University of Essex

Department of Computer Science

INHABITED INTELLIGENT ENVIRONMENTS GROUP

TECHNICAL REPORT

CSM 612

Adam King
Victor Callaghan,
Graham Clarke,

July 2005

Developing Intelligent Surfaces for Pervasive Computing – Architectural Issues

Adam King (amking@essex.ac.uk), Vic Callaghan (vic@essex.ac.uk), Graham Clarke (graham@essex.ac.uk): Department of Computer Science, University of Essex, Colchester, Essex, CO4 3SQ, United Kingdom

***Abstract*— The notion of an Intelligent Surface (iSurface) is the topic of several futurists' visions. They predict the use of Smart Matter, nanotechnology with computational ability, as ubiquitous; literally everywhere. It will be sewn into our clothing, or painted on the surfaces of our environments. The futurists suggest that we will use these surfaces as video displays, user interfaces, and composite sensor arrays. The implication is that a coat of paint will be enough to add this functionality to a surface. In this paper we argue that such a vision is fundamentally flawed. Such a surface can be seen as an amorphous computer – a multitude of identical tiny computers with local communication capability. Much work has been successfully carried out on providing functionality to idealised simulated amorphous computers by groups such as the MIT Amorphous Computing (AC) group. An iSurface is a specialised version of an amorphous computer using a simulator with a physical grounding. We have implemented one of the AC group's experiments to show how the realities of an intelligent surface adversely affect a successful amorphous computing project.**

I. Introduction

"Today, people's domestic spaces are becoming increasingly 'decorated' by electronic or computer based artefacts (gadgets) varying from mobile phones, through CD players, to transport systems and beyond".[1] This conjures the vision of a rich, dynamic ecosystem of interacting devices with computational capabilities. The advent of nanotechnology opens up new possibilities for pervasive computing; mass-manufactured nano-devices could literally saturate the environment. "Smart matter" [2] could be everywhere; sewn into the fabric of our clothes, or painted on the surfaces of our homes as an "intelligent" skin. Future visions [3] of the home suggest that applications such as video and user interfaces will be commonplace on these surfaces, and that they will also act as composite sensors such as video cameras. The implication is that a simple coat of nano-scale device bearing paint will be enough to add this type of functionality to a surface. Such an "intelligent surface" (iSurface) can be seen as a specialisation of an Amorphous Computer [4] - a multitude of identical tiny computers (particles), each with a CPU, memory and communication capability. A possible precursor to the nano-scale particles can be seen in the Smart Dust mote [5], a MEMS based sensor and processing node.

An amorphous computer is a massively parallel system limited to local communication between neighbouring particles. Amorphous computing is the development of organisational principles and programming languages for obtaining coherent behaviour from the interaction between large numbers of unreliable particles that are interconnected in unknown, random, and time-varying ways. Investigations into massively parallel systems, such as cellular automata, are one source of ideas for dealing with amorphous systems. An alternative is research into self-organising systems which Abelson et al [6] suggest offers a possible solution which is to embed all the required code into the particles at the time of manufacture. With an amorphous computing system where each particle has this code preloaded, the particular functions that are activated within any particle depends on the messages it receives from the local environment. The specific issue that makes this a hard problem, is predicting the range of functions that must be pre-coded given the vast number of possible states the system can be in. Butera argues that: "A programming model employing a self-organising ecology of mobile process fragments supports a variety of useful applications on an [amorphous computer]." [7]. In support of this, he offers a distributed programming methodology, known as 'process self-assembly'. A programming model is introduced, as is the "process fragment" - the atomic element of process self-assembly. His project demonstrates the feasibility of a mobile agent paradigm.

II. Intelligent Skin (iSurface) Requirements

Returning to the future vision, we see that what are proposed as applications for an iSurface - video, composite sensors and user interfaces - are all data-heavy, time-critical applications. For this to work an iSurface needs to be able to display images and transfer large amounts of data. It needs to be able to locate elements of an interface and allow the ability to click buttons and to drag and drop interface elements. It also needs to support connections between the functional features of the iSurface. The question is whether an iSurface, derived from the work on Amorphous Computing and Smart Dust, would be a feasible platform for these applications. If so, then what is necessary in terms of capabilities and software? If not, then what are the stumbling blocks that prevent the concept being a success? This paper presents experimental results, based on a critical application for an iSurface, which suggest that this vision is unrealisable now, or in the future, because of a number of severe problems. Specifically run-time damage, poor response times, and, most importantly, the load on the particles' resources.

