

# Understanding and Avoiding Interaction Based Instability in Pervasive Computing Environments

Victor Zamudio and Victor Callaghan

*Department of Computing and Electronic Systems, University of Essex,  
Wivenhoe Park, Colchester CO4 3SQ  
United Kingdom*

*Email: {vmzamu, vic}@essex.ac.uk*

*http://iieg.essex.ac.uk*

This paper addresses a fundamental problem related to the interaction of rule-based autonomous agents in pervasive and intelligent environments. Some rules of behaviour can lead a multi-agent system to display unwanted periodic behaviour, such as networked appliances cycling on and off. We present a framework called *Interaction Networks* (IN) as a tool to describe and analyse this phenomena. In support of this, and as an aid to the visualisation and understanding of the temporal evolution of agent states, we offer a graphical *Multidimensional Model* (MDM). We describe an *Instability Prevention System* (INPRES) based in identifying and locking network nodes. Both IN, MDM and INPRES enable system designers to identify and prevent such unwanted behaviour. Finally we introduce an *Interaction Benchmark* (IB) that we use to evaluate, experimentally, the effectiveness of our approach using both simulated and physical implementations.

## 1. INTRODUCTION

One vision for the future is that people will inhabit high-tech environments in which services such as heating, lighting, entertainment, security etc are derived from a host of interacting networked embedded-computers, the so-called digital home, intelligent building or smart office (to use but some terms). This is a natural extension of the Internet in which appliances such as Mp3 players, TVs, telephones, lights, heaters, etc. are all networked together and programmed, either manually [1] or automatically [2], to meet the user's desires and needs [2,3]. By sensing user actions and coordinating actions, environments can be orchestrated to create an ambient holistic intelligence. For instance, an incoming video conference call could be programmed to pause a DVD player, raise the lights (if it was dark) and patch the call through to the video screen (previously used for replaying the DVD). Clearly such environments form a complex social interaction of people and agents resulting in complex behaviour [4]. However, without careful design, such convenience is susceptible to unwanted disruptive behaviour can arise. This is the case for example, of continuous loop oscillating between cooling and heating [5].

This problem is rooted in the presence of interacting rules in the connected devices. Such rules are fundamental to most ambient intelligence or interacting pervasive computing systems. Also, temporal delays related to network latency, speeds of processing, etc. can result in some devices receiving old information, contributing to unstable behaviour [6]. This problem has been observed in our own systems (EU eGadgets project [7]), and is being observed increasingly in pervasive computing systems with distributed control [8].

In this paper we describe this phenomenon in more detail and present a theoretical framework, an *Interaction Network* (IN), which captures the functional dependencies of the rules of behaviour in pervasive computing systems, and represents them as a digraph. A *Multidimensional Model* (MDM) for task representation is introduced, which enables reasoning about the devices composing the environment, in terms of their local state and temporal evolution. We present *INPRES* (*Instability Prevention System*), a set of algorithms to prevent unwanted cyclic behaviour. To evaluate our methods we have devised an *Interaction Benchmark* (IB) that we use to evaluate, experimentally, the effectiveness of our approach using both simulated and physical implementations based in our digital-home testbed, the iDorm. Finally, we evaluate a hybrid strategy comprising INPRES together with a user-driven selective node disabling mechanism, which we term *Intelligent Locking*.

## 2. RELATED WORK

In home automation, instability due to cyclic behaviour has been reported in the EU Project CUSTODIAN (Conceptualisation for User involvement in Specification and Tools Offering the Delivery of system Integration Around home Networks). This project was funded through the European Commission's Telematics for Improving Employment and Quality of Life Sector. Its central objective was to *enable access to technology and services for disabled and elderly people and use information and communication technologies to improve the quality, effectiveness and efficiency of services which support the independent living and integration in society of disabled and elderly people* [9-10].

In this project, it is possible to specify the functionality of

each smart device, giving the logical condition that must be TRUE for the device to be activated. A simulation module propagated any change that occurred in the network until it settled down, and each smart device modified its status according to its rules. When a device changed its status, the simulation module was notified by means of a message, and the process continued until all devices had settled down to their final status. In this work they noticed that under some circumstances the network didn't stabilize due to a user-programmed "livelock", requiring them to terminate the process so that the network could be manually debugged [11].

