In the First International Conference on New Technologies, Mobility and Security (NTMS'2007), Telecom Paris, France, 2 to 4 May, 2007

# Preventing Instability in Rule-Based Multi-Agent Systems; A Challenge to the Ambient Intelligence Vision

Victor Zamudio[1] and Vic Callaghan[1]

[1] Department of Computer Science, University of Essex. Wivenhoe Park CO4 3SQ, United Kingdom
{vmzamu, vic}@essex.ac.uk

**Abstract.** Multi-agent systems underpin the vision for ambient intelligence. However, developing multi-agent systems is a complex and challenging process. For example, pervasive computing has been found susceptible to instability, due to unwanted behaviour arising from unplanned interaction between rule based agents. This instability is impossible to predict, as it depends on the rules of interaction, the initial state of the system, the user interaction, and in the time delay of the system (due to network traffic, different speed of processing, etc). In this paper we present a theoretical framework, an Interaction Network (IN), together with a communication locking strategy that can be used to identify and eliminate this problem. We present experimental results based on simulations and a physical implementation that demonstrate the effectiveness of these methods.

**Keywords:** Agent Challenges, Pervasive Computing, Instability, Periodic Behaviour, Multi-Agents.

## 1 Introduction

Multi-agent systems underpin the vision for ambient intelligence. At the heart of this vision is the interconnection of vast numbers of devices such as lights, heaters, TVs, telephones, etc., each programmed according to a certain rules based on the state of the world, including other devices These interconnections enable the system to be programmed with interdependent actions in a simple way, whether it be manual or automatic [1, 2, 3]. This is a very challenging problem, not only due to the complexity of the rules of the interconnected devices, but also because some of the devices could be nomadic, and there could be synchronizations problems due to temporal delays (network latency, speed of processing, etc). These temporal delays could contribute to unstable behaviour. We have seen this phenomenon in our own systems and it is being observed increasingly in pervasive computing system as the architectures move from centralized to distributed control [4].

In other domains, such as complex and dynamic systems, it has been shown that it is not possible to determine, based on the rules of interaction, if a given system will suffer from instability [5]. However, it is possible to prevent it and we have developed and tested such a strategy, based on the detection of loops in an Interaction Network introduced in the next section, and a method for locking devices with least functional impact on the performance of the system.

 

## 2 Interaction Networks

A ***directed graph*** $G$ consists of a finite set $V$ of vertices or nodes, and a binary relation $E$ on $V$. The graph $G$ is denoted as $(V, E)$. The relation is called the adjacency relation. If $w$ is relative of $v$ (ie, $(v, w) \in E$) then $w$ is adjacent to $v$ [6]. An agent $A$ is an autonomous device with a binary state $s \in \{0,1\}$, where $0$ means that the agent is *off*, and $1$ means that the agent is *on*. If we have $n$ autonomous devices agents $A_1, A_2, \ldots A_n$ the state of the system is $S = (s_1 s_2 \ldots s_n)$. Each agent $A_i$ has two rules: i) if $\phi_i$ then $s_i = 1$ ii) If $\psi_i$ then $s_i = 0$ where $\phi$ and $\psi$ are boolean functions that depend on the states of the agents.

An ***Interaction Network (IN)*** is a directed graph $(V, E)$ in which the vertex $v \in V$ is a pervasive autonomous agent $A$ and $(v_i, v_j) \in E$ if the Boolean functions $\phi_j$ or $\psi_j$ of the pervasive autonomous agent $A_j$ depends on the binary state $s_i$ of the agent $A_i$. Let $U \subseteq S$ be a subset of $S$. Because of the dynamics of the system, the system will produce a sequence of states $U_1, U_2, \ldots U_p$. If this sequence of states is periodic then the subsystem $U$ is said to be *periodic*.
The ***functionality of a node*** is defined as the number of descendants in the Interaction Network. This characteristic of a node is very important, as it shows the impact of a device in the system, in terms of the number of devices whose rules could be triggered. Fig. 1 provides an example of an Interaction Network, showing the dependencies of 5 devices or services: Sofa Sensor, Light Sensor, MP3 Player, Light, and Word.