III. Emergent Pathways

A major application problem that an iSurface would need to solve is the reliable transmission of data from one area to another. Communications on an iSurface are generally undirected and a method involving propagation of a signal throughout the network is inefficient, albeit guaranteed to reach its target if any path exists. Directed routing of data is possible. Use of a coordinate system allows messages to be passed in the right direction by comparing the current node's coordinates to those of the target node. However the problem lies with the reliability of such coordinate systems and the requirement of a signal knowing its target coordinates. Another solution is the creation of "channels" for the signals to pass along. A channel consists of an uninterrupted line of iCells between two or more points that relays signals along its length. It is possible for a user to explicitly define a channel between two points but should damage occur and this channel be interrupted, a user is needed to make manual repairs.
Clement and Nagpal [8], proposed a process of growing connections between two endpoints and for these connections to be self-repairing in the event of damage to an amorphous computer. As an iSurface is a specialisation of an amorphous computer, this process should be implementable on the iSurface. Clement and Nagpal's aims appear to be based on the creation of shapes rather than developing connections for communication, but their approach provides the foundations for a possible solution.

This Amorphous Computing approach builds upon gradient fields emitted from one of the endpoints of a line. The source endpoint uses a high value for the gradient to begin on. This value is broadcast to all neighbours in the form of a message tuple {Processor ID, Gradient Value, State, Successor ID}. Processor ID is a value randomly chosen by the particle (iCell) as a unique identifier. Gradient Value is the gradient at that particle. State denotes whether that particle is part of a line. Finally, Successor ID is the ID of the particle that provided the highest gradient value to this particle. When a particle receives this message, it compares the gradient value with the one it has stored. Should this new value be higher, the particle decrements it and stores it along with the ID of the sender as the Successor ID. The successor node allows the gradient field to exist as a chain from each particle back to the gradient source; no data aside from the gradient value and the successor ID need be stored.

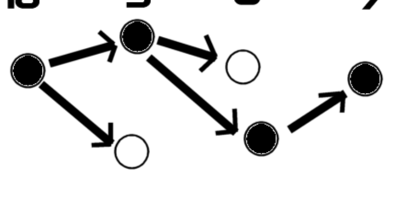

Figure 1. The node with a value of 10 is the gradient source. Reversing the flow of the arrows identifies successor nodes.

When the predetermined endpoint of the line receives a gradient value, it changes its State to on (meaning it considers itself to be part of the line). It then broadcasts a message including the new state. Its successor node will detect that it is identified in the message, switch itself to an on state and then broadcast its own message. The line forms by a process of backtracking through the successor nodes from the endpoint to the gradient source point.

IV. Self-Repair Using Active Gradient

As the line is entirely dependent on the chain of successor nodes in the gradient field, any alteration of the field will alter the successor chains and thus alter the line. This is the reasoning behind the self-repair section of the Amorphous Computing approach. This is called an Active Gradient approach - a gradient field that can adapt itself to situations such as surface damage thus causing the line to re-route accordingly. Particles need to periodically broadcast their state and gradient information and remember a timestamp based on the processors internal clock. When a particle hears from its successor it will update the timestamp. Should this timestamp become too old then the stored values for the gradient and successor ID are considered to be unreliable. In this event the particle will decrement its stored gradient value. As time passes with no signal from its successor this gradient value will continue to decrease. Eventually another neighbour's broadcast gradient signal will be higher than the stored gradient, thus meaning that the neighbour is now closer to the source. This gradient is adopted and the successor link replaced accordingly. Using this approach causes the gradient to slowly adapt itself to damage. According to the theory this will also cause the line to adapt to this new gradient i.e. self-repair.

