The worm that turned:
A social use of computer viruses

Ian H. Witten
Computer Science
Calgary University
Calgary
T2N 1N4
Canada

Harold Thimbleby
Computing Science
Stirling University
Stirling
FK9 4LA
Scotland

July 31, 1989

Computer viruses have become the bane of personal computers. But can similar mechanisms be used to spread new information and update old information for the benefit of users?

A virus is a piece of computer program that attaches itself to other programs, incorporating itself into them so that as well as performing their intended function they surreptitiously do other things. Programs so corrupted seek others to which to attach the virus, and so the "infection" spreads.

Although first developed on multi-user computer systems with shared disc facilities, viruses can spread in a personal computer environment where users share floppy discs or other removable storage media. Inserting an infected disc into the system invisibly infects the system itself, and other, "clean," floppy discs inserted later on become contaminated too. The whole operation is performed invisibly—and indeed users only discover the signs much later, long after damage is done and countless floppy discs have become riddled with the pest. In practice it is very hard to guard against infection, for people quite naturally want to share with others their information and the programs they write.

Viruses spread rapidly, and are an antisocial menace. But they do point out an effective way of spreading information between personal computer users, without requiring any special communication equipment or update procedures. The medium is floppy disc; the mechanism is social. A benign virus—for which we have invented the term "Liveware" to emphasize its more positive connotations—can exploit the same mechanism to spread useful information. Liveware silently updates itself whenever a floppy disc is inserted into the computer. It relies on cooperative computers to act as carriers of information. Unlike a real virus, it does not act without the consent of the computer user; it does not spread from program to program; it allows itself to be deleted without trace. In short, it remains under control.

Liveware allows a group of users to share widely distributed information almost as effectively as if they had a common database. There are many structures that support shared information services (hypertexts, filestores, bulletin boards), with many important applications ranging from airline reservation systems through city tourist guides to hobbyist newsletters and networked electronic noticeboards. In contrast to Liveware, these information sources are centralized, transaction-oriented, proprietary—and therefore expensive! Their information resides on a single machine or installation that requires a large support infrastructure. There are carefully prescribed procedures for accessing and updating the information. Someone has a financial stake in the service, and will naturally want to exploit or protect their investment.

The value of the information itself should not be confused with the value of the system that supports it. One of the growing paradoxes of computing technology is that many individuals now operate personal computers whose power rivals that of large installations, yet without the infrastructure of support that traditionally accompanies such systems. Such individuals have the technical resources to make excellent use of shared information—if only there was a good enough way to maintain it. Liveware provides precisely such a mechanism.

In contrast to mainframe machines with controlled clients, Liveware just assumes small computers with the ability to read and write on low-cost portable media such as floppy discs. Information is communicated by users passing discs amongst each other. Although they will be aware that they are acting as carriers of information, they need take no explicit action to ensure that transmission takes place—apart from inserting floppy discs into disc drives while using the system. We assume that the network is connected socially in a rich enough way to provide as much information flow as is required to maintain appropriate currency of each person's database. A testament to the power of social networks is provided by the success and rapidity which has been observed for the transmission of computer viruses, and indeed the extreme difficulty of *avoiding* infection!

While it is hard to make many guarantees about the flow of information, the fact that information distribution in Liveware is completely under user control bestows a number of advantages. The mechanism—social connectivity—underlies much of our non-computer information gathering activity anyway. After becoming enmeshed in a social group, one can expect as a matter of course to be in regular first- or second-hand contact with the most useful sources of information. Liveware takes advantage of the fact that individuals often travel to their colleagues from time to time, and can easily carry discs with them. Users are motivated since the Liveware method means that as they share their own information, *they* will automatically pick up new information contributed by other people. The low overhead for both transmission and reception of information makes it easy to get updates from appropriate sources, either as needed or on a regular basis. Users can gauge the currency of their information and, if necessary, personally request an update directly from the information's owner. Finally, social mechanisms and conventions (legal, technical, etc.) can be imposed on the information flow where necessary to ensure guaranteed currency.

The mechanism of Liveware is an automatic instrument that propagates new information and handles technicalities such as version control and integrity. We have embedded it in a hypertext system that permits users to browse through screenfuls or "cards" of information, linked together in arbitrary ways. The card is the basic unit, and for Liveware to operate correctly each one requires additional information which is hidden during normal use. This information enables Liveware to work automatically and is normally of no interest to the user. It includes the *signature* of the card's owner, an *identification code* which distinguishes the card from others belonging to that person, and a *time stamp* which records when the card was last modified (or originally created). Normally the signature is the owner's actual name together with a secret password supplied by him, encrypted so that others cannot read it back.

It is necessary that some cards be nominated as "controller" of the hypertext. As well as containing the program that supports the Liveware mechanism itself, these perform some other functions. For example, they declare the name and purpose of the database and may identify some person who is responsible for its entire operation, and to whom users may direct enquiries. That person may be empowered to introduce new owners into the Liveware, and to reset passwords for owners who have forgotten them. He or she may also be able to remove information and eliminate owners. When a database is first designed, the facilities that its controller is to provide are specified too.

A user is "enlivened" with a particular Liveware database simply by giving him a copy of it. From then on, his database will be updated automatically whenever the opportunity arises. However, there is no permanent effect: to rid himself of it he simply deletes all copies in the normal way and no trace will remain. (This is quite unlike a virus.)

The essence of Liveware is that information is merged from separate versions of the database whenever possible, updating cards that are obsolete and introducing new ones. Whenever a Liveware database is entered, the system searches the computer's disc drives for other versions of the same database. If any are found, a merge is performed. This operation examines each card in turn, updating it if a more recent card with the same signature and identification code appears in the other version. Cards which appear in one version but not the other are copied. After merging, both versions of the database are identical. The activity occurs quite automatically and invisibly, without the user's intervention.