Another example of cyclic instability in Ambient Intelligence is found in heating and air conditioning services. In this case, on reaching certain temperature, an air conditioning is turned on, causing the temperature to drop below a certain level and triggering a heating service, resulting in periodic behaviour [12-14].

In other applications software agents may be involved in cyclic loops, for example in email mailing list, where users have configured auto-replies that answer each other [15].

Also, in communications and distributed asynchronous systems, a similar problem relating to the assignment of resources, known as a deadlock, has been encountered [16]. A deadlock state occurs when two or more processes are waiting indefinitely for an event that can only be completed by one of the waiting processes. In order to explain this more clearly, the concept of a WAITFOR graph is useful.

More formally, a WAITFOR graph for the allocation status of a set of resources is a directed bipartite graph  $D_{V_1, V_2, E}$  where  $V_1$  and  $V_2$  are a bipartition of  $V$ . The elements of  $V_1 = \{U_1, U_2, \dots, U_n\}$  represent users of the resources, and the elements of  $V_2 = \{R_1, R_2, \dots, R_m\}$  represent resources. For  $P \in V_1$  and  $R \in V_2$  there is an edge  $(P, R) \in E$  if the user  $P$  requested resource  $R$  and has not been granted resource  $R$ , and an edge  $(R, P) \in E$  if user  $P$  has been allocated the resource  $R$ .

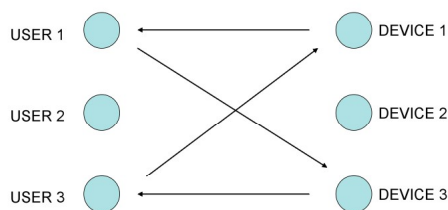


FIG. 1: Example of a deadlock in a WAITFOR graph: USER1-DEVICE 3-USER3-DEVICE 1-USER1

If the WAITFOR graph contains a directed cycle, then there is a deadlock in the system. The simplest case is when two

transactions are waiting, and each is dependent on the other. In Figure 1 we can see a deadlock in a WAITFOR graph, involving the loop USER1-DEVICE 3-USER3-DEVICE 1-USER1.

As mentioned previously, a deadlock may be detected by finding cycles in the WAITFOR graph, and a transaction could be selected for abortion to break the cycle. If the transactions have different priorities, the transaction with the lowest priority is aborted.

Deadlock can be viewed as being the opposite problem to the cyclic instability addressed by this paper as, in the case of a deadlock, every device in the loop is static, but in cyclic instability every device or agent in the loops changes indefinitely. However, the idea of "breaking the cycle" is very useful, and we apply this principle as part of the Interaction Networks framework which we describe in the next section to find the cycles associated with rules of interaction between agents. In addition, the ability to associate different priorities to transactions is a useful concept which we use this to capture the preferences of a user, thereby allowing the least inconvenient transaction (programmed action) to the user to be disabled.

Research into 'system dynamics' has advanced the understanding of the dynamics of massively interconnected systems. From this work what is termed an attractor of a dynamic system is an infinite loop in the state space (the same sequences of states are visited each time). The *basin of attraction* is defined as the set of configurations which converge toward an attractor. However, although a dynamic system can be solved theoretically (ie, finding the attractors and the basin) an important result is that it is not possible to solve this for a general boolean network a problem that matches the type of pervasive computing environment we are addressing [17]. An additional problem is that perturbations to the system (when the user interacts with the environment) add significant complexity to the dynamics. It is useful to observe that the complexity of the periodic behaviour problem addressed by this paper is rooted in the rules of interaction between the devices as the time delays can be regarded as being equivalent to a new initial condition to the system.

Even though system dynamics work has shown it is not possible to solve an arbitrary system, it is possible to detect and prevent cyclic instability. We have developed a framework called Interaction Networks, which enables the interrelationships between the agents due to the rules to be visualised, and a mechanism to prevent cyclic behaviour called INPRES (Instability Prevention System) to be applied.

### 3. THE PROBLEM

In the previous section we reviewed several problems related to our research. From Complex Dynamics Systems, we found that it is not possible to theoretically analyse a Boolean network to isolate cases of instability. In addition, initial

conditions play a very important role on the dynamics of a system. From the domain of Distributed Systems, we found that a deadlock state occurs when two or more processes are waiting indefinitely for an event that can be completed only by one of the waiting processes. This could be prevented by aborting one of the transactions on the loop. The problem addressed in this paper is the other way around: two or more processes are running indefinitely.