The iSurface simulator on which all experimentation was performed simulates a grid of 65536 (256 by 256) instances of an iCell. The iCell is a completely self-contained simulation of a "real" iCell that maintains its own message buffers for I/O and its own list of Agents. It is capable of communicating separately with each of its neighbours, assuming exclusive full-duplex communication with each. Each communication in this simulation is the same size, each cycle length equal to the time to transmit a message of this size; the result being that an iCell can transmit one message to its neighbours per cycle. Any other messages that the iCell tries to send during the cycle are added to a queue for transmission in future cycles. In this simulation, the message size, and hence communication time and cycle duration, are determined by the gradient information message. The Agents used in this simulation are stored as hard-coded classes that are instantiated by each iCell. They spread across the iSurface by replicating themselves on startup to all the "uninfected" neighbours of their host iCell. Each Agent has access to the sensing, effecting, processing, and communications capabilities of their host iCell. However, Agents on the same iCell are not necessarily aware of each other's existence.

For the purpose of implementation, the way gradients are handled is altered. Zero is now the start point, and values increase the further away the particles are from here. This brings the gradient system into line with the system used in other iSurface experimentation, but, as the systems are isomorphic, this makes no difference to the workings of the gradient field.

V. Comparison Of Three Types Of Gradient Creation

While implementing the Amorphous Computing approach on the iSurface it was discovered that three methods of creating a gradient using much the same algorithm were possible.

The algorithm for the first approach is exactly as described earlier. When an iCell receives a gradient information message, it increments the new gradient value and compares it to the stored value. Should the new gradient be lower, it will replace the current gradient and the originator of this new gradient value becomes the new successor. Then a gradient information message is sent with the updated information to all of the iCell's neighbours. This approach establishes a complete gradient in a single pass and follows up with "update waves" whenever the agents send their periodic update messages. The advantage of this approach is that it is relatively quick to propagate and form a complete gradient. Figure 2 shows the total bandwidth, for the entire iSurface, used to complete a gradient with respect to time for both the first, single-pass approach and the second, multi-pass approach.

The second, multi-pass approach is identical to the single-pass system except that when it receives an update message from its successor node, it will update its information to match this input, even if the new information is worse than its current value. This has the effect of breaking down the data-heavy single-pass result and creates an incomplete gradient, refined by the periodic update messages (Figure 2).

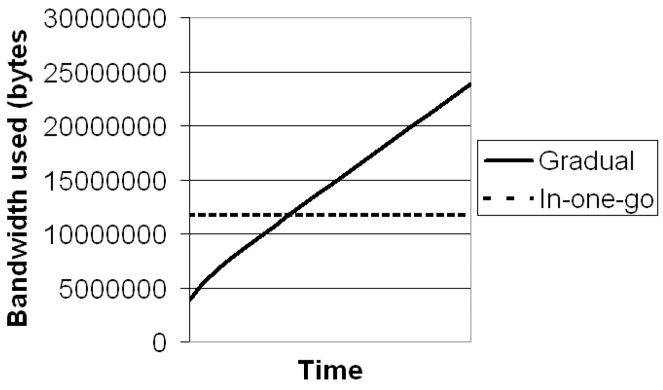

Figure 2.

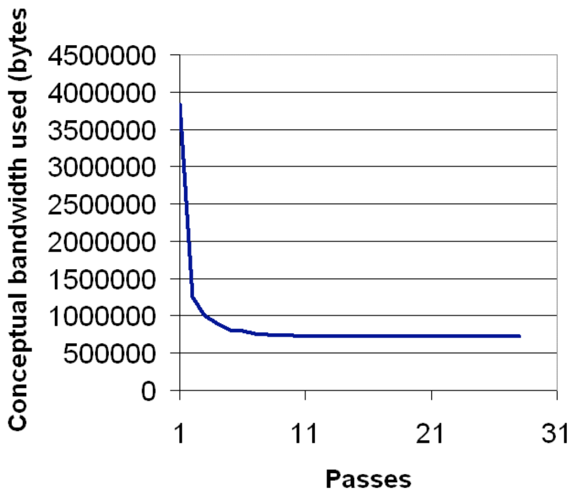

Figure 3.