Users may make slips and accidentally corrupt the database. Many accidents delete information, which Liveware is well set up to restore through the normal merging procedure. Indeed. a sufficiently large community of Liveware users can protect each other from disasters like disc crashes. Thus if our disc dies on us, destroying our Liveware, we have only to take a blank disc to a friend whom we recently visited to get an up-to-date version. Other accidents introduce new—but trivial!—information, through mistakenly typing over someone else's entry, for example. Since Liveware is intended to propagate new information, it is certainly very important to protect against accidental changes, particularly since they may be more recent than—and hence overwrite—genuine updates made elsewhere.

So, to make things safe, when a user wants to add or update information, he must "log in" to the Liveware by supplying a secret password. This identifies him as the owner of some or all of the cards. Once logged in, the user may change information he owns, and the Liveware mechanism takes note of any changes. It is not possible to implement a really secure log-in mechanism on a personal computer without hardware support. Determined hackers could overcome any security system that might be devised. In the loosely distributed system that supports Liveware, they can have all the uninterrupted time they need, and work completely unobserved. Anyway, as a last resort, they could re-implement an exact look-alike system (a so-called *Trojan Horse*) without a great deal of difficulty, and it is quite impossible to prevent this. Under these circumstances, Liveware can do no more than provide the *appearance* of security. That is sufficient to protect databases against accidents, and perhaps against the idly curious. Given human nature, or rather certain inescapable manifestations of it, it is an unfortunate fact that for large-scale applications, hardware implementations of security will be necessary.

Perhaps the most unusual aspect of Liveware is the intertwining of social and technical issues that it stimulates. It effectively requires the imposition of a certain social etiquette. Specifically:

> Owners are responsible for their own cards.
> Owners are not to change other owners' cards.
> Owners are known publicly by their names.

This is normal information ownership: what's ours is ours, what's yours is yours, and we all know who's who. These rules apply even when information is not personal: they ensure that precisely one person is responsible for each card and that there is no possibility of a card being updated at different times unless the last update is required by the actual owner. The final rule is required because information owners may join the same Liveware in different places and times: there must be a naming convention to avoid any later conflict with ownership.

Liveware cannot work correctly if this convention is flaunted. It is assumed that when two cards with identical identification meet (during merging), then the one with the most recent time stamp can supersede the other. This assumption is not correct if gratuitous updates can be made at a later time than real updates, for instance if we correct a spelling mistake in a card of yours, but at a time after you most recently updated that card. When these two cards meet, our minor correction would replace whatever corrections you had made.

It is worth emphasizing that the rules of information ownership may, in practice, be irritatingly restrictive. For example, it is not permissible to personalise cards by introducing notes, doodling, reformatting, correcting spelling, or whatever. One's own version of the database is *public* information, for it may be transmitted directly to others, and not just a private copy. And the consequences of altering someone else's card are not simply that the interference will be transmitted to others, but—much worse—that *his* updates elsewhere may be superseded and lost. The reason for this situation is that the time stamp is the maximum update time for *all* the information on one card; it therefore cannot distinguish one change from another on the same card.

There are various options for handling the registration of new information contributors. The simplest is to allow anyone to register himself as a new owner, and choose a password at the same time. This gives free access to all, but runs the risk of pollution of the database by unwanted information. At the other extreme is a centralised policy where new contributors must be registered by a person responsible for the Liveware database, who also gives them initial passwords. Numerous more elaborate schemes are possible: existing owners may be empowered to introduce new ones, or several may have to collaborate to propose a new one. In this case the Liveware may enforce collaboration in a single interactive session, or permit nominations to be stored on cards owned by the proposers so that the process may be distributed in time and place. If the process is distributed, the nominee could take his Liveware disc round potential proposers until he has collected the requisite number of nominations. Moreover, each owner's heritage may be stored to allow limits to be placed on the number of introductions that owners may participate in. All of these possibilities are quite simple to implement; the chief problem is in deciding which scheme is suitable for any particular purpose. It is an intriguing thought that Liveware highlights live issues of democratic process—and also provides a forum for their precise formulation and assessment.

An information owner may wish to delete part of a Liveware database. This is difficult, since the Liveware may already be widely distributed. There are several possible mechanisms. The first, which has the advantage of being extremely simple, allows a user to withdraw information *so far as he is concerned*, but does not address the persistence of the information elsewhere. A card can be declared "dormant," and dormancy propagates in the normal way, a record of the card being retained to prevent it being reinstated by future merging from other versions of the database. The second mechanism is to make information "self destruct" when it passes its expiry date, ensuring that cards disappear autonomously no matter how widely the database is distributed. However, here the need for removal must be anticipated when the information is created. The third mechanism is to chase a card that has been distributed by an "anti-card" that destroys it on meeting. This requires Liveware to record the recipients of information in order to distribute anti-cards correctly.

Overall, then, Liveware is an effective mechanism to support the distribution of computer information by social means. It has surprisingly wide applicability—especially considering the simplicity of the idea—whenever people need to share information using informal distribution channels. Although originally conceived for use in situations where information changes slowly and users are not overly concerned with getting the most up-to-date versions, it can be used more generally because users can gauge the currency of their information and impose additional social conventions on the information flow. An important special case is when there is only one user: Liveware is ideal for backing up personal files and for sharing one's information between several workplaces. Another application is to support news or mail networks. If you know somebody who knows somebody ... who is witnessing news (and every intermediary has computers or can pass on discs), then you have an opportunity to keep up to date with that news. Just how "up to date" depends on social factors such as connectivity of the network and willingness of people to allow their computers to act as carriers. Liveware can manage news arriving in pieces via different routes, possibly out of order, possibly with losses.

Although we have used the metaphor of viruses to characterise the autonomous, invisible, merge operation that is at the heart of Liveware, the scheme, being benign, is nowhere near as virulent as it might be. A computer virus attaches copies of itself to other programs indiscriminately, seeking to spread itself as widely as possible regardless of the users' wishes. Liveware respects the right of the user to his computer and does not undermine his authority over it.

