictly static, implicit, and group-oriented — may be more appropriate;

RESOLVED, that a second, and separate, fundamental mechanism can be distinguished: the class–instance, or generator, relationship, in which one object (the "class") determines the type of the objects it generates, and that although this relationship has always in the past been associated with static, implicit, per group systems, these restrictions are neither necessary nor inherently desirable, and it is time to free the class–instance relationship from its traditional subservience and consider it as a separate and equal partner in the building of object oriented systems;

RESOLVED, that as systems follow a natural evolution from dynamic and disorganized to static and more highly optimized, the object representation should also have a natural evolutionary path; and that the development environment should itself provide more flexible representations, together with tools — ideally automatic — for adding those structures (of class, of hierarchy, and of collection, for example) as the design (or portions thereof) stabilizes;

and *RESOLVED*, that this agreement shall henceforth be known as the *TREATY OF ORLANDO*.

Henry Lieberman
Massachusetts Institute of Technology

Lynn Andrea Stein
Brown University

David Ungar
Stanford University

6 October 1987
Orlando, Florida, USA

[Lieberman 86]	Lieberman, H., Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems, OOPSLA'86, October 1986.
[Stein 87]	Stein, L. A., Delegation Is Inheritance, OOPSLA'87, October 1987.
[Ungar 87]	Ungar, D. and Smith B., Self: The Power of Simplicity, OOPSLA'87, October 1987.