To begin with the multi-pass system creates an incomplete gradient for much less bandwidth cost than the single-pass approach. However, the update passes carry a fixed cost as well as costing more bandwidth when newly updated iCells send their updated gradient message to their neighbours. After several passes the total cost far surpasses that of the single-pass approach. Figure 3 shows the drop off in cost as these passes continue. The initial pass, like the single-pass, takes up a considerable amount in order to lay the foundations for the incomplete gradient to be refined. During the latter passes very few updates occur, and thus there is no significant extra bandwidth cost other than that of the update wave itself. The initial high bandwidth is what causes the offset on the y-axis and the slight curve in Figure 2. As the bandwidth usage levels off, the increase of total bandwidth in figure 2 becomes linear.

The third version of the algorithm takes the multi-pass approach but removes the behaviour of immediately broadcasting to its neighbours the moment its information is updated. Instead, the system relies on the gradient message periodically sent out by the iCells in order to grow and maintain the gradient. Thus the time it takes to complete can be tied to the rate at which the periodic updates occur.

VI. Responsiveness For Gradient Creation

Figure 4 details the data rates of the single- and multi-pass approaches. The data rate is defined as the average number of bytes transferred per iCell per cycle. Figure 5 conversely shows the time taken for both approaches to complete the gradient fully. "All-in-one" indicates the first method. The numbers represent the second method, specifically the time between periodic updates.

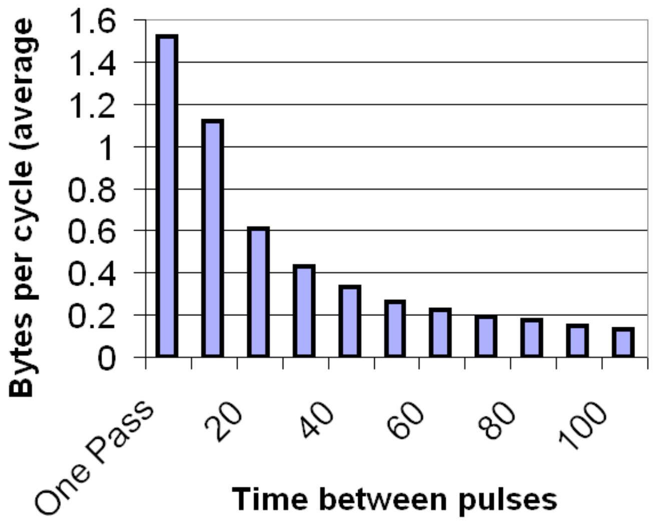

Figure 4.

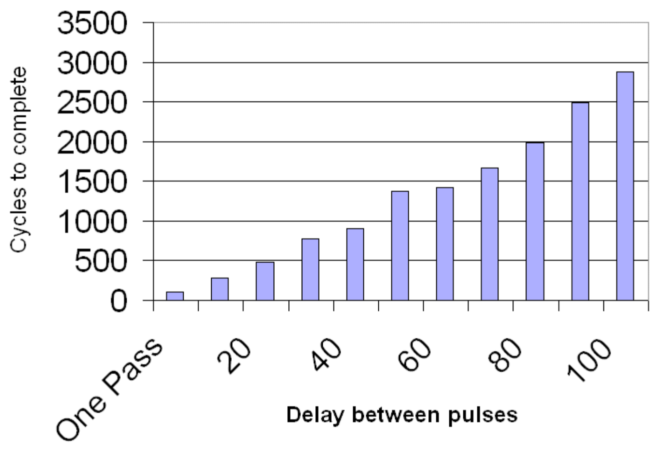

Figure 5.

As we saw in figure 2, the single-pass approach utilises a briefly sustained high-bandwidth burst to create a complete gradient. By working in waves, the multi-pass approach can keep its data rates low. However, these low rates need to be sustained for significantly longer and also result in an overall higher total bandwidth cost.

