The Success, Failure, and Future of Distributed Objects

FOOF position statement

Jonathan Aldrich, Carnegie Mellon University

Abstract:

Thirty years ago Emerald proposed a compelling vision of a distributed object-based future, in which objects with identity and state could migrate seamlessly from one host to another [BHJ+87]. While Emerald improved on many of its successors, I argue that neither Emerald's specific choices for maintaining consistency and availability in the face of failure, nor any set of fixed choices, will be sufficient for use in modern distributed applications. Emerald and other distributed languages have implicitly assumed that some fixed choice must be made, and as a result, distributed objects as a language construct have not yet gone mainstream. I believe distributed object languages have a future, but we must find a way to support a diverse set of connection and failure semantics within a language infrastructure. Some choices of semantics may also be incompatible with advanced features of languages such as Emerald; in particular, guaranteeing consistency and availability when migrating stateful objects is problematic.

Additional discussion:

Like many others, I find Emerald's model of distributed objects to be compelling [JLHB88]. The system provides a pure model of objects with identity and mutable state, whose methods can be invoked in a transparent way either locally or remotely. Furthermore, an object can be moved from one node to another, retaining its identity in the process if it is mutable. I thus find it disappointing that this attractive model has not become mainstream, nearly 30 years later, and I wonder why it has not. Is it an idea that is still promising but biding its time? Or is there a fundamental flaw with the Emerald model? If the latter, can it be fixed, or should we give up hope of an elegant distributed object model in this tradition?

Several possible sources of trouble are suggested by Waldo et al.'s Note on Distributed Computing [WWWK94]. Their basic argument is that distributed computing has fundamental differences in latency, memory access, partial failure, and concurrency--differences so significant that unifying the computing models results in unacceptable compromises. Of the four arguments, I find 3 unconvincing. Memory access, in the sense of accessing local memory resources that are not objects, is simply irrelevant to high-level, pure object-oriented languages. Latency is certainly important, but it requires awareness of distribution at the design level, not a distinction between local and remote objects in the object model. Finally, concurrency and asynchronous method invocation are common elements of mainstream object-oriented languages today.

That leaves partial failure. Emerald assumes a "local area network with a modest number of nodes (e.g., 100)" [JLHB88] and relies on exhaustive search to find lost objects, an approach that would not work on a wide-area network [Jul]. Indeed, it is possible that an object is entirely lost (e.g. if the node it was on crashed), in which case the object reference is replaced with nil, which would seem likely to result in a run-time error if operations are subsequently invoked on the object. Losing objects may be acceptable for some distributed system applications, but may not be acceptable for others. More problematic, but probably easier to fix, is the limitation to local area networks.

Brewer's CAP theorem, stating that no distributed system can simultaneously provide consistency, availability, and partition tolerance [Bre00], implies there can be no perfect solution to the problem of partial failure. A major challenge for a distributed object language is that distributed systems vary widely in their requirements for the CAP properties. If any given approach is selected and built into the language, it will be inappropriate for many distributed systems. Worse, an ultra-large-scale system may be a confederation of multiple cooperating distributed nodes with no central administration point; in this setting, there may not be any possible way to choose a single point in the CAP space that would be used by the entire connected system [Fei06].

Given the reality of partial failures and the diversity of distributed systems, I believe that a viable distributed object language will have to support a diverse set of failure-handling semantics that can be customized from one application to another. Furthermore, it will likely be important to allow different nodes within a federated system and/or different applications running on a single set of nodes to use different failure semantics.

There are particularly interesting challenges with migrating stateful objects with identity, because one naturally wants to preserve the consistency property that an object exists at one or the other location, and is not duplicated or lost. By the CAP theorem, any practical system will have to give up migration of stateful objects (copying immutable objects is still safe), accept the possibility that objects may be lost (inconsistency), or accept a loss of availability or partition tolerance.

References

[Bre00] Eric A. Brewer. Towards Robust Distributed Systems. PODC Keynote, 2000.

[Fei06] Peter H. Feiler et al. Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute, 2006.

[JLHB88] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the Emerald system. ACM Trans. Computer Systems 6(1):109-133, 1988.

[WWWK94] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A Note on Distributed Computing. Sun Microsystems Laboratories Technical Report, 1994.