MESHWORKS, HIERARCHIES AND INTERFACES

by Manuel De Landa

The world of interface design is today undergoing dramatic changes which in their impact promise to rival those brought about by the use of the point-and-click graphical interfaces popularized by the Macintosh in the early 1980's. The new concepts and metaphors which are aiming to replace the familiar desk-top metaphor all revolve around the notion of semi-autonomous, semi-intelligent software agents. To be sure, different researchers and commercial companies have divergent conceptions of what these agents should be capable of, and how they should interact with computer users. But whether one aims to give these software creatures the ability to learn about the users habits, as in the non-commercial research performed at MIT autonomous agents group, or to endow them with the ability to perform transactions in the users name, as in the commercial products pioneered by General Magic, the basic thrust seems to be in the direction of giving software programs more autonomy in their decision-making capabilities.

For a philosopher there are several interesting issues involved in this new interface paradigm. The first one has to do with the history of the software infrastructure that has made this proliferation of agents possible. From the point of view of the conceptual history of software, the creation of worlds populated by semi-autonomous virtual creatures, as well as the more familiar world of mice, windows and pull-down menus, have been made possible by certain advances in programming language design. Specifically, programming languages needed to be transformed from the rigid hierarchies which they were for many years, to the more flexible and decentralized structure which they gradually adopted as they became more "object-oriented". One useful way to picture this transformation is as a migration of control from a master program (which contains the general task to be performed) to the software modules which perform all the individual tasks. Indeed, to grasp just what is at stake in this dispersal of control, I find it useful to view this change as a part of a larger migration of control from the human body, to the hardware of the machine, then to the software, then to the data and finally to the world outside the machine. Since this is a crucial part of my argument let me develop it in some detail.

The first part of this migration, when control of machine-aided processes moved from the human body to the hardware, may be said to have taken place in the eighteenth century when a series of inventors and builders of automata created the elements which later came together in the famous Jacquard loom, a machine which automated some of the tasks involved in weaving patterns in textiles. Jacquards loom used a primitive form of software, in which holes punched into cards coded for some of the operations behind the creation of patterned designs. {1} This software, however, contained only data and not control structures. In other words, all that was coded in the punched cards was the patterns to be weaved and not any directions to alter the reading of the cards or the performance of the operations, such as the lifting of the warp threads. Therefore, it was the machine's hardware component that "read" the cards and translated the data into motion, in which control of the process resided. Textile workers at the time were fully aware that they had lost some control to Jacquards loom, and they manifested their outrage by destroying the machines in several occasions.

The idea of coding data into punched cards spread slowly during the 1800's, and by the beginning of our century it had found its way into computing machinery, first the tabulators used by Hollerith to process the 1890 United States census, then into other tabulators and calculators. In all these cases control remained embodied in the machine's hardware. One may go as far as saying that even the first modern computer, the imaginary computer created by Alan Turing in the 1930's still kept control in the hardware, the scanning head of the Turing machine. The tape that his machine scanned held nothing but data. But this abstract computer already had the seed of the next step, since as Turing himself understood, the actions of the scanning head could themselves be represented by a table of behavior, and the table itself could now be coded into the tape. Even though people may not have realized this at the time, coding both numbers and operations on numbers side by side on the tape, was the beginning of computer software as we know it. {2} When in the 1950's Turing created the notion of a subroutine, that is, the notion that the tasks that a computer must perform can be embodied into separate sub-programs

all controlled by a master program residing in the tape, the migration of control from hardware to software became fully realized. From then on, computer hardware became an abstract mesh of logical gates, its operations fully controlled by the software.

The next step in this migration took place when control of a given computational process moved from the software to the very data that the software operates on. For as long as computer languages such as FORTRAN or Pascal dominated the computer industry, control remained hierarchically embedded in the software. A master program would surrender control to a subroutine whenever that sub-task was needed to be performed, and the subroutine itself may pass control to an even more basic subroutine. But the moment the specific task was completed, control would move up the hierarchy until it reached the master program again. Although this arrangement remained satisfactory for many years, and indeed, many computer programs are still written that way, more flexible schemes were needed for some specific, and at the time, esoteric applications of computers, mostly in Artificial Intelligence.

