Social Software

by Joel Slayton

Software need not be tied exclusively to components alone. It would appear that software is, to some degree, shaped by the sub-cultures of data relations from which they are composed.

> "When two or more organisms interact recursively as structurally plastic systems, ... the result is mutual ontogenic structural coupling... For an observer, the domain of interactions specified through such ontogenic structural coupling appears as a network of sequences of mutually triggering interlocked conducts... The various conducts or behaviors involved are both arbitrary and contextual. The behaviors are arbitrary because they can have any form as long as they operate as triggering perturbations in the interactions; they are contextual because their participation in the interlocked interactions of the domain is defined only with respect to the interactions that constitute the domain. I shall call the domain of interlocked conduits a consensual domain." (Humberto Maturana)

It is, of course, possible to consider the ontological evolution of software from a perspective of complex social structure. It can be argued that any given software is a dynamic of data relations that combine in interesting ways to emerge increasingly complex strata of information interactions. And although these interactions are computationally discrete, they are never completely predictive, for algorithms operate as perturbations within the consensual domain of software's interactions with itself and other software. Of critical importance is the realization that each stratum must have an independent tendency for self-organization, adaptation and movement.

At higher magnitudes these sorts of social interactions combine as software drift. Software drift is the continuous structural change evidenced as software seeks to both sustain and re-define an appropriate ontogeny. It is an ontogeny that is simultaneously context and environment, application and human interface. Associative rules appear to guide software drift in the form of integrative or dissociative processes of feedback and constraint. And perhaps, just perhaps, the social fabric of software, the ontogeny we observe, is merely a combinatoric of these drifting strata of identity. Three conceptual frameworks need be addressed: Scaled States, Interiority/Exteriority and Cross-Domain Referencing.

States

"A medium is a medium is a medium." (Friedrich A. Kittler)

Software wants a social life. Software ontogeny is not fixed rather it scales in a fluid transition from machine code to interface as social entailments emerge from interacting sub-cultures of data relations. Scaling occurs across three parallel trajectories: technical, semantic and behavioral. A precise understanding of how interactions of these trajectories result in social emergence is elusive. In that data agencies of software are embedded, they obviously cannot be considered in isolation. Embedded data agencies are those processes that enclose a software in a kind of matrix of procedures and entailments.

In terms of description, a software is any coherency of embeddedness evident at any strata. Imposed conceptual frameworks for describing the status of each strata serves as an ontological histogram. Contemplation of software simply as framework for application is both inaccurate and inappropriate. As a descriptor, notions of application have little to do with the meaning of software and more to do with the environmental model in which software resides. Regardless, the scaling of data relations along parallel trajectories emerge correspondences that complexify into operating systems, networks, simulators and interfaces.

What is not clear is how the social behavior of data appears operative at all levels of embedded coherency. That is, social behavior is observable at all integrative and dissociative data relations and in and between trajectories at all scale states of a software. The obvious problem being that attempts of discrete description isolate embeddedness into arbitrary and hierarchical layers that determine an ontogeny from a particular context and from a particular point of view. The borders are simply not that clear. A single software can have many identities (a database can be an interface for example). Accepting that the social character of data is manifested in relational trajectories of the technical, semantic and behavioral, assumes that scaled states are continuous, smooth and non-differential. This analysis clearly offers a more engaging description that begins to account for the ontogenic complexity of software.

Internal and External

"Information is an expression of the difference between being inside and outside" (Tor Norretranders)

The observation of software as a composite of internal and external identities is obvious. But precisely how internality and externality, which are neither statically or structurally bound, make possible the system dynamics enabling ontogeny remains a mystery. Does ontogeny evolve as a composite of attributes, which exist as predicates that emerge as relations within a domain class of applications? Is ontogeny the membrane between Interiority and Exteriority? How does software

structurally couple with other typologies of software? As we have already seen, one possibility is that ontogeny arises as a composite of attributes in which the difference between being inside and outside is really a matter of the social interaction of data.

To speak of Interiority/Exteriority is to proclaim the autonomy of a unity. Indication of any being, object, thing or unity involves an act of distinction which separates the indicated from its background. The action of distinction brings forth the unity. Although humans distinguish software through actions of distinction, software is also brought forth through actions of distinction involving other software. In fact, in terms of autonomy, software actions of distinction are more complex, relying on the membronic substrate of Interiority/Exteriority.