In summary, as explained earlier, this paper focuses on the behaviour of autonomous interacting devices (eg rule based agents). Such devices can interact with each other according to rules provided by, in general, several users. Besides that, there can be some delays in the propagation of information between devices, due to different speeds of processing in each device, differences in computational load or because of differing network paths. In situations where the state of one device is dependent on that of a second device, and vice versa, there is the potential for a *closed loop and for oscillation*. If oscillation is not wanted, then this is a problem that needs to be identified and eliminated, which is the motivation driving this research.

The objective of this research may be stated as providing the means to identify and stop unwanted periodic behaviour in pervasive autonomous environments. In order to stop this behaviour, we propose the use of an Intelligent Locking strategy which:

- a) Detects loops in the Interaction Network
- b) Locks a variable in a candidate loop
- c) Learns, from the user based on his response to locking actions, the locking priorities, and acts on these (stored) priorities whenever a variable needs to be locked in the future.

This process of detecting loops and locking variables raises several issues: How to choose the variable to be locked, if there are several variables in the loop? What happens if there are coupled loops (ie more than one interacting loop)? When the variable should be locked; immediately, after one oscillation (or more) and for how long? What impact could this locking have in the original dynamics of the system? For example would it inhibit critical behaviour or prevent vital information being propagated throughout the system? In this paper we present results from both coupled and uncoupled systems. In order to have the least impact on the network, the node with least connectivity is locked.

As described earlier, the strategy to eliminate instability is based on detecting cycles: either isolated cycles (with potentially independent oscillations) or coupled cycles (potentially, with very complex oscillations). In terms of the functional dependencies, the behaviour of a device could depend on a single or several devices. Figure 2 presents a taxonomy of the problem, which reflects the topological and functional aspects of the agent instability problem.

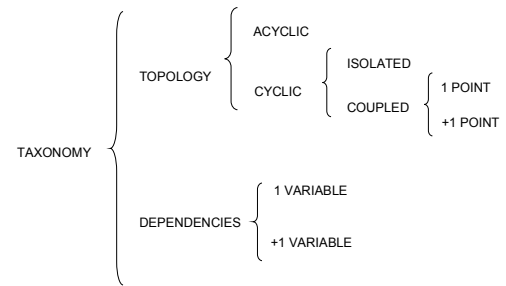


FIG. 2: Taxonomy of the inter-agent instability problem.

As will be described later, the user may interact with the agents which could alter the behaviour of the system, leading it new periodic behaviours (ie a new initial state), which may be unwanted and need to be stopped.

In order to study this problem, we first introduce some definitions and use these to describe the principles of an interaction network. In the following description we use the term agent as meaning any autonomous interacting rule-based control device.

#### 4. THEORETICAL FRAMEWORK

In order to solve the problem of instabilities, we have developed a graph-based formalism, the Interaction Networks (IN). This Interaction Networks let us reason about multiagent system, showing the mutual interdependencies of the rules of behaviour of them.

##### 4.1 Interaction Networks

A *directed graph*  $G$  (also known as *digraph*) 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  $v$ , i.e.  $(v, w) \in E$ , then  $w$  is adjacent to  $v$  [18].

An agent  $A$  is an autonomous device with a state  $s \in \{0, 1\}$ , where 0 and 1 mean off and on respectively. If we have  $n$  autonomous devices agents  $A_1, A_2, \dots, A_n$ , the state of the system is  $S = (s_1, s_2, \dots, s_n)$ . Each agent  $A_i$  has two rules:

- If  $\phi_i$  then  $s_i = 1$  (1)
- If  $\psi_i$  then  $s_i = 0$  (2)

where

$$\phi_i = \bigwedge_{j \in \mathcal{I}_i} s_j \quad \psi_i = \bigvee_{j \in \mathcal{O}_i} s_j \quad (3)$$

An *Interaction Network (IN)* is a digraph  $(V, E)$  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 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  $U_1 \rightarrow U_2 \rightarrow \dots \rightarrow U_k$ . 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.

