

Friends

It is impossible for us to know the initial conditions in sufficient detail to know if there is deterministic order in our universe. The impossibility of perfect knowledge of initial conditions is a fundamental aspect of what it means to be human. Thus, as a practical matter, the question becomes: how do we deal safely with a world of which we have imperfect knowledge and which we can not treat as deterministic, whether it is or is not?

It is now generally accepted that we are moving away from our "industrial," clockwork, past. The associated organizing principles, the architecture and building blocks, which were used to build its foundations, are not necessarily the best ones for our future. Before we go much further, we need to consider our choices for the architecture and building blocks for the "Information Technology" driven future we are building willy-nilly.

The current Y2K problem is an example of how well thought out current choices can come back to hurt you at a latter time. In the 1950s it is alleged that only two columns for dates were allocated in the design of 80 column paper punch cards. At the time, and with traditional risk management tools, it seemed to make perfect sense. The problem isn't that two additional columns should have been specified. The real problem is that the system was so highly specified at a very early stage, especially its data structures, that it was brittle. As the Y2K issue is proving, brittle systems are almost impossible to adopt to unforeseen circumstances or new requirements. Of course we must understand that alternatives were perceived to be too costly, especially when considered in the short term. The architecture of the system, unfortunately for us, did not support adaptable bindings to allow for adjustments for changes in context or requirements.

This is the real issue with Y2K: the current conventional early binding design has almost zero evolutionary competence. Can we imagine that a 1954 Ford automobile could be expected to anticipate the requirements of an auto in 2010?

It is not an accident that more than a few computer scientists estimate that the 'conventional early binding design' we are currently using as a foundation, as typified by Microsoft/Intel/Unix/Mac systems, is a very poor choice. They may allow a wide range applications to run on top of them, but they themselves are very hard to 'rebind'. They are watchmaker's solutions. As Bob Frankston writes, late binding makes sense when you observe that "there are many situations in which we can and must change our bets during the race." Is it wise to bet on hard-to-change systems when we live in an unpredictable and ever changing world?

David Reed writes: "What we are looking for is structure that gives strength and coherence while preserving the ability to change. Adaptable structure. The late-binding we are talking about is late-bound design structure." David also understands that we need the same sorts of tools to manage the risks associated with development of new ideas. We

need a way to create incentives to take risks and to overcome the limits of accounting systems that create negative incentives. To this end, David is exploring 'real options' as a more effective approach.

An architecture much more like the Internet, which, excepting for the definitions of a packet, is "minimally committed" would most likely be a far better foundation for our future. The strength of the Internet is that at any point in time, one can "specify" its state, but the architectural implementation anticipates and embraces evolution by committing to as little as possible. We must be careful that we preserve this aspect of the internet. We must not enshrine the first order approximations it began with as 'essential' and immutable. For example, is the DNS system, [Domain Name Servers] the best solution we can think of today? Would we be able to build a much more robust Internet on a 128-bit name space rather than the 32 bits we currently use?

More robust architectures and systems, an alternative branch of the computer science evolutionary tree, have been known and developed in the computer science profession for some time. David Reed tell me that these more robust architectural approaches have names like "object oriented systems," "reflective systems," "late binding" systems, "information hiding" modularity, etc. Typical of these systems are the programming languages Lisp, Smalltalk and Self. In the database area, late binding concepts include "views" and "object oriented databases". In the computer security area, these concepts include the "principle of least privilege". And in the networking area, a primary concept is the so-called "end-to-end design principle".

In the late 70s, for example, Intel went so far as to build a very interesting chip, the 432. For a number of reasons, this was a grand failure. In fact, the commercial failure of the Intel 432 is one of the reasons the alternative branch of computer science got a bad name. The 432 were, however, was not a particularly good exemplar of a practical and advanced implementation of the ideas mentioned above.

A great example of these ideas is the Smalltalk virtual machine, which was elegantly executed on the particular hardware state of the art a few years ago in a project called SOAR - Smalltalk on A RISC. Smalltalk's conceptual base continues to evolve in an "open source" community under the name Squeak.

A simple search of the Internet, "432 Intel" for example, will show that the Japanese and others continue with research in this area. Informants trying to educate me also suggest listening for terms such as "a tagged architecture tightly integrated in hardware and software" or "systems which can reconfigure themselves on the fly".

Another way to look at the question of early vs. late binding, Frankston writes, is to see the bindings as tentative. It's not so much early/late as much as binding early but retaining the ability to rebind – to change our bets in the middle of the race. As Isenberg points out, this suggests both loose and tight binding as well.

It may be worth noting that "conventional early binding designs," as typified by Microsoft/Intel/Unix/Mac systems, have little in them that support the ideas mentioned above. They do, however, have many design features that actually get in the way of the more robust architectural approaches.

The challenge we face today, then, is to build the best possible foundation for the Information Technology driven, post-industrial society which is coming like it or not. Are we responsibly and carefully evaluating our alternatives for architectures and building blocks? To put the question in another way: Would a culture built on a late binding architecture with the ability to rebind effectively enjoy strategic advantages in competition with cultures relying upon current conventional early binding design, as typified by Microsoft/Intel/Unix/Mac systems?

Regards,

Jock

For more on these ideas, you might want to read two essays by Bob Frankston:

An essay on operating systems:

http://www.frankston.com/public/Writings/OS_Essay.asp

Originally titled Operating Systems: relics of the past, and also his Y2K comments:

<http://www.frankston.com/public/Essays/Y2Ketal.asp>.

PS: I wish to give proper credit to David P. Reed for his many contributions, suggestions and edits. This short essay would not have been possible without his patient help. Additional help was received from Bob Frankston, David Isenberg, and Ed Gerck.

JPG