VII. Dealing With Surface Damage

One of the stated aims for this application area is the ability for the line is to adapt to damage to the iSurface. For the purposes of experimentation it was decided to use a similar form of damage to that presented by Clement and Nagpal. All damage repair experiments discussed here utilise a large, solid rectangle cut out of the centre of the surface, bisecting any line present there.
The single-pass approach was used to create an "ideal" target gradient by growing on a pre-damaged system. All experiments with gradient repair were compared to this ideal to find out when the repair was complete. The first problem that became apparent was that the single-pass system was incapable of adapting itself to damage in any significant way.

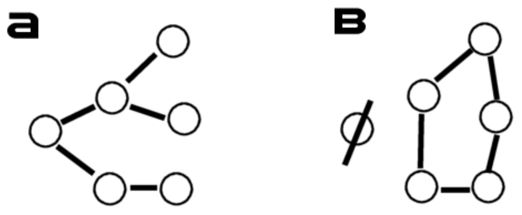

Figure 6.

Figure 6 is an example of what happens when the single-pass approach tries to deal with damage. In "A" we see the successor chain. In "B" the root of this chain is dead and its children detect this. They then alter their gradient, as their programming requires. However, they subsequently form self-contained loops that constantly pass gradient information between themselves, even though that information is obsolete. This is due to an Agent only changing it's stored gradient if a lower value comes along, or if its successor node is dead.
The multi-pass system doesn't suffer from this problem. The difference being that gradient change is passed down the successor chain and Agents update themselves according to their successor data instead of waiting to adopt the lowest available gradient as a successor. In this way the altered gradient propagates through the successor network and thus aids adaptation.

Figure 7 shows the bandwidth used by the multi-pass approach as it accumulates with each pass of the update wave. As it increases linearly with a fixed rate we can see that the majority of the bandwidth is simply taken up by the update waves' activity. There is no repeat of the behaviour observed in figure 3 where large sections of iCells updated at once causing a bandwidth rise. We can conclude from this that the repair of the surface takes place very slowly on a very small scale each cycle. However, compare the bandwidth used by a single-pass complete reconstruction (experimental results provide an average of about 1,400,000 bytes across the surface) to that used by the multi-pass repair (77,177,375 bytes, see figure 7). Despite the higher data-rate (as evidenced in figure 4) used by the single-pass, it seems more efficient to scrap the damaged gradient and start from scratch.

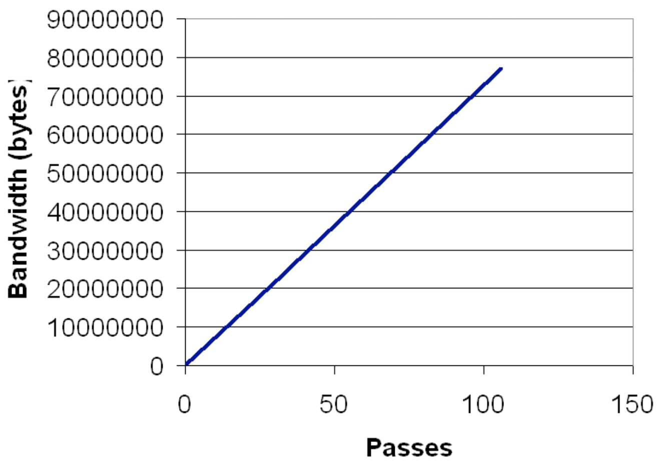

Figure 7.

However, detection of the damage in order to reinitialise the gradient would be a problem. The gradient needs to be reinitialised at the source. This means that the source needs to determine if damage has occurred. This can be accomplished by regular signals being sent along the wire between the source and the destination. Should a predetermined length of time expire without such a signal, the source may determine that the line is broken and start a re-initialisation. There are many issues of responsiveness with this.

VIII. iCELL LOAD

iCell load is the overheads of an iCell's processing and communications resources generated by the Agents and messages resident within them. When significant pressure is placed on the communication system, massive backlogs appear which can be fatal for time critical applications. For the purpose of experimentation, a special agent was created to inhabit an iCell alongside the main agent. This new agent generates a random amount of noise that will overload the communication systems of iCells and cause backlogs.
Figure 8 shows the amount of data generated by four levels of noise at various concentrations. Real world applications would generate far more than this, but the simulator is unable to cope because of the demands placed on the host machine. The gradients of the lines in the graph follow a series of the form
y = m(x^c)
allowing calculation of the resulting data rate for any concentration of any noise level.