Figure 3 provides an example of an Interaction Network, showing the dependencies of 5 devices or services: Sofa Sensor, Light Sensor, MP3 Player, Light, and Word. The *light* depends on the state of the *light sensor*, and in the state of the *MP3 player*. The software application *word* depends on the state of the *light*, and in the occupancy of the sofa (*sofa sensor*).

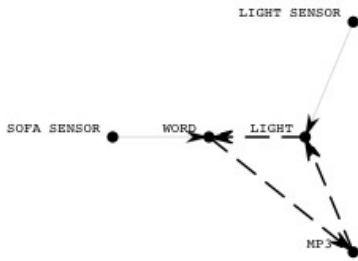


FIG. 3: Example of the Interaction Network showing the dependencies of 5 devices in pervasive computing environments.

#### 4.2 Multidimensional Model for Task Representation

A *temporal allocation* is a tuple  $(d, T, t_i, t_f)$ , where  $d$  is a simple device,  $T$  is a simple task,  $t_i$  is the initial time and  $t_f$  is the final time. In other words, the device  $d$  will be performing the task  $T$  during  $t_f - t_i$  units of time, beginning on the instant  $t_i$ .

So, a temporal community, denoted by  $C_t$ , is a non-empty set of temporal allocations:

$$C_t = \bigcup_{j=1}^k \{(d_j, T_j, t_{ji}, t_{jf})\} \quad (4)$$

This definition allows us to locate the devices in a 3-axes graph: device, task (or state) and time [19].

This model is used in section 5.2 in order to show, in detail, the dynamics of the systems tested in the iDorm.

#### 4.3 Instability Prevention System (INPRES) and Intelligent Locking

Our strategy (Patent No: GB 0624827.2) is based on finding all the simple cycles in the digraph, and for each cycle, lock the node with minimum functionality:

```
Cycles C =extractCycles(Graph g);
for each cycle c in C:
    list.empty(); //initialize list
    for each node n in c:
        f=functionality(g,n,c);
        list.add((n,f));
    od;
    nodeMin=list.min();
    nodeMin.lock();
od;
```

The strategy has several auxiliary functions. The first is *functionality*. This function receives as arguments the graph  $g$ , the node  $n$  and the corresponding cycle  $c$ . It temporally deletes all the nodes of the cycle  $c$  in  $g$ , returning the number of total descendants of the node  $n$  in the graph, using ‘depth-first’ traversal:

```
int functionality(Graph g, node n, cycle c):
    g'=g-c+n;
    return DepthFirst(g',n).Length();
```

The function *extractCycles* prunes all the root nodes in the graph (as they cannot be part of a cycle), storing all the remaining nodes in *list*. For each node  $n$  in *list*, a search for cycles is performed by the function *analyze*: it receives the node  $n$  as argument, and maintains a list for each branch of a ‘breadth-first’ traversal. If the node  $n$  is found, a cycle has been found and it will not be expanded further. If a repeated node is found, the list containing it is pruned. The process terminates when it is not possible to expand the list of branches further, and the function *analyze* returns the list of all cycles, which will be stored in the list *cycles*. When all the nodes in *list* have been searched by the function *analyze*, repeated cycles are removed and the remaining list of cycles is returned:

```
Cycles extractCycles(Graph g):
    list = prunedList(g);
    for each node n in list:
        c = analyze(g,n);
        cycles.add(c);
    od;
    return cycles.removeRepeatedCycles();
```

It is important to notice that, as we are interested on finding the members of a cycle, permutations of the members of a cycle are considered indistinguishable (ie we are considering the cycles as sets of nodes).

5 IMPLEMENTATION AND RESULTS

5.1 Computer Simulations

The experiments used 64 interacting agents. These agents were allocated in a grid of  $8 \times 8$  (see Figure. 4). The experiments were implemented using *Mathematica*® 6 [20], a programming language with powerful tools for quick prototyping. In particular, the package *Combinatorica* [21] was extremely useful, as it provided tools for graph theory, graphics and combinatorics.

Using the simulator it was possible to control a number of parameters, such as the number of agents involved, probability of perturbations, generation of random topologies and random rules of interaction, amongst others. Boolean functions were assigned randomly, as rule of behaviour, to each device represented as a binary string, where 0 and 1 would be interpreted as an OR and AND gates respectively.

The experiments used the following topologies:

- a) non-coupled systems,
- b) coupled in 1 point
- c) coupled in two points
- d) random systems.