## 3 Intelligent Locking

In pervasive environments, rule-based devices could be interacting according to the rules programmed by several users. This complex rules, together with the state of the system and the times delays could lead the system to unwanted instable states (oscillations). As we have commented previously, it is not possible to predict, in general, if a set of rules will produce such instabilities, as in any dynamic system, the behaviour of the system will depend not only on the rules, but on the initial conditions. Besides that, the user could be interacting with the environment, generating perturbations to the system. However, the presence of a loop in the Interaction Network associated indicates that the potential problem of instability and cyclic oscillations could merge at any time. Our strategy to prevent this unwanted behaviour is based on

1. the detection of loops in the Interaction Network
2. For each loop
    - Find the node member of the loop which minimizes the functionality function

- Lock this node

Step (2) includes learning from the user their preferred “locking preferences” (where there are choices to affect the user)

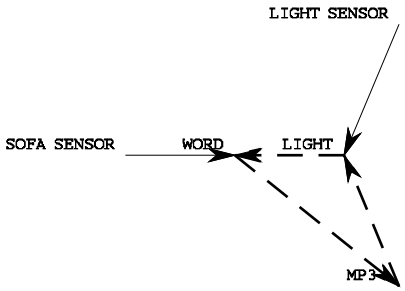


**Fig. 1.** Interaction Network showing a loop in dashed lines.

## 4 Experimental Results

Our strategy was tested in two ways, first with computer simulations and secondly using a real UPnP (Universal Plug and Play) implementation based around the Siemens Java SDK for UPnP technologies [9].

### 4.1 Simulations

The simulator was programmed using Mathematica ™ **5.1** [7], a programming language with powerful tools for quick and sound implementation. In particular, it includes the package Combinatorica, supporting graph theory, graphics, and combinatorics [8]. The simulation had the advantage over the real implementation (see next section) that it was able to mimic larger numbers of devices and support a more flexible experimental sturucture (eg arbitrary devices and rules could quickly be created).

Using Mathematica ™, a number of parameters can be controlled, for example the number of agents involved, the number of iterations, the probability of perturbations, the probability of interconnection between two agents. In order to test the general approach, we generated random topologies of differing densities (controlling the probability of connexion between agents). It is well known that the gates AND and OR (in conjunction with the negation) are able to reproduce any Boolean function. Using this principle, we assigned a random (and fixed) number of boolean function to

each device, as a rule of behaviour; thus the rules assigned to each agent could be represented as a binary string, where a 0 and 1 mean an OR and AND gate respectively. As mentioned before, besides the rules of interaction, one of the key factors involved is the initial state of the system (in this case, we always begin with a random initial state) which is then perturbed by user actions.

One of the important parts of the algorithm, besides the finding of loops, is the process of choosing which agent (or node) to lock. Thus, for each loop we would normally need to calculate the functionality of each node. However as, in our experiment, the functionality function of all the members of a loop is the same, we excluded the descendant members of the loop from the calculations.

We tested our approach successfully with different and randomly produced topologies and rules of interaction, together with random perturbations.

In Fig. 2 we have a system with 7 nodes. This topology has one loop {6,4,6}. The rules of interaction are coded as {0,0,0,1,0,1,0}.

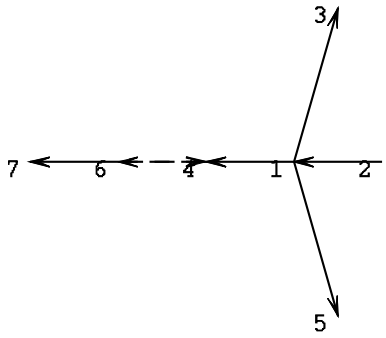


**Fig. 2.** Interaction Network with only one cycle {6,4,6}. Detail of the cycle in dashed lines.

Node 6 has 1 descendant, and node 4 has no descendants (as we do not include members of the loop). Node 4 has the minimum functionality, and as a result, the locking vector is {1,1,1,0,1,1,1}. In Fig. 3 we have the response of the system without any locking, showing cyclic instability; this instability can be removed effectively using the locking mechanism (see Fig. 4)