Trying to build a robot using a hierarchy of subroutines meant that researchers had to completely foresee all the tasks that a robot would need to do and to centralize all decision-making into a master program. But this, of course, would strongly limit the responsiveness of the robot to events occurring in its surroundings, particularly if those events diverged from the predictions made by the programmers. One solution to this was to decentralize control. The basic tasks that a robot had to perform were still coded into programs, but unlike subroutines these programs were not commanded into action by a master program. Instead, these programs were given some autonomy and the ability to scan the data base on their own. Whenever the found a specific pattern in the data they would perform whatever task they were supposed to do. In a very real sense, it was now the data itself that controlled the process. And, more importantly, if the data base was connected to the outside world via sensors, so that patterns of data reflected patterns of events outside the robot, then the world itself was now controlling the computational process, and it was this that gave the robot a degree of responsiveness to its surroundings.

Thus, machines went from being hardware-driven, to being software-driven, then data-driven and finally event-driven. Your typical Macintosh computer is indeed an event-driven machine even if the class of real world events that it is responsive to is very limited, including only events happening to the mouse (such as position changes and clicking) as well as to other input devices. But regardless of the narrow class of events that personal computers are responsive to, it is in these events that much of the control of the processes now resides. Hence, behind the innovative use of windows, icons, menus and the other familiar elements of graphical interfaces, there is this deep conceptual shift in the location of control which is embodied in object-oriented languages. Even the new interface designs based on semi-autonomous agents were made possible by this decentralization of control. Indeed, simplifying a little, we may say that the new worlds of agents, whether those that inhabit computer screens or more generally, those that inhabit any kind of virtual environment (such as those used in Artificial Life), have been the result of pushing the trend away from software command hierarchies ever further.

The distinction between centralized and decentralized control of given process has come to occupy center-stage in many different contemporary philosophies. It will be useful to summarize some of this philosophical currents before I continue my description of agent-based interfaces, since this will reveal that the paradigm-shift is by no means confined to the area of software design. Economist and Artificial Intelligence guru Herbert Simon views bureaucracies and markets as the human institutions which best embody these two conceptions of control.{3} Hierarchical institutions are the easiest ones to analyze, since much of what happens within a bureaucracy in planned by someone of higher rank, and the hierarchy as a whole has goals and behaves in ways that are consistent with those goals. Markets, on the other hand, are tricky. Indeed, the term "market" needs to be used with care because it has been greatly abused over the last century by theorists on the left and the right. As Simon remarks, the term does not refer to the world of corporations, whether monopolies or oligopolies, since in these commercial institutions decision-making is highly centralized, and prices are set by command.

I would indeed limit the sense of the term even more to refer exclusively to those weakly gatherings of people at a predefined place in town, and not to a dispersed set of consumers catered by a system of middleman (as when one speaks of the "market" for personal computers). The reason is that, as historian Fernand Braudel has made it clear, it is only in markets in the first sense that we have any idea of what the dynamics of price formation are. In other words, it is only in peasant and small town markets that decentralized decision-making leads to prices setting themselves up in a way that we can understand. In any other type of market economists simply assume that supply and demand connect to each other in a functional way, but they do not give us any specific dynamics through which this connection is effected. {4} Moreover, unlike the idealized version of markets guided by an "invisible

hand" to achieve an optimal allocation of resources, real markets are not in any sense optimal. Indeed, like most decentralized, self-organized structures, they are only viable, and since they are not hierarchical they have no goals, and grow and develop mostly by drift. {5}