Ian Witten is with the Department of Computer Science, University of Calgary, Canada, and during the Summer of 1989 was supported by the Science and Engineering Research Council as a Visiting Fellow at Stirling. Harold Thimbleby is a Professor of Information Technology at Stirling University, Scotland.

Viruses and other nasty programs

The term "virus" is a popular catch-all for many kinds of malicious software. A "logic bomb" or "time bomb" is a destructive program activated by a certain combination of circumstances, or an a certain date. A "Trojan horse" is any bug inserted into a computer program that takes advantage of the trusted status of its host by surreptitiously performing unintended functions. A "worm" is a robust kind of distributed program that invades workstations on a network: it consists of several processes or "segments" that keep in touch through the network; when one is lost (for example, by a workstation being rebooted), the others conspire to replace it on another processor—they search for an idle workstation, load it with a copy of themselves, and start it up.

Viruses attach copies of themselves to other programs. They work by altering disc files that contain the compiled version of otherwise harmless programs. When an infected program is invoked, it seeks other programs stored in files which it can change, and infects them by modifying the files to include a copy of the virus code and inserting an instruction to branch to that code at the old program's starting point. After doing its work, the virus quickly starts up the original program so that the user is unaware of its intervention.

The idea of a maliciously self-propagating computer program originated in Gerrold's 1972 novel *When Harlie Was One*, in which a program called telephone numbers at random until it found other computers into which it could spread. Worms were also presaged in science fiction, by Brunner's 1975 novel *The Shockwave Rider*. The first actual virus program seems to have been created in 1983, as the result of a discussion in a computer security seminar, and described at the AFIPS Computer Security Conference the following year. In 1984 Thompson in his Turing Award Lecture showed how a self-replicating bug can infest a compiler or other language processor.

Virus attacks were not reported until a few years thereafter, and so far have been more in the nature of electronic vandalism than serious subversion. One of the first occurred in late 1987, when over a two-month period a virus quietly insinuated itself into IBM PC programs in a Jerusalem university. It was noticed because it caused programs to grow longer (due to a bug, it repeatedly re-infected files). Once discovered, it was analyzed and an antidote devised. It was designed to slow processors down on certain Fridays, and to erase all files on Friday 13 May.

At about the same time another PC virus invaded LeHigh University, while a much-publicized "chain letter" Christmas message spread itself by self-replication, clogging the BITNET network—it was eradicated by a massive network shutdown. Early 1988 saw a relatively harmless Macintosh virus designed to distribute a "message of peace," and a number of other viruses for personal computers. By that time talk about viruses had invaded the news media.

Late in 1988 a worm program was inserted into the American Internet computer network. It exploited several security flaws in SUN and VAX systems running Unix to spread itself from system to system. Although discovered within a few hours, it required a huge effort (estimated at 50,000 man-hours) by programmers at affected sites to counteract and eliminate the worm over a period of weeks. Again, it was unmasked by a bug (a bug in a bug!): under some circumstances it replicated itself so fast that it seriously slowed down the infected host.

Hypertext

"If Emma takes the parcel home, go to page 34; if Emma decides to open it here and now, go to page 14" ... a sentence from a 'branching story' of the sort that is becoming very popular in childrens' detective stories, and was once very popular for 'programmed learning texts.'

The same idea works nicely on a computer, but instead of paper pages, we have screens or cards (like 3 x 5 card-index cards) and instead of the reader turning to page so-and-so, there are 'buttons' that can be pressed, and the computer 'turns the page.' This is the basic idea behind hypertext: bits of text and pictures linked together in routes that depend entirely on the way the reader takes the story.

Stories and programmed learning are far from the only ways of using hypertext. First, since each 'page' of the hypertext is on the computer it can do *anything* a computer can do. Secondly, pages or cards can contain text, pictures and even sound effects as well. In a programmed learning text 'buttons' are always the same bit of '**`If you think the answer is 5 turn to page 1234`**,' but in Hypertext the button can be a picture, perhaps a real button like ● or anything else. The pictures of Liveware in the article give other examples of buttons; one is a magnifying glass, suggesting (we hope) 'press this and take a closer look,' which is what it does.

Two examples:

- The BBC in conjunction with Apple Computer have produced a hypertext document designed for teaching about nature conservancy. The *BBC Ecodisc* is a large simulation of a nature reserve that allows students to get a feel for what running a nature reserve involves. They can take walks around the reserve, as they press buttons on a map of the reserve they are taken from place to place. On a card talking about the wood, for instance, the student can press a button to have a closer look at the flowers, and he will see a photograph of the flower in summer, or whatever time of year it is. Or they can stop to listen to the sounds, perhaps a particular sort of bird calling. At any point, the user can consult experts, that is every card has a button that brings up an expert's prerecorded film—in sound and colour—discussing some interesting feature of the habitat.
- *Glasgow Online* is a different sort of hypertext to help tourists to Glasgow find tourist attractions. Some pages are maps of Glasgow, containing lots of buttons covering each site of interest, like cinemas or museums: other pages are indexes, so that a tourist could find information about, say, accommodation, shops, places of worship, or leisure activities. On pressing the 'accommodation' button, the user would be taken to a further page asking him to choose between hotels, guest houses, self-catering, hostels or camping facilities. On pressing one of those buttons, he gets, for instance, hotels listed by name, and then he can find out more about each hotel, for instance where it is on the map.

Liveware terminology

To underline the more positive social connotations of Liveware as opposed to viruses, it is necessary to use new terms for concepts otherwise known as infection and perpetration. Here are some terms used in the article.