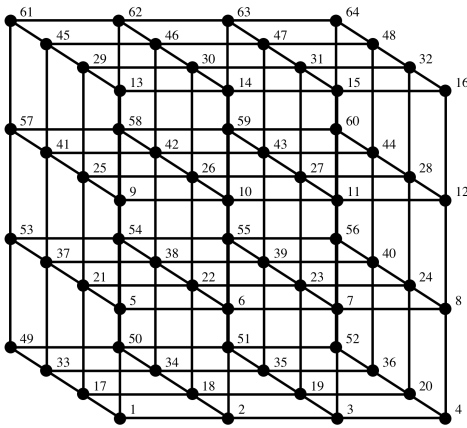


FIG. 4: 64 agents allocated in the nodes of a grid.

5.1.1 Non Coupled Systems

For the case of non coupled systems, 64 agents were distributed over a grid of  $8 \times 8$  (see Fig 5.) with 16 cycles of length 4 being used.

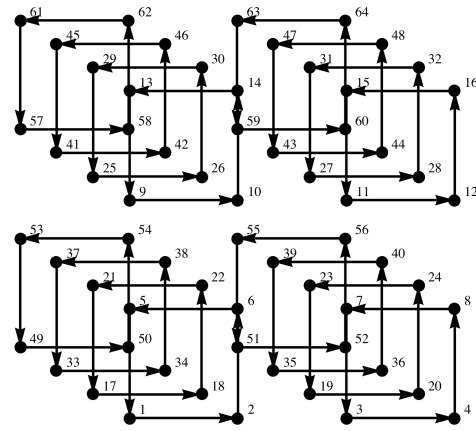


FIG. 5: Grid with 64 nodes, and 16 uncoupled cycles.

The rules of interaction were set randomly:  $\{0,1,1,0,0,0,1,1,0, 0,0,1,1,1,1,0,1,1,0,0,0,1,0,1,0,1,0,1,0,0,0,0,1,1,1,1,1,0,1,1, 1,1,1,0,0,0,1,0,1,0,0,1,0,1,0,1,0,0,0,1\}$ . The initial condition was set randomly also. Under these conditions, the system oscillated as shown in Figure 6. As we can see, the perturbations take the system from one mode of oscillation to another.

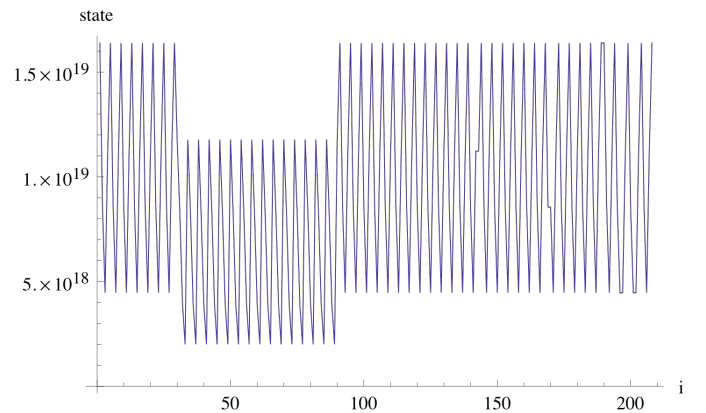


FIG. 6: Oscillations for the case of a system with 16 uncoupled cycles. No locking was applied.

When the locking mechanism was applied, the instability was removed, as shown in Figure 7. 16 nodes have been locked, and the locking vector of the system was:  $\{0,1,0,1,1,1, 1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1,0,1,1,1,1, 1,0,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1\}$ .

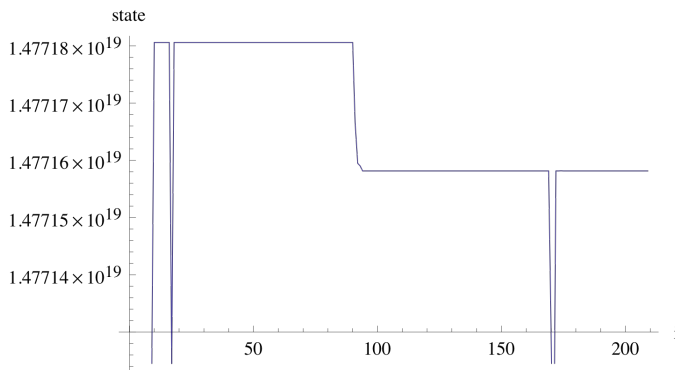


FIG. 7: Systems with 16 cycles. The systems has been locked and the oscillations have stopped.