Herbert Simon's distinction between command hierarchies and markets may turn out to be a special case of a more general dichotomy. In the view of philosophers Gilles Deleuze and Felix Guattari, this more abstract classes, which they call strata and self-consistent aggregates (or trees and rhizomes), are defined not so much by the locus of control, as by the nature of elements that are connected together. Strata are composed of homogenous elements, whereas self-consistent aggregates articulate heterogeneous elements as such. {6} For example, a military hierarchy sorts people into internally homogenous ranks before joining them together through a chain of command. Markets, on the other hand, allow for a set of heterogeneous needs and offers to become articulated through the price mechanism, without reducing this diversity. In biology, species are an example of strata, particularly if selection pressures have operated unobstructedly for long periods of time allowing the homogenization of the species gene pool. On the other hand, ecosystems are examples of self-consistent aggregates, since they link together into complex food webs a wide variety of animals and plants, without reducing their heterogeneity. I have developed this theory in more detail elsewhere, but for our purposes here let's simply keep the idea that besides centralization and decentralization of control, what defines these two types of structure is the homogeneity or heterogeneity of its composing elements.

Before returning to our discussion of agent-based interfaces, there is one more point that needs to be stressed. As both Simon and Deleuze and Guattari emphasize, the dichotomy between bureaucracies and markets, or to use the terms that I prefer, between hierarchies and meshworks, should be understood in purely relative terms. In the first place, in reality it is hard to find pure cases of these two structures: even the most goal-oriented organization will still show some drift in its growth and development, and most markets even in small towns contain some hierarchical elements, even if it is just the local wholesaler which manipulates prices by dumping (or withdrawing) large amounts of a product on (or from) the market. Moreover, hierarchies give rise to meshworks and meshworks to hierarchies. Thus, when several bureaucracies coexist (governmental, academic, ecclesiastic), and in the absence of a super-hierarchy to coordinate their interactions, the whole set of institutions will tend to form a meshwork of hierarchies, articulated mostly through local and temporary links. Similarly, as local markets grow in size, as in those gigantic fairs which have taken place periodically since the Middle Ages, they give rise to commercial hierarchies, with a money market on top, a luxury goods market underneath and, after several layers, a grain market at the bottom. A real society, then, is made of complex and changing mixtures of these two types of structure, and only in a few cases it will be easy to decide to what type a given institution belongs.

A similar point may be made about the worlds inhabited by software agents. The Internet, to take the clearest example first, is a meshwork which grew mostly by drift. No one planned either the extent or the direction of its development, and indeed, no one is in charge of it even today. The Internet, or rather its predecessor, the Arpanet, acquired its decentralized structure because of the needs of U.S. military hierarchies for a command and communications infrastructure which would be capable of surviving a nuclear attack. As analysts from the Rand Corporation made it clear, only if the routing of the messages was performed without the need for a central computer could bottlenecks and delays be avoided, and more importantly, could the meshwork put itself back together once a portion of it had been nuclearly vaporized. But in the Internet only the decision-making behind routing is of the meshwork type. Decision-making regarding its two main resources, computer (or CPU) time and memory, is still hierarchical.

Schemes to decentralize this aspect do exist, as in Drexler's Agoric Systems, where the messages which flow through the meshwork have become autonomous agents capable of trading among themselves both memory and CPU time. {7} The creation by General Magic of its Teletext operating system, and of agents able to perform transactions on behalf of users, is one of the first real-life steps in the direction of a true decentralization of resources. But in the meanwhile, the Internet will remain a hybrid of meshwork and hierarchy components, and the imminent entry of big corporations into the network business may in fact increase the amount of command components in its mix.