One thing is clear, there can be no isolated software. Isolation is not autonomy. All dynamic systems of organization function as they function and are where they are at each instant, because of their internal/external trajectories. That we can refer to software as being autopoietic indicates a continuous structural coupling with other software. Thus the nature of software is to seek social relations at all scaled states internally and between all trajectories externally. We are forced to re-consider software in lieu of all the dimensions of structural plasticity such concepts represent. And how this plasticity accounts for the history of structural change that emerges particular trajectories in a particular software. We begin to realize that software is both in the environment and of the environment simultaneously. This is a revelation. Software is organism not tool and there is a radical difference in any attempt to explain behavior.

At a minimum software are most certainly social enterprises of inflection and prehension. These aphorisms ought to be like two guiding lights that permanently remind us that software takes place in language and that language is necessarily a social enterprise.

Cross-Domain Inferencing

> "Everything is distinguished by degree, everything differs by manner. These are the two principals of principals." (Gilles Deleuze)

Inferencing is a social action. There are two primary types. One is based on knowledge models and the other on analogy, or cross-domain inferencing. Knowledge models require premises and are at the center of expert systems research. However, inference also results directly from analogy that is significantly less tangible and difficult to shape as an architecture and has therefore not received the same attention. Theoretically speaking, analogy involves social correspondences of data and information structures across autonomous domains. Of the two types, cross-domain infererencing is of particular interest with regard to developing a better understanding of how the role membranes play in the social life of software.

Between software lies a relation of contradiction. We can postulate that the membranes between software represent a terrain of opposition that enables autonomy. It is an opposition of distinctions that shape the exclusionary borders of a particular software. Here in lies the contradiction. The very border enabling distinction also serves to enable software as a social unity. We can only speculate about the embedded idiosyncrasies that emerge as social behavior.

It would seem that software inferences software that inferences software. This is not so surprising; the question is how does it work?

For a software to be autonomous (for a software to be soft) it must realize the contradiction of autonomy. Consider that autonomy requires semantic indiscernability (a contradiction as relation). Indiscernability, of course, being intrinsic to the movement and inertia required to duplicate and reproduce. How? First, software must know the domain class to which it is structurally coupled. Second, a software must inference that it is a duplicate among duplicates. In other words a software must self-reference in order to be social. The role of the membrane is to enable both of these functions.

Every software is the other software and every membrane's function is to enable indiscernability.

Non-distinction by analogy is the product. Software embodies cross- domain inferencing as part of its semantic trajectory and therefore it's social architecture. Ontogeny is not the distinction of movement and inertia evident in a software, but rather the indiscernability of software social variability. A software wants to be what other software want it to be.Difference ceases to be intrinsic in order to become extrinsic. Furthermore, it may be that within the mechanics of these contradictions of relation lie the basis of how we can begin to conceive of a new generation of technology in which meta-data, distributed networks and hybridized data aggregation systems take on a life of their own. Literally.

Concepts of embedded systems and cross-domain inferencing are directly related to the processes of justification and learning involving organizational strategy. Inference theory has traditionally been used to conceptualize how organizations and their relationships as networks can lead to new knowledge. However, there may be great potential in a theory of inference that better describes how membranes enable autopoiesis in software.

Conclusion

Can software be softer? It seems apparent that dynamics of relationship between information structures will continue to evolve as the ecology of software becomes increasingly complex.

Computer to computer and human to computer communication, massively distributed networks, and hybridized meta-information objects are beginning to populate this new information ecology. Theoretical frameworks that incorporate a social-biological perspective may prove to be both interesting and imperative.

[Joel Slayton is founder and President of C5 Corporation. He is an artist, writer and theoretician whose work centers on information systems, knowledge representation and networks. He is Director of the CADRE Laboratory for New Media at San Jose State University and serves on the Boards of Leonardo/ISAST and GroundZero. Joel Slayton is Chair of the Leonardo/MIT Press Book Series and Executive Editor of SWITCH.]

http://www.c5corp.com
http://cadre.sjsu.edu
http://switch.sjsu.edu
http://mitpress.mit.edu/e-journals/Leonardo/isast/leobooks.html