- *Liveware* — A hypertext (or other) database that updates itself autonomously whenever the occasion arises
- *Enliven* — To inoculate a person's computer with a Liveware database
- *Information owner* — an owner of one or more cards in the database, and the only person permitted to change them
- *Database owner* — the person responsible for the Liveware database as a whole. He is *not* empowered to alter information belonging to others
- *Signature* — a code identifying an owner, including his full name and perhaps an encrypted secret password that only he can generate
- *Livestamp* — the Liveware information recorded on each card: signature, identification code, and time stamp
- *Merge* — the joining of two Liveware databases together so that both contain the most recent information.

An example of Liveware

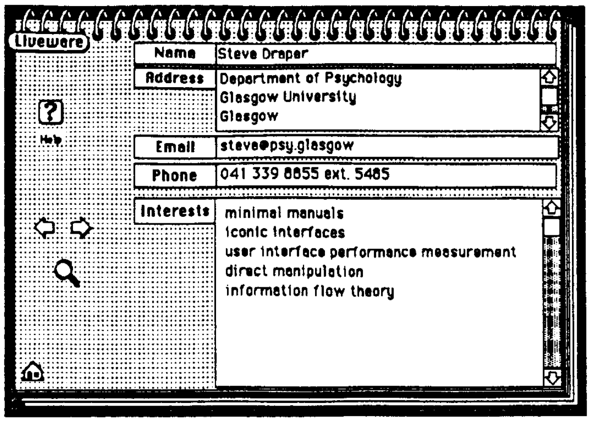
A pilot Liveware hypertext database has been implemented which stores information on research workers in Scotland who are interested in human-computer interaction. Setting it up in the conventional way would have required soliciting information from everyone, encouraging them to respond, collating data, and presenting it. It seemed that a far more effective way would be to distribute as much of the information processing as possible. Instead of a paper questionnaire, an outline database could be distributed on disc. Interested people would fill in their details—doing their own data entry—perhaps copying their part of the database on to further colleagues. Completed databases (that is, locally completed) would be returned for central, automated collation. We supposed that respondents would be further encouraged, since if they filled in and returned their database, they would receive a complete collated database in return. It was a short step from this practical application of distributed databases to also distribute the collation process itself, and a virus-like mechanism was suggested.

The purpose of the database is to supply information on each person registered in it. It provides a card for each person that records his or her name, address, phone number and email address, and a list of one-line phrases describing research interests. A complete list of people's interests is automatically collected together on a separate card. Clicking on a line of this index takes you to the card of the first person who has declared that interest; repeated clicking cycles through all relevant people's cards.

The front card includes a list of people represented (collected automatically on entry to the stack by examining all cards in it); clicking on a line will go to that person's card. The 'Add a new person' button allows new owners to register; they enter a dialogue that solicits their name and initial password, and a card is created which they can fill in. Also included on the first card is a list of versions of this stack. Other cards include information and help about the Liveware system itself, and a summary of the social conventions discussed earlier.

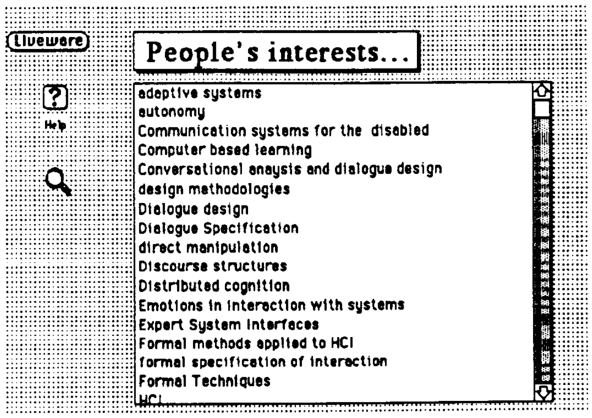
The database is implemented in the Apple HyperCard system, and the merge operation that is at the centre of the Liveware idea is written entirely in HyperTalk, a programming language provided with HyperCard. As the system locates new versions of the stack on disc, or as the user adds new versions manually to the list, they are used to update the current database and, conversely, it is used to update them.

Each card has an owner name and password, and the 'Liveware' button at the top left of each card gives controlled access to this hidden information, showing the owner and allowing one to log in or change the password. Passwords are used to impose a degree of integrity on the information. They are encrypted on entry by an encryption facility that is built in to HyperCard, and stored (invisibly) on each card in encrypted form. The same password is stored on all cards that belong to a given owner, so that cards are self-contained and can be treated independently during the merge operation.



Name	Steve Draper
Address	Department of Psychology Glasgow University Glasgow
Email	steve@psy.glasgow
Phone	041 339 8855 ext. 5485
Interests	minimal manuals iconic interfaces user interface performance measurement direct manipulation information flow theory

Figure 1: Information on a person in the Scottish HCI Database. The main part of the database consists of many cards of this form. Note the 'Liveware' button at the top left.



People's interests...

adaptive systems
autonomy
Communication systems for the disabled
Computer based learning
Conversational enaysis and dialogue design
design methodologies
Dialogue design
Dialogue Specification
direct manipulation
Discourse structures
Distributed cognition
Emotions in interaction with systems
Expert System Interfaces
Formal methods applied to HCI
formal specification of interaction
Formal Techniques
HCI

Figure 2: The list of research interests in the Scottish HCI Database. This collects all research interests mentioned in the database. Clicking on one takes you to the card of a person with that interest.

The worm that turned: A social use of computer viruses

Ian H. Witten
Computer Science
Calgary University
Calgary
T2N 1N4
Canada

Harold Thimbleby
Computing Science
Stirling University
Stirling
FK9 4LA
Scotland

July 31, 1989

Computer viruses have become the bane of personal computers. But can similar mechanisms be used to spread new information and update old information for the benefit of users?

A virus is a piece of computer program that attaches itself to other programs, incorporating itself into them so that as well as performing their intended function they surreptitiously do other things. Programs so corrupted seek others to which to attach the virus, and so the "infection" spreads.