These ideas are today being hotly debated in the field of interface design. The general consensus is that interfaces must become more intelligent to be able to guide users in the tapping of computer resources, both the informational wealth of the Internet, as well as the resources of ever more elaborate software applications. But if the debaters agree that interfaces must become smarter, and even that this intelligence will be embodied in agents, they disagree on how the agents should acquire their new capabilities. The debate pits two different traditions of Artificial Intelligence against each other: Symbolic

AI, in which hierarchical components predominate, against Behavioral AI, where the meshwork elements are dominant. Basically, while in the former discipline one attempts to endow machines with intelligence by depositing a homogenous set of rules and symbols into a robot's brain, in the latter one attempts to get intelligent behavior to emerge from the interactions of a few simple task-specific modules in the robot's head, and the heterogeneous affordances of its environment. Thus, to build a robot that walks around a room, the first approach would give the robot a map of the room, together with the ability to reason about possible walking scenarios in that model of the room. The second approach, on the other hand, endows the robot with a much simpler set of abilities, embodied in modules that perform simple tasks such as collision-avoidance, and walking-around-the-room behavior emerges from the interactions of these modules and the obstacles and openings that the real room affords the robot as it moves.{8}

Translated to the case of interface agents, for instance, personal assistants in charge of aiding the user to understand the complexities of particular software applications, Symbolic AI would attempt to create a model of the application as well as a model of the working environment, including a model of an idealized user, and make these models available in the form of rules or other symbols to the agent. Behavioral AI, on the other hand, gives the agent only the ability to detect patterns of behavior in the actual user, and to interact with the user in different ways so as to learn not only from his or her actual behavior but also from feedback that the user gives it. For example, the agent in question would be constantly looking over the user's shoulder keeping track on whatever regular or repetitive patterns it observes. It then attempts to establish statistical correlations between certain pairs of actions that tend to occur together. At some point the agent suggests to the user the possibility of automating these actions, that is, that whenever the first occurs, the second should be automatically performed. Whether the user accepts or refuses, this gives feedback to the agent. The agent may also solicit feedback directly, and the user may also teach the agent by giving some hypothetical examples. {9}

In terms of the location of control, there is very little difference between the agents that would result, and in this sense, both approaches are equally decentralized. The rules that Symbolic AI would put in the agents head, most likely derived from interviews of users and programmers by a Knowledge Engineer, are independent software objects. Indeed, in one of the most widely used programming languages in this kind of approach (called a "production system") the individual rules have even more of a meshwork structure that many object-oriented systems, which still cling to a hierarchy of objects. But in terms of the overall human-machine system, the approach of Symbolic AI is much more hierarchical. In particular, by assuming the existence of an ideal user, with homogenous and unchanging habits, and of a workplace where all users are similar, agents created by this approach are not only less adaptive and more commanding, they themselves promote homogeneity in their environment. The second class of agents, on the other hand, are not only sensitive to heterogeneities, since they adapt to individual users and change as the habits of this users change, they promote heterogeneity in the work place by not subordinating every user to the demands of an idealized model.

One drawback of the approach of Behavioral AI is that, given that the agent has very little knowledge at the beginning of a relationship with a user, it will be of little assistance for a while until it learns about his or her habits. Also, since the agent can only learn about situations that have recurred in the past, it will be of little help when the user encounters new problems. One possible solution, is to increase the amount of meshwork in the mix and allow agents from different users to interact with each other in a decentralized way. {10} Thus, when a new agent begins a relation with a user, it can consult with other agents and speed up the learning process, assuming that is, that what other agents have learned is applicable to the new user. This, of course, will depend on the existence of some homogeneity of habits, but at least it does not assume a complete homogenous situation from the outset, an assumption which in turn promotes further uniformization. Besides, endowing agents with a static model of the users makes them unable to cope with novel situations. This is also a problem in the Behavioral AI approach but here agents may aid one another in coping with novelty. Knowledge gained in one part of the workplace can be shared with the rest, and new knowledge may be generated out of the interactions among agents. In effect, a dynamic model of the workplace would be constantly generated and improved by the collective of agents in a decentralized way, instead of each one being a replica of each other operating on the basis of a static model centrally created by a knowledge engineer.