### 5.1.2 Coupled Systems: one point

For the case of coupled systems ‘in one point’, we used the topology defined in Figure 8. In order to create this topology, we added a cycle to the middle of the corresponding plane of the topology shown in Figure 7. The rules of interaction were  $\{1,0,1,0,1,1,1,0,0,1,1,1,1,0,1,0,1,1,1,0,0,1,1,0,0,0,1,0,1,1,0,1,1,1,0,0,1,1,1,1,0,1,0,1,1,1,1,0,0,0,0\}$ .

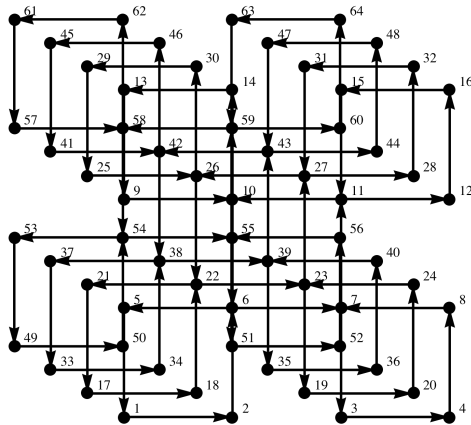


FIG. 8: Grid with 64 nodes, and 20 cycles coupled in one node.

Without locking, the system oscillated as shown in Figure 9. The discontinuities in Figure 9 were not related to the dynamics of the system, but to the scale of the graph, as we were focusing on the oscillations.

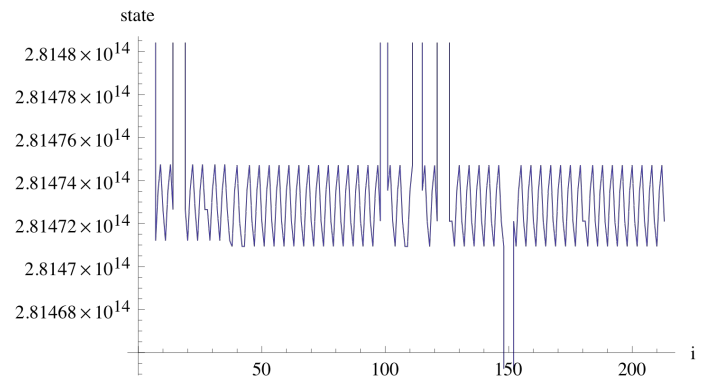


FIG. 9: Oscillations for the case of 20 cycles coupled ‘in one point’.

When the locking was applied, the oscillations were prevented. In this case 20 nodes were locked, and the locking vector was:

$\{0,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1,0,1,0,1,1,0,1,1,0,1,1,1,1,0,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1,1,1,0,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1\}$ .

Figure 10 shows the evolution of the system without oscillations. There were some spikes due to the random perturbations (emulating a user interacting with the system), but after the user perturbations (ie spikes) the system stabilised again.

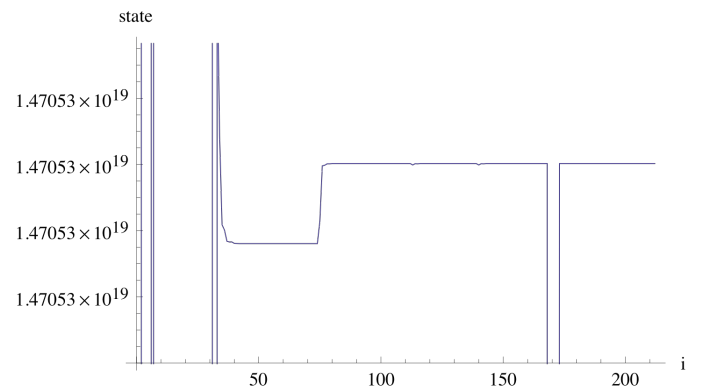


FIG. 10: System with 20 cycles coupled in one point. The system is stable after locking.