Although first developed on multi-user computer systems with shared disc facilities, viruses can spread in a personal computer environment where users share floppy discs or other removable storage media. Inserting an infected disc into the system invisibly infects the system itself, and other, "clean," floppy discs inserted later on become contaminated too. The whole operation is performed invisibly—and indeed users only discover the signs much later, long after damage is done and countless floppy discs have become riddled with the pest. In practice it is very hard to guard against infection, for people quite naturally want to share with others their information and the programs they write.

Viruses spread rapidly, and are an antisocial menace. But they do point out an effective way of spreading information between personal computer users, without requiring any special communication equipment or update procedures. The medium is floppy disc; the mechanism is social. A benign virus—for which we have invented the term "Liveware" to emphasize its more positive connotations—can exploit the same mechanism to spread useful information. Liveware silently updates itself whenever a floppy disc is inserted into the computer. It relies on cooperative computers to act as carriers of information. Unlike a real virus, it does not act without the consent of the computer user; it does not spread from program to program; it allows itself to be deleted without trace. In short, it remains under control.

Liveware allows a group of users to share widely distributed information almost as effectively as if they had a common database. There are many structures that support shared information services (hypertexts, filestores, bulletin boards), with many important applications ranging from airline reservation systems through city tourist guides to hobbyist newsletters and networked electronic noticeboards. In contrast to Liveware, these information sources are centralized, transaction-oriented, proprietary—and therefore expensive! Their information resides on a single machine or installation that requires a large support infrastructure. There are carefully prescribed procedures for accessing and updating the information. Someone has a financial stake in the service, and will naturally want to exploit or protect their investment.

The value of the information itself should not be confused with the value of the system that supports it. One of the growing paradoxes of computing technology is that many individuals now operate personal computers whose power rivals that of large installations, yet without the infrastructure of support that traditionally accompanies such systems. Such individuals have the technical resources to make excellent use of shared information—if only there was a good enough way to maintain it. Liveware provides precisely such a mechanism.

In contrast to mainframe machines with controlled clients, Liveware just assumes small computers with the ability to read and write on low-cost portable media such as floppy discs. Information is communicated by users passing discs amongst each other. Although they will be aware that they are acting as carriers of information, they need take no explicit action to ensure that transmission takes place—apart from inserting floppy discs into disc drives while using the system. We assume that the network is connected socially in a rich enough way to provide as much information flow as is required to maintain appropriate currency of each person's database. A testament to the power of social networks is provided by the success and rapidity which has been observed for the transmission of computer viruses, and indeed the extreme difficulty of *avoiding* infection!

While it is hard to make many guarantees about the flow of information, the fact that information distribution in Liveware is completely under user control bestows a number of advantages. The mechanism—social connectivity—underlies much of our non-computer information gathering activity anyway. After becoming enmeshed in a social group, one can expect as a matter of course to be in regular first- or second-hand contact with the most useful sources of information. Liveware takes advantage of the fact that individuals often travel to their colleagues from time to time, and can easily carry discs with them. Users are motivated since the Liveware method means that as they share their own information, *they* will automatically pick up new information contributed by other people. The low overhead for both transmission and reception of information makes it easy to get updates from appropriate sources, either as needed or on a regular basis. Users can gauge the currency of their information and, if necessary, personally request an update directly from the information's owner. Finally, social mechanisms and conventions (legal, technical, etc.) can be imposed on the information flow where necessary to ensure guaranteed currency.

The mechanism of Liveware is an automatic instrument that propagates new information and handles technicalities such as version control and integrity. We have embedded it in a hypertext system that permits users to browse through screenfuls or "cards" of information, linked together in arbitrary ways. The card is the basic unit, and for Liveware to operate correctly each one requires additional information which is hidden during normal use. This information enables Liveware to work automatically and is normally of no interest to the user. It includes the *signature* of the card's owner, an *identification code* which distinguishes the card from others belonging to that person, and a *time stamp* which records when the card was last modified (or originally created). Normally the signature is the owner's actual name together with a secret password supplied by him, encrypted so that others cannot read it back.

It is necessary that some cards be nominated as "controller" of the hypertext. As well as containing the program that supports the Liveware mechanism itself, these perform some other functions. For example, they declare the name and purpose of the database and may identify some person who is responsible for its entire operation, and to whom users may direct enquiries. That person may be empowered to introduce new owners into the Liveware, and to reset passwords for owners who have forgotten them. He or she may also be able to remove information and eliminate owners. When a database is first designed, the facilities that its controller is to provide are specified too.

A user is "enlivened" with a particular Liveware database simply by giving him a copy of it. From then on, his database will be updated automatically whenever the opportunity arises. However, there is no permanent effect: to rid himself of it he simply deletes all copies in the normal way and no trace will remain. (This is quite unlike a virus.)

The essence of Liveware is that information is merged from separate versions of the database whenever possible, updating cards that are obsolete and introducing new ones. Whenever a Liveware database is entered, the system searches the computer's disc drives for other versions of the same database. If any are found, a merge is performed. This operation examines each card in turn, updating it if a more recent card with the same signature and identification code appears in the other version. Cards which appear in one version but not the other are copied. After merging, both versions of the database are identical. The activity occurs quite automatically and invisibly, without the user's intervention.

Users may make slips and accidentally corrupt the database. Many accidents delete information, which Liveware is well set up to restore through the normal merging procedure. Indeed, a sufficiently large community of Liveware users can protect each other from disasters like disc crashes. Thus if our disc dies on us, destroying our Liveware, we have only to take a blank disc to a friend whom we recently visited to get an up-to-date version. Other accidents introduce new—but trivial!—information, through mistakenly typing over someone else's entry, for example. Since Liveware is intended to propagate new information, it is certainly very important to protect against accidental changes, particularly since they may be more recent than—and hence overwrite—genuine updates made elsewhere.