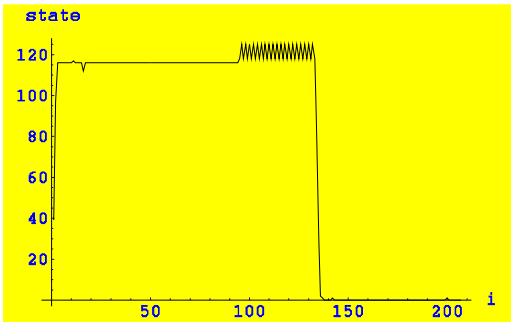

**Fig. 3.** Evolution of the system with 7 agents and one cycle {6,4,6}

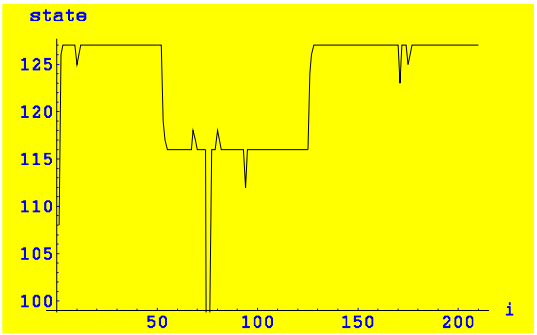


**Fig. 4.** Evolution of the system with 7 agents, one isolated cycle {6,4,6}, where the node 4 has been locked, and the oscillations have been removed

In Fig 5 we have a system with 10 nodes and two cycles {{8,5,6,7,8},{4,2,3,5,4}}, that share the node 5. The rules of interaction are coded as {0,0,0,0,1,1,1,1,1,0}. Before the locking mechanism was activated, the system showed instabilities (see Fig. 6). For our graphical representation we use the decimal equivalence of the binary representation of the global state of the system. The list of parent-descendants for the loops are {{4,1},{2,1},{3,2},{5,4}} and {{8,0},{5,6},{6,1},{7,0}}. In the first loop the two nodes 4 and 2 both have the minimum number of descendants (1), and 4 is taken. In the second case 8 and 7 minimizes the functionality function, and 8 is taken. With these results the locking vector for the system is {1,1,1,0,1,1,1,0,1,1}

We ran the simulation several times, and our strategy removed the oscillations, as can be seen in Fig. 7.

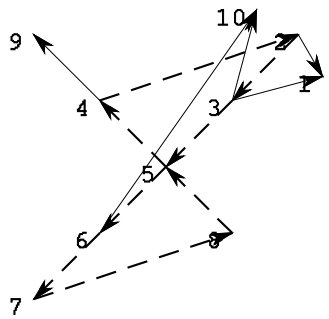


**Fig. 5** Interaction Network with 10 nodes and two coupled cycles. In dashed lines we have the two loops, sharing node 5.

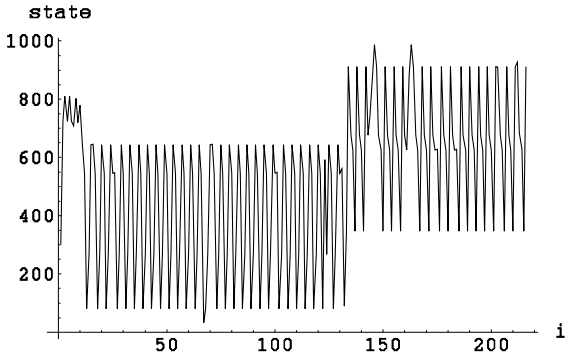


**Fig. 6** Behaviour of the system with 10 agents. Two modes of oscillation can be seen, together with some perturbations. In this case the locking mechanism has not been applied.

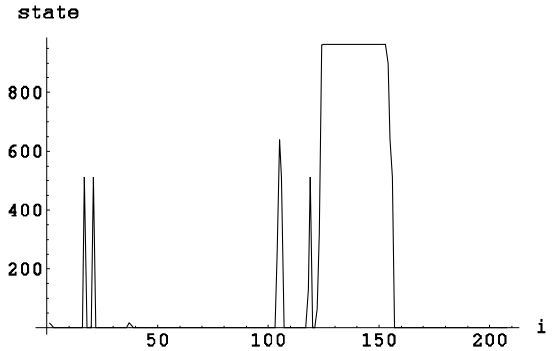


**Fig. 7** Behaviour of the system with 10 agents. The instabilities have been removed by the locking mechanism. Some perturbations are shown in the figure.