### 5.1.3 Coupled Systems: two points

In order to have coupled systems ‘in two points’, we grew the digraph with 20 cycles coupled in one node shown in Figure 8. For each plane 4 edges were added as shown in Figure 11. With this, each plane has 10 cycles, and 40 cycles in total. Central cycles share one or two nodes with each of the 8 remaining cycles.



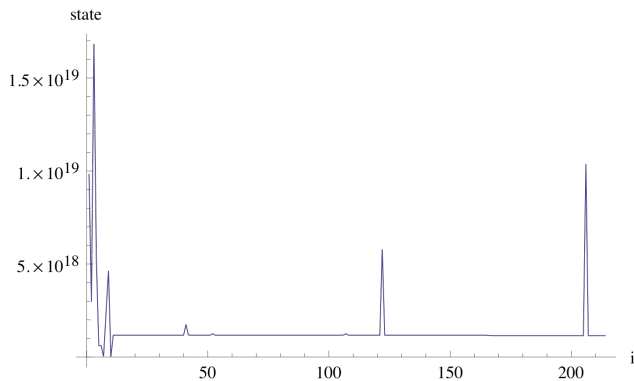


FIG. 16: Response of the system with 64 nodes and 81 cycles. When the locking is applied, the system was stable.

## 5.2 Implementation in the iDorm

### 5.2.1 Experimental Setup

The iDorm is a multipurpose space, taking the form of a domestic apartment, with areas for varied activities such as sleeping, working and entertaining. It is based around three wired networks, Lonworks, 1-wire (TINI) and IP plus two wireless networks; WiFi & Bluetooth. Universal Plug and Play (UPnP) is used as the common interface to the iDorm, enabling automatic discovery and configuration. Our system was built on top of the low level UPnP control architecture, enabling it to communicate with the UPnP devices and orchestrate their action in the iDorm [22].

### 5.2.2 Experiments: Automatic Locking using INPRES

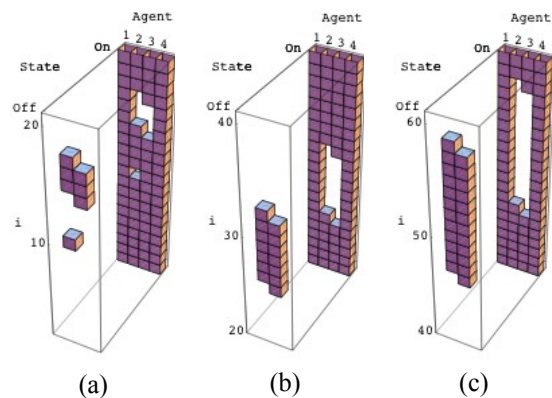
In order to test our approach with a real intelligent environment, we used a system with 4 devices. The topology of the Interaction Network (IN), in terms of the adjacency list for this 4 devices is  $\{\{1,2\},\{2,3\},\{3,2\},\{4,3\}\}$ , where devices 1 and 4 are software-based UPnP lights. Device 2 is the bed light, and device 3 is the desk lamp. The system has one cycle including the bed light and the desk lamp. In Figure 17 we can see the system's Interaction Network. In these experiments we utilized the Multidimensional Model (MDM) to represent and visualize the local state of the devices in the environment.



FIG. 17: Topology of the Interaction Network (IN) for 4 agents. Agents 2 and 3 are lamps, and agents 1 and 4 are software-based lamps. There is a cycle involving lamps 2 and 3.

The rules of interaction are encoded as  $\{0,1,1,0\}$ , where 1 is an AND gate, and 0 is an OR gate. The bed and desk lamp were allocated an AND gate, whilst agents 1 and 4 an OR gate. As in the simulations, if there was only one argument for the Boolean functions AND and OR, they behave as identical functions:  $AND(x) = OR(x) = x$ .

In this experiment, the user interacted with the system. If there was no locking, the system was unstable. In Figure 18 the evolution of the system was presented using the MDM. The agents involved in the oscillations were agents 2 and 3. Figure 19 shows the evolution of the system, using a decimal representation of the states of the system, corresponding to the MDM in Fig. 18. This shows the oscillations of the system, together with the delays due to the different velocities of processing, network delays, etc.