So, to make things safe, when a user wants to add or update information, he must "log in" to the Liveware by supplying a secret password. This identifies him as the owner of some or all of the cards. Once logged in, the user may change information he owns, and the Liveware mechanism takes note of any changes. It is not possible to implement a really secure log-in mechanism on a personal computer without hardware support. Determined hackers could overcome any security system that might be devised. In the loosely distributed system that supports Liveware, they can have all the uninterrupted time they need, and work completely unobserved. Anyway, as a last resort, they could re-implement an exact look-alike system (a so-called *Trojan Horse*) without a great deal of difficulty, and it is quite impossible to prevent this. Under these circumstances, Liveware can do no more than provide the *appearance* of security. That is sufficient to protect databases against accidents, and perhaps against the idly curious. Given human nature, or rather certain inescapable manifestations of it, it is an unfortunate fact that for large-scale applications, hardware implementations of security will be necessary.

Perhaps the most unusual aspect of Liveware is the intertwining of social and technical issues that it stimulates. It effectively requires the imposition of a certain social etiquette. Specifically:

> Owners are responsible for their own cards.
> Owners are not to change other owners' cards.
> Owners are known publicly by their names.

This is normal information ownership: what's ours is ours, what's yours is yours, and we all know who's who. These rules apply even when information is not personal: they ensure that precisely one person is responsible for each card and that there is no possibility of a card being updated at different times unless the last update is required by the actual owner. The final rule is required because information owners may join the same Liveware in different places and times: there must be a naming convention to avoid any later conflict with ownership.

Liveware cannot work correctly if this convention is flaunted. It is assumed that when two cards with identical identification meet (during merging), then the one with the most recent time stamp can supersede the other. This assumption is not correct if gratuitous updates can be made at a later time than real updates, for instance if we correct a spelling mistake in a card of yours, but at a time after you most recently updated that card. When these two cards meet, our minor correction would replace whatever corrections you had made.

It is worth emphasizing that the rules of information ownership may, in practice, be irritatingly restrictive. For example, it is not permissible to personalise cards by introducing notes, doodling, reformatting, correcting spelling, or whatever. One's own version of the database is *public* information, for it may be transmitted directly to others, and not just a private copy. And the consequences of altering someone else's card are not simply that the interference will be transmitted to others, but—much worse—that *his* updates elsewhere may be superseded and lost. The reason for this situation is that the time stamp is the maximum update time for *all* the information on one card; it therefore cannot distinguish one change from another on the same card.

There are various options for handling the registration of new information contributors. The simplest is to allow anyone to register himself as a new owner, and choose a password at the same time. This gives free access to all, but runs the risk of pollution of the database by unwanted information. At the other extreme is a centralised policy where new contributors must be registered by a person responsible for the Liveware database, who also gives them initial passwords. Numerous more elaborate schemes are possible: existing owners may be empowered to introduce new ones, or several may have to collaborate to propose a new one. In this case the Liveware may enforce collaboration in a single interactive session, or permit nominations to be stored on cards owned by the proposers so that the process may be distributed in time and place. If the process is distributed, the nominee could take his Liveware disc round potential proposers until he has collected the requisite number of nominations. Moreover, each owner's heritage may be stored to allow limits to be placed on the number of introductions that owners may participate in. All of these possibilities are quite simple to implement; the chief problem is in deciding which scheme is suitable for any particular purpose. It is an intriguing thought that Liveware highlights live issues of democratic process—and also provides a forum for their precise formulation and assessment.

An information owner may wish to delete part of a Liveware database. This is difficult, since the Liveware may already be widely distributed. There are several possible mechanisms. The first, which has the advantage of being extremely simple, allows a user to withdraw information *so far as he is concerned*, but does not address the persistence of the information elsewhere. A card can be declared "dormant," and dormancy propagates in the normal way, a record of the card being retained to prevent it being reinstated by future merging from other versions of the database. The second mechanism is to make information "self destruct" when it passes its expiry date, ensuring that cards disappear autonomously no matter how widely the database is distributed. However, here the need for removal must be anticipated when the information is created. The third mechanism is to chase a card that has been distributed by an "anti-card" that destroys it on meeting. This requires Liveware to record the recipients of information in order to distribute anti-cards correctly.

Overall, then, Liveware is an effective mechanism to support the distribution of computer information by social means. It has surprisingly wide applicability—especially considering the simplicity of the idea—whenever people need to share information using informal distribution channels. Although originally conceived for use in situations where information changes slowly and users are not overly concerned with getting the most up-to-date versions, it can be used more generally because users can gauge the currency of their information and impose additional social conventions on the information flow. An important special case is when there is only one user: Liveware is ideal for backing up personal files and for sharing one's information between several workplaces. Another application is to support news or mail networks. If you know somebody who knows somebody ... who is witnessing news (and every intermediary has computers or can pass on discs), then you have an opportunity to keep up to date with that news. Just how "up to date" depends on social factors such as connectivity of the network and willingness of people to allow their computers to act as carriers. Liveware can manage news arriving in pieces via different routes, possibly out of order, possibly with losses.

Although we have used the metaphor of viruses to characterise the autonomous, invisible, merge operation that is at the heart of Liveware, the scheme, being benign, is nowhere near as virulent as it might be. A computer virus attaches copies of itself to other programs indiscriminately, seeking to spread itself as widely as possible regardless of the users' wishes. Liveware respects the right of the user to his computer and does not undermine his authority over it.

Ian Witten is with the Department of Computer Science, University of Calgary, Canada, and during the Summer of 1989 was supported by the Science and Engineering Research Council as a Visiting Fellow at Stirling. Harold Thimbleby is a Professor of Information Technology at Stirling University, Scotland.