I would like to conclude this brief analysis of the issues raised by agent-based interfaces with some general remarks. First of all, from the previous comments it should be clear that the degree of hierarchical and homogenizing components in a given interface is a question which affects more than just events taking place in the computer's screen. In particular, the very structure of the workplace, and the relative status of humans and machines is what is at stake here. Western societies have undergone at least two centuries of homogenization, of which the most visible element is the assembly-line and related mass-production techniques, in which the overall thrust was to let machines discipline and

control humans. In this circumstances, the arrival of the personal computer was a welcome antidote to the development of increasingly more centralized computer machinery, such as systems of Numerical Control in factories. But this is hardly a victory. After two hundred years of constant homogenization, working skills have been homogenized via routinization and Taylorization, building materials have been given constant properties, the gene pools of our domestic species homogenized through cloning, and our languages made uniform through standardization.

To make things worse, the solution to this is not simply to begin adding meshwork components to the mix. Indeed, one must resist the temptation to make hierarchies into villains and meshworks into heroes, not only because, as I said, they are constantly turning into one another, but because in real life we find only mixtures and hybrids, and the properties of these cannot be established through theory alone but demand concrete experimentation. Certain standardizations, say, of electric outlet designs or of data-structures traveling through the Internet, may actually turn out to promote heterogenization at another level, in terms of the appliances that may be designed around the standard outlet, or of the services that a common data-structure may make possible. On the other hand, the mere presence of increased heterogeneity is no guarantee that a better state for society has been achieved. After all, the territory occupied by former Yugoslavia is more heterogeneous now than it was ten years ago, but the lack of uniformity at one level simply hides an increase of homogeneity at the level of the warring ethnic communities. But even if we managed to promote not only heterogeneity, but diversity articulated into a meshwork, that still would not be a perfect solution. After all, meshworks grow by drift and they may drift to places where we do not want to go. The goal-directedness of hierarchies is the kind of property that we may desire to keep at least for certain institutions. Hence, demonizing centralization and glorifying decentralization as the solution to all our problems would be wrong. An open and experimental attitude towards the question of different hybrids and mixtures is what the complexity of reality itself seems to call for. To paraphrase Deleuze and Guattari, never believe that a meshwork will suffice to save us. {11}

Footnotes:

{1} Abbot Payson Usher. The Textile Industry, 1750-1830. In Technology in Western Civilization. Vol. 1. Melvin Kranzberg and Carrol W. Pursell eds. (Oxford University Press, New York 1967). p. 243

{2} Andrew Hodges. Alan Turing: The Enigma. (Simon & Schuster, New York 1983). Ch. 2

{3} Herbert Simon. The Sciences of the Artificial. (MIT Press, 1994). p.43

{4} Fernand Braudel. The Wheels of Commerce. (Harper and Row, New York, 1986). Ch. I

{5} Humberto R. Maturana and Francisco J. Varela. The Tree of Knowledge. The Biological Roots of Human Understanding. (Shambhala, Boston 1992). p. 47 and 115.

{6} Gilles Deleuze and Felix Guattari. A Thousand Plateaus. (University of Minnesota Press, Minneapolis, 1987). p. 335

{7} M.S. Miller and K.E. Drexler. Markets and Computation: Agoric Open Systems. In The Ecology of Computation. Bernardo Huberman ed. (North-Holland, Amsterdam 1988).

{8} Pattie Maes. Behaviour-Based Artificial Intelligence. In From Animals to Animats. Vol. 2. Jean-Arcady Meyer, Herbert L. Roitblat and Stewart W. Wilson. (MIT Press, Cambridge Mass, 1993). p. 3

{9} Pattie Maes and Robyn Kozierok. Learning Interface Agents. In Proceedings of AAAI È93 Conference. (AAAI Press, Seattle WA. 1993). p. 459-465

{10} Yezdi Lashari, Max Metral and Pattie Maes. Collaborative Interface Agents. In Proceedings of 12th National Conference on AI. (AAAI Press, Seattle WA. 1994). p. 444-449

{11} Deleuze and Guattari. op. cit. p. 500. (Their remark is framed in terms of "smooth spaces" but it may be argued that this is just another term for meshworks).


e~
SCAPE