### 4.2 Testing Real Devices

We implemented the strategy in UPnP (Universal Plug and Play) based around the Siemens Java SDK for UPnP technologies [9]. An important difference to the simulation experiments is that this network includes both delays (eg propagation, stack handling etc) and user interaction (eg turning lights on off) that are a more accurate refection of a real environment. Thus, every device (lights in this case) has a user interface which allows the user to turn it on and off.

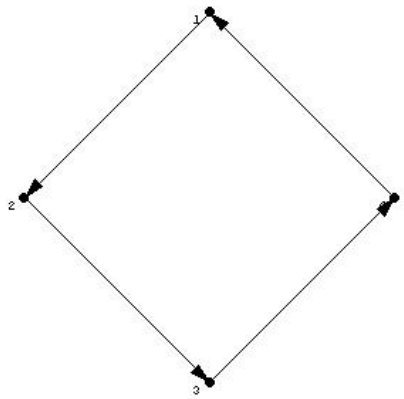


**Fig. 8.** Interaction Network (IN) of the experiment. In this case we have 4 devices (lights) with the rules defined previously.

The rules of interaction were set to try to emulate the state (non-inverted and inverted) of the adjacent device. The code can be seen in Table 1.

**Table 1.** Rules of interaction for the system with 4 devices.

```
//********rules for service1**********
if (  tempService[3] &&   lockService[0]){
scp.turnOn(testService1);
service1 =  tempService[3];
}
else if( ! tempService[3] &&   lockService[0]){
scp.turnOff(testService1);
service1 =  tempService[3];
}
//********rules for service2**********
if (  tempService[0] &&   lockService[1]){
scp.turnOn(testService2);
service2 =  tempService[0];
}
else if(! tempService[0] &&   lockService[1]){
scp.turnOff(testService2);
service2 =  tempService[0];
}
//********rules for service3**********
if (  tempService[1] &&   lockService[2] ){
scp.turnOn(testService3);
service3 =  tempService[1];
}
else if(! tempService[1] &&   lockService[2]){
scp.turnOff(testService3);
service3 =  tempService[1];
}
//********rules for service4**********
if (  tempService[2] &&   lockService[3] ){
scp.turnOn(testService4);
service4 =  tempService[2];
```

```
}
else if(! tempService[2] &&   lockService[3]){
scp.turnOff(testService4);
service4 =  tempService[2];
}
```

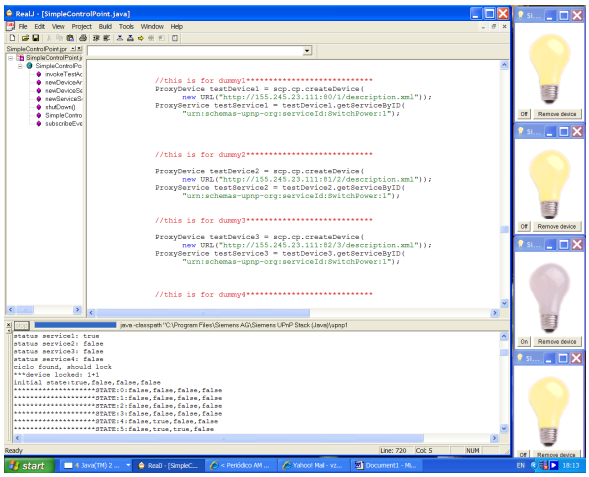

**Fig. 9.** In this figure we are showing a screenshot of the system running, with 4 devices (lights in this case). The lights are changing according to the rules, and the state of them is shown in the image.

In Fig. 10 se can see the behaviour of a system with 4 devices (lights) resulting in different modes of oscillations due to the perturbations from the user interaction.