Viruses and other nasty programs

The term "virus" is a popular catch-all for many kinds of malicious software. A "logic bomb" or "time bomb" is a destructive program activated by a certain combination of circumstances, or an a certain date. A "Trojan horse" is any bug inserted into a computer program that takes advantage of the trusted status of its host by surreptitiously performing unintended functions. A "worm" is a robust kind of distributed program that invades workstations on a network: it consists of several processes or "segments" that keep in touch through the network; when one is lost (for example, by a workstation being rebooted), the others conspire to replace it on another processor—they search for an idle workstation, load it with a copy of themselves, and start it up.

Viruses attach copies of themselves to other programs. They work by altering disc files that contain the compiled version of otherwise harmless programs. When an infected program is invoked, it seeks other programs stored in files which it can change, and infects them by modifying the files to include a copy of the virus code and inserting an instruction to branch to that code at the old program's starting point. After doing its work, the virus quickly starts up the original program so that the user is unaware of its intervention.

The idea of a maliciously self-propagating computer program originated in Gerrold's 1972 novel *When Harlie Was One*, in which a program called telephone numbers at random until it found other computers into which it could spread. Worms were also presaged in science fiction, by Brunner's 1975 novel *The Shockwave Rider*. The first actual virus program seems to have been created in 1983, as the result of a discussion in a computer security seminar, and described at the AFIPS Computer Security Conference the following year. In 1984 Thompson in his Turing Award Lecture showed how a self-replicating bug can infest a compiler or other language processor.

Virus attacks were not reported until a few years thereafter, and so far have been more in the nature of electronic vandalism than serious subversion. One of the first occurred in late 1987, when over a two-month period a virus quietly insinuated itself into IBM PC programs in a Jerusalem university. It was noticed because it caused programs to grow longer (due to a bug, it repeatedly re-infected files). Once discovered, it was analyzed and an antidote devised. It was designed to slow processors down on certain Fridays, and to erase all files on Friday 13 May.

At about the same time another PC virus invaded LeHigh University, while a much-publicized "chain letter" Christmas message spread itself by self-replication, clogging the BITNET network—it was eradicated by a massive network shutdown. Early 1988 saw a relatively harmless Macintosh virus designed to distribute a "message of peace," and a number of other viruses for personal computers. By that time talk about viruses had invaded the news media.

Late in 1988 a worm program was inserted into the American Internet computer network. It exploited several security flaws in SUN and VAX systems running Unix to spread itself from system to system. Although discovered within a few hours, it required a huge effort (estimated at 50,000 man-hours) by programmers at affected sites to counteract and eliminate the worm over a period of weeks. Again, it was unmasked by a bug (a bug in a bug!): under some circumstances it replicated itself so fast that it seriously slowed down the infected host.

Hypertext

"If Emma takes the parcel home, go to page 34; if Emma decides to open it here and now, go to page 14" ... a sentence from a 'branching story' of the sort that is becoming very popular in childrens' detective stories. and was once very popular for 'programmed learning texts.'

The same idea works nicely on a computer, but instead of paper pages, we have screens or cards (like 3 x 5 card-index cards) and instead of the reader turning to page so-and-so, there are 'buttons' that can be pressed, and the computer 'turns the page.' This is the basic idea behind hypertext: bits of text and pictures linked together in routes that depend entirely on the way the reader takes the story.

Stories and programmed learning are far from the only ways of using hypertext. First, since each 'page' of the hypertext is on the computer it can do ***anything*** a computer can do. Secondly, pages or cards can contain text, pictures and even sound effects as well. In a programmed learning text 'buttons' are always the same bit of '**`If you think the answer is 5 turn to page 1234`**,' but in Hypertext the button can be a picture, perhaps a real button like ● or anything else. The pictures of Liveware in the article give other examples of buttons; one is a magnifying glass, suggesting (we hope) 'press this and take a closer look,' which is what it does.

Two examples:

• The BBC in conjunction with Apple Computer have produced a hypertext document designed for teaching about nature conservancy. The *BBC Ecodisc* is a large simulation of a nature reserve that allows students to get a feel for what running a nature reserve involves. They can take walks around the reserve, as they press buttons on a map of the reserve they are taken from place to place. On a card talking about the wood, for instance, the student can press a button to have a closer look at the flowers, and he will see a photograph of the flower in summer, or whatever time of year it is. Or they can stop to listen to the sounds, perhaps a particular sort of bird calling. At any point, the user can consult experts, that is every card has a button that brings up an expert's prerecorded film—in sound and colour—discussing some interesting feature of the habitat.

• *Glasgow Online* is a different sort of hypertext to help tourists to Glasgow find tourist attractions. Some pages are maps of Glasgow, containing lots of buttons covering each site of interest, like cinemas or museums: other pages are indexes, so that a tourist could find information about, say, accommodation. shops, places of worship, or leisure activities. On pressing the 'accommodation' button, the user would be taken to a further page asking him to choose between hotels, guest houses, self-catering, hostels or camping facilities. On pressing one of those buttons, he gets, for instance, hotels listed by name, and then he can find out more about each hotel, for instance where it is on the map.

Liveware terminology

To underline the more positive social connotations of Liveware as opposed to viruses, it is necessary to use new terms for concepts otherwise known as infection and perpetration. Here are some terms used in the article.

- *Liveware* — A hypertext (or other) database that updates itself autonomously whenever the occasion arises
- *Enliven* — To inoculate a person's computer with a Liveware database
- *Information owner* — an owner of one or more cards in the database, and the only person permitted to change them
- *Database owner* — the person responsible for the Liveware database as a whole. He is *not* empowered to alter information belonging to others
- *Signature* — a code identifying an owner, including his full name and perhaps an encrypted secret password that only he can generate
- *Livestamp* — the Liveware information recorded on each card: signature, identification code, and time stamp
- *Merge* — the joining of two Liveware databases together so that both contain the most recent information.