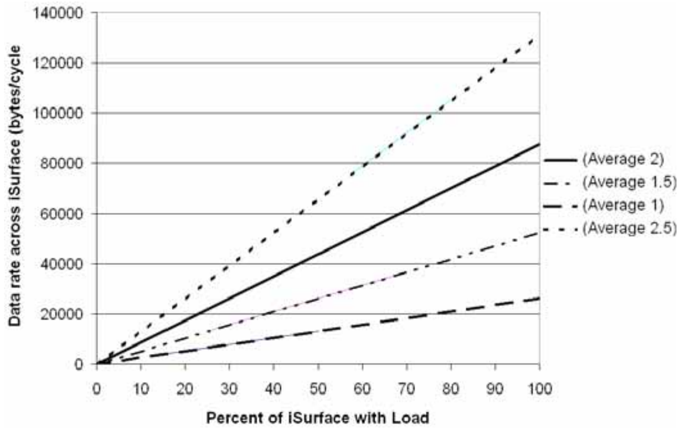

Figure 8.

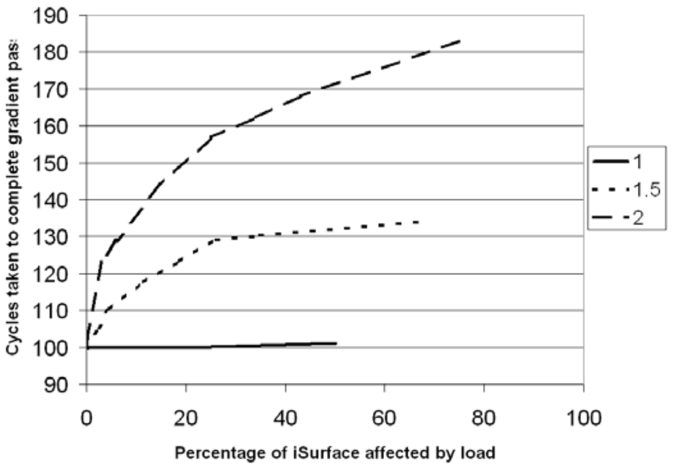

Figure 9.

Figure 9 demonstrates the effect this iCell load has on the single-pass approach. An unencumbered system takes about 100 cycles to complete. We can see that noise level 1 had little effect. As noise levels increase in density we see that they have a significant delaying effect on the propagating gradient. The high data rate of the gradient data compounds the problems caused by the high data rate of the noise making agents.

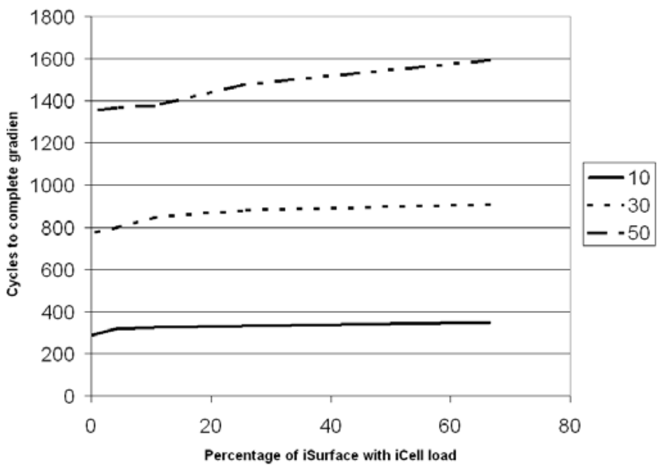

Figure 10.

Figure 10 shows the results of applying noise level 1.5 to three speeds of multi-pass gradient creation. The speeds refer to the time between passes. We see a similar increase in time as density increases for all three speeds; this is due to the waves themselves being delayed. The real differences in time are simply due to the delay between sending passes. The actual data rate of the passes is too low to compound the problem to any real extent.