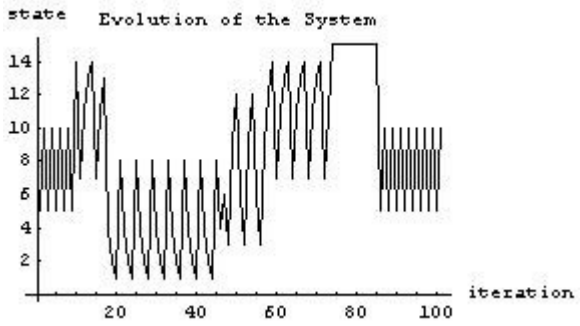


**Fig. 10.** In this figure we are showing the evolution of the system with two lights on as initial condition, no locking and perturbation of the user.

When the locking is enabled, the oscillations are clearly prevented, leading the system to a stable state (unless the user decides something else), as we can see in Fig. 11.

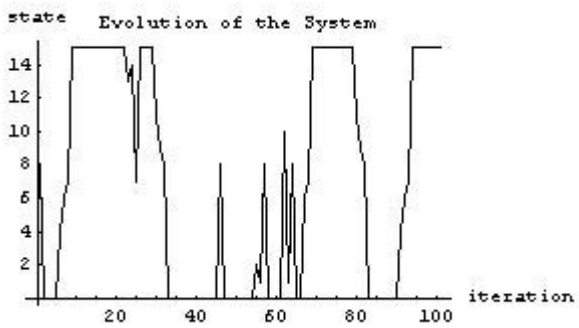


**Fig. 11.** In this figure we are showing the evolution of the system with one light on initially, with locking and perturbations from the user (due to the user interaction, we could have more than one light on at a time).

With 11 devices (lights), the strategy proved to work successfully. In Fig. 12 we can see a screenshot of the system running. Unstable behaviour was found under different initial conditions, but removed successfully, as can bee seen in Fig. 13.

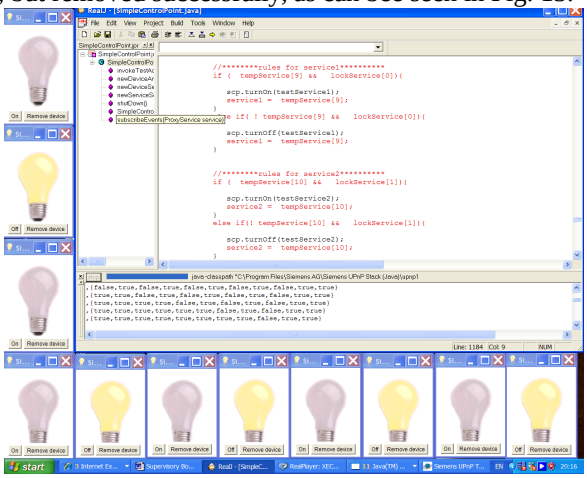

**Fig. 12.** Screenshot of the system, running the system with 11 devices. .

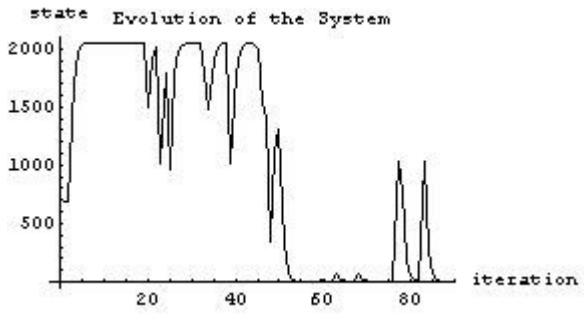


**Fig. 13.** Evolution of the system with 11 devices, implementing locking and showing user interaction.

### 4.3 Results Discussion

We have implemented and tested the strategy of locking using both computer simulations and real devices based on a UPnP implementation using the Siemens Java SDK for UPnP technologies. In the first case we had two interaction network, with 7 and 10 nodes respectively (see Figs. 2 and Fig. 5) and randomly generated rules of interaction, together with random perturbations emulating the user interaction with the system. The systems showed instability (see Fig. 3 and Fig. 6); however, when the locking mechanism is implemented (in the first case locking node 4, and in the second case locking nodes 4 and 8, all of them minimizing the impact on the network), the instability is satisfactory removed (see Fig. 4 and Fig. 7). In the UPnP implementation, we had an interaction network consisting of 4 nodes forming a single loop in which the user was able to interact with the system (turning on/off the lights). Without the locking cyclic instability can be observed (Fig 10), showing different modes of oscillation. However, the locking mechanism effectively stopped the cyclic behaviour (see Fig. 11).The same experiments were carried out successfully with a system with 11 devices (see Fig. 13). These results are encouraging, even with the preliminary UPnP implementation, as the computer simulations have shown the locking to be effective on larger and complex topologies as the one shown in Fig. 5, with two overlapping loops (sharing node 5).