An example of Liveware

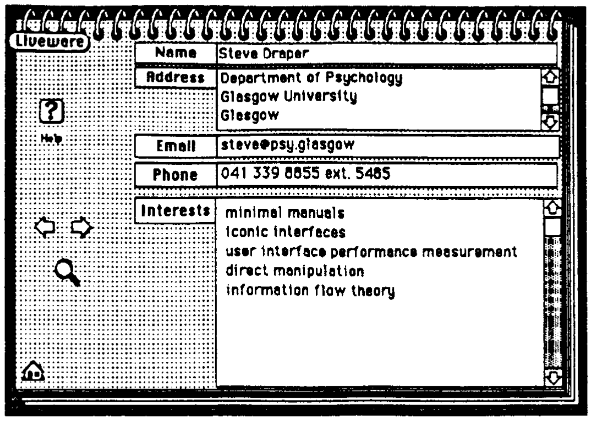
A pilot Liveware hypertext database has been implemented which stores information on research workers in Scotland who are interested in human-computer interaction. Setting it up in the conventional way would have required soliciting information from everyone, encouraging them to respond, collating data, and presenting it. It seemed that a far more effective way would be to distribute as much of the information processing as possible. Instead of a paper questionnaire, an outline database could be distributed on disc. Interested people would fill in their details—doing their own data entry—perhaps copying their part of the database on to further colleagues. Completed databases (that is, locally completed) would be returned for central, automated collation. We supposed that respondents would be further encouraged, since if they filled in and returned their database, they would receive a complete collated database in return. It was a short step from this practical application of distributed databases to also distribute the collation process itself, and a virus-like mechanism was suggested.

The purpose of the database is to supply information on each person registered in it. It provides a card for each person that records his or her name, address, phone number and email address, and a list of one-line phrases describing research interests. A complete list of people's interests is automatically collected together on a separate card. Clicking on a line of this index takes you to the card of the first person who has declared that interest; repeated clicking cycles through all relevant people's cards.

The front card includes a list of people represented (collected automatically on entry to the stack by examining all cards in it); clicking on a line will go to that person's card. The '**Add a new person**' button allows new owners to register; they enter a dialogue that solicits their name and initial password, and a card is created which they can fill in. Also included on the first card is a list of versions of this stack. Other cards include information and help about the Liveware system itself, and a summary of the social conventions discussed earlier.

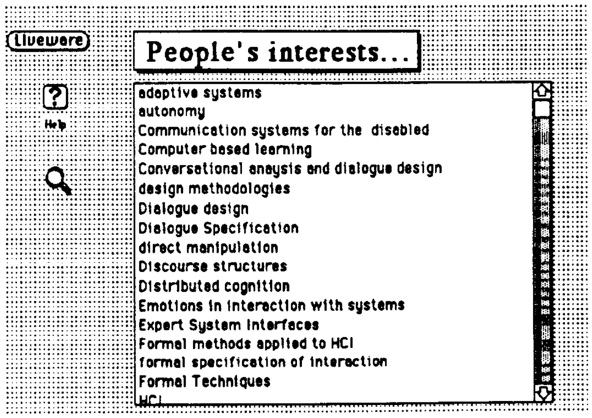
The database is implemented in the Apple HyperCard system, and the merge operation that is at the centre of the Liveware idea is written entirely in HyperTalk, a programming language provided with HyperCard. As the system locates new versions of the stack on disc, or as the user adds new versions manually to the list, they are used to update the current database and, conversely, it is used to update them.

Each card has an owner name and password, and the '**Liveware**' button at the top left of each card gives controlled access to this hidden information, showing the owner and allowing one to log in or change the password. Passwords are used to impose a degree of integrity on the information. They are encrypted on entry by an encryption facility that is built in to HyperCard, and stored (invisibly) on each card in encrypted form. The same password is stored on all cards that belong to a given owner, so that cards are self-contained and can be treated independently during the merge operation.



Name	Steve Draper
Address	Department of Psychology Glasgow University Glasgow
Email	steve@psy.glasgow
Phone	041 339 8855 ext. 5485
Interests	minimal manuals iconic interfaces user interface performance measurement direct manipulation information flow theory

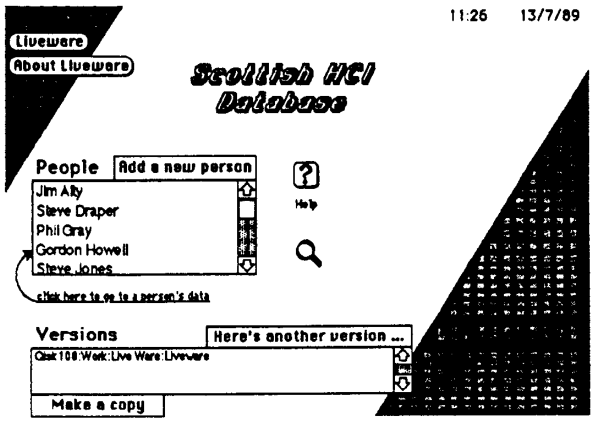
Figure 1: Information on a person in the Scottish HCI Database. The main part of the database consists of many cards of this form. Note the 'Liveware' button at the top left.



People's interests...

adaptive systems
autonomy
Communication systems for the disabled
Computer based learning
Conversational anaysis and dialogue design
design methodologies
Dialogue design
Dialogue Specification
direct manipulation
Discourse structures
Distributed cognition
Emotions in interaction with systems
Expert System Interfaces
Formal methods applied to HCI
formal specification of interaction
Formal Techniques
HCI

Figure 2: The list of research interests in the Scottish HCI Database. This collects all research interests mentioned in the database. Clicking on one takes you to the card of a person with that interest.



Jim Alty
Steve Draper
Phil Gray
Gordon Howell
Steve Jones

Disk 100:Work:Live Ware:Liveware

Figure 3: The front card of the Scottish HCI Database. Clicking on a person's name takes you to his card. The 'Versions' field shows where versions of this database have been found in the computer system. The 'Make a copy' button at the bottom is provided to make it easy to copy the database for someone else.