IX. CREATING THE LINE

Utilising the successor chains created as part of the gradient, the line develops exactly as expected. However, at the time of writing, attempts to get the line to respond to changing gradients cause a highly localised concentrated form of iCell load to occur. Agents in their line state retransmit while the gradient field is in flux. This causes new lines to be created. These new lines are quickly broken by the changing gradient, but not before spreading onwards. Lines can be created faster than the timeout that destroys broken lines. Coupled with the gradient information messages, these cause queues of messages that continue to increase, eventually crippling the iCells.

X. Conclusions

Clement and Nagpal's line drawing system can be successfully ported over to a "real-world" Amorphous Computer derivative such as the iSurface. The general principle of using this system to directly link two areas is sound; the resulting gradient and line are robust when dealing with pre-damaged surfaces and the gradient itself is successful at dealing with repairing itself. Clement and Nagpal themselves say that responsiveness is determined by the timestamps used to control the updating of the gradient. This has been shown to be true in this system. However, using this system as a communication channel between two or more points requires reliability and this would entail quick responses to damage, both diagnosis and repair. The obvious solution would be to increase the rate of gradient update passes. However, as we have seen, high data rates contribute heavily towards iCell load, which in turn lowers responsiveness of the system. To compound the problem increased responsiveness means the timestamps expire sooner. iCell load delays the necessary messages to renew the timestamp and so it expires. This is a false result caused by data being delayed.

A possible solution to the problem would be to prioritise tasks on the iCell. This would involve the host iCell allocating its resources to the agents based on need, and there are established methods that could be adapted to do this. The result would be areas of the iSurface specialised towards certain applications, which is certainly feasible. However, much of the common functionality, such as the system presented in this paper, is time-critical. By assigning priority to one agent the others will suffer, and, following the example of this paper, these time-critical systems will have to become less responsive to compensate for the lower allocation of resources. The result may be optimal given the system, but it probably will not be acceptable in terms of responsiveness.

This problem is typical of applications proposed for an iSurface or similar devices. These applications demand responsiveness. To obtain this, the solution is usually to increase data rate. However, an increase in data rate can lead to iCell load and compound the error. A simple answer is to increase the capabilities of the iCells in terms of processing speed and communication throughput. Every time a similar problem occurs, the answer would be to increase these capabilities. Eventually the capabilities required get so high, that the devices they require are so far into the future that the applications themselves become redundant or achieved by other means.

References

[1] Callaghan, V., Clarke, G., Colley, M., Hagras, H.: "Embedding Intelligence, Research Issues for Ubiquitous Computing", The 1st Equator IRC Workshop on Ubiquitous Computing - Nottingham UK, September 2001.

[2] PARC: "MEMS / Smart Matter Research at PARC", Available at http://www2.parc.com/spl/projects/smart-matter/

[3] Masens, C.: "Smart walls, smart windows, smart bricks and tiles..", July 2004, Available at http://www.smarthouse.com.au/articlesbytopic/homesystems/automation/1685

[4] Katzenelson, J.: "Notes on Amorphous Computing", MIT Artificial Intelligence Laboratory, 2000, Available at http://citeseer.ist.psu.edu/325443.html

[5] Pister, K., Kahn, J., Boser, B.: "Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes. Highlight Article in 1999 Electronics Research Laboratory Research Summary", 1999, Available at http://robotics.eecs.berkeley.edu/~pister/SmartDust/

[6] Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T., Nagpal, R., Rauch, E., Sussman, G., Weiss, R.: "Amorphous computing", Communications of the ACM, 43(5), May 2000.

[7] Butera, W.: "Programming a Paintable Computer", PhD Thesis, MIT Media Lab, 2001

[8] Clement, L., Nagpal, R.: "Self-Assembly and Self-Repairing Topologies", Workshop on Adaptability in Multi-Agent Systems - RoboCup Australian Open, January 2003, Available at http://www.swiss.ai.mit.edu/projects/amorphous/papers/arobocup-2003.ps