## 5 Conclusions and Future Work

In this paper we have described a challenge to achieving the vision for ambient intelligence; how to overcome cyclic instability in coordinating multi agent systems. As pervasive computing paradigms, such as ambient intelligence, utilise systems of interdependent agents, we contend that such behaviour represents a significant obstacle to the commercial exploitation of this technology.

In a bid to address this challenge we have devised a formal framework for describing the problem (Interaction Networks) and offer a methodology for

overcoming the problem based on locking nodes in a pervasive computing network. We have used both simulation and real devices to show the effectiveness of our methods. Simulation gives us great flexibility allowing us to experiment with arbitrary structures and sizes of networks (eg showing that the approach is scalable) whilst the experiments with actual devices has allowed us to see the effects of network and processing delays, together with user interaction. User interaction plays a fundamental role, which was not easy to see in computer simulations. For example, when the system has reached a stable state and the user interacts with it, it is possible to see the changes suffered by the other devices, and when the locking mechanism is activated, how these changes are stopped, as the device locked prevents the propagation of the changes. On the other hand, the inclusion of sensors (light, movement, pressure, temperature, etc) will increase the complexity of the topology of the Interaction Network (but not the dynamic properties of the system!), as they cannot be part of a loop (they could only be fathers in the digraph), because its state depends on environmental conditions or user behaviour

Using these approaches we have shown that the locking mechanism is effective in the elimination of the unwanted cyclic behaviour, although the cost on the overall system is some temporary loss of functionality

As a future work we are planning to test our strategy with larger more complex topologies (in particular with multiple coupled loops) and with more complex rules. Also, as locking a node will impair, temporarily, some functionality of the system, the choice of what to lock and how long to lock (where there are options) is of some significance to the user. Thus a next step in our work is to experiment with a user based "locking preference" system learning. For this we will run experiments in our test bed (iDorm[1] – a full size apartment that is fitted with pervasive computing technology and agents) in order to provide additional evidence of the strategy, and to refine the locking mechanism with information of the user's preferences.

**Acknowledgments.** Victor Zamudio would like to acknowledge the support of the National Mexican Council for Science and Technology (CONACyT).

## References

1. V. Callaghan, M. Colley, H. Hagras, J. Chin, F. Doctor, G. Clark. "Programming iSpaces: A Tale of Two Paradigms", in iSpaces. Springer Verlag, 2005, Chapter 24.
2. J. Chin, V. Callaghan, G. Clarke. "An End-User Programming Paradigm for Pervasive Computing Applications", International Conference on Pervasive Services, 26-29 June 2006, Lyon, France.
3. Hagras,H.A.K., Callaghan,V., Colley,M.J., Clarke, G.S., Pounds-Cornish,A., Duman,H., 'A Fuzzy Logic Embedded-Agent Approach to Ambient Intelligence in Pervasive Computing Environments', IEEE on Intelligent Systems, 2004.
4. Estrin, D. Culler, D. Pister, K. Sukhatme, G. Connecting the physical world with pervasive networks. Pervasive Computing, IEEE. Jan-March 2002, Volume: 1, Issue: 1, pages: 59-69.
5. Weisbuch G. Complex Systems. Lecture Notes Volume II. Santa Fe Institute Studies In the Sciences of Complexity. 1991.
6. Discrete Mathematics for Computer Science. G. Haggard, J. Schlipf and S. Whitesides. Thomson 2006
7. Wolfram S. The Mathematica Book, 5th ed. Wolfram Media, 2003.

1 http://iieg.essex.ac.uk/idorm

8. Pemmaraju S, Skiena S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica™. Cambridge University Press 2003.
9. http://www.plug-n-play-technologies.com/