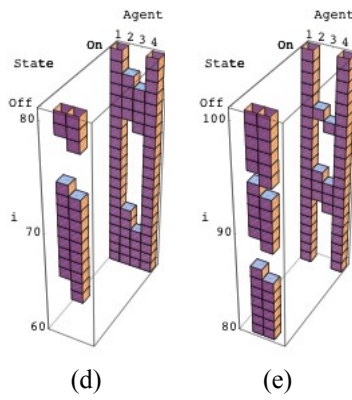


FIG. 18: Evolution of the system using the MDM, and showing instability.

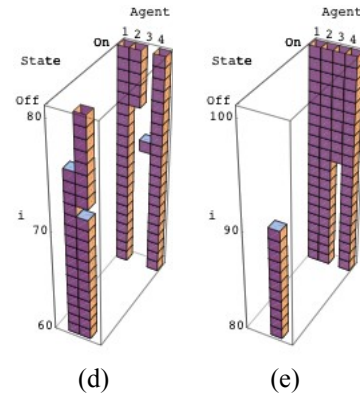
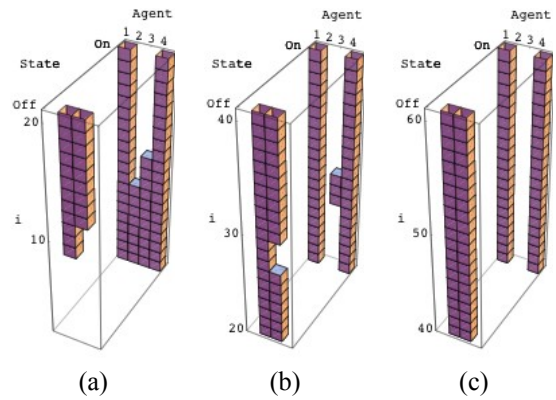


FIG. 20: MDM of the system with agent 2 locked

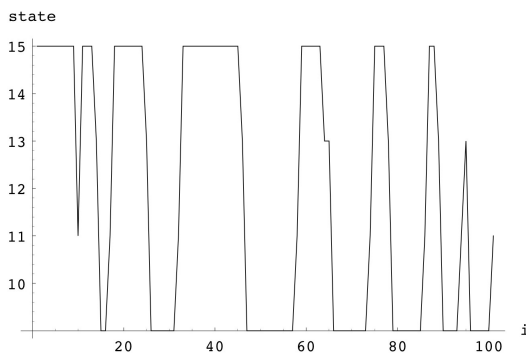


FIG. 19: Evolution of the system without locking, showing delayed oscillatory behaviour.

When the locking mechanism is activated, agent 2 (bed lamp) had been locked. The only way agent 2 could change its state was when the user turned it on/off. Figure 20 (a) shows the initial state of the system (1,1,1,1) followed by the user turning device 2 off. After some delay, due to the rules, agent 3 is in an off state, and the system is stable. In Figure 20 (b) the user turns on agent 3, but because of the rules and the locking of agent 2, the system goes back to the state (1,0,0,1). In Figure 20 (d) the user turns on agent 2 and, after a delay, the system goes to state (1,1,1,1). This is consistent with the representation in Figure 21.

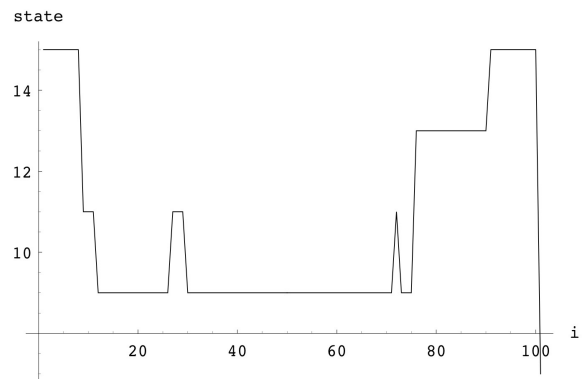


FIG. 21: Evolution of the system when agent 2 is locked. The spikes represent the user interaction with the system, which rapidly goes back to the previous state due to the locking. The flat spike represents a delay on the response of the system (see Fig. 20 b).

Figure 22 and 23 shows another example, when the system is locked in agent 2. In Fig. 22 (b) agent 2 is turned on, when agent 1 and 3 are off; due to the rules, agent 2 should be off, however, because it is locked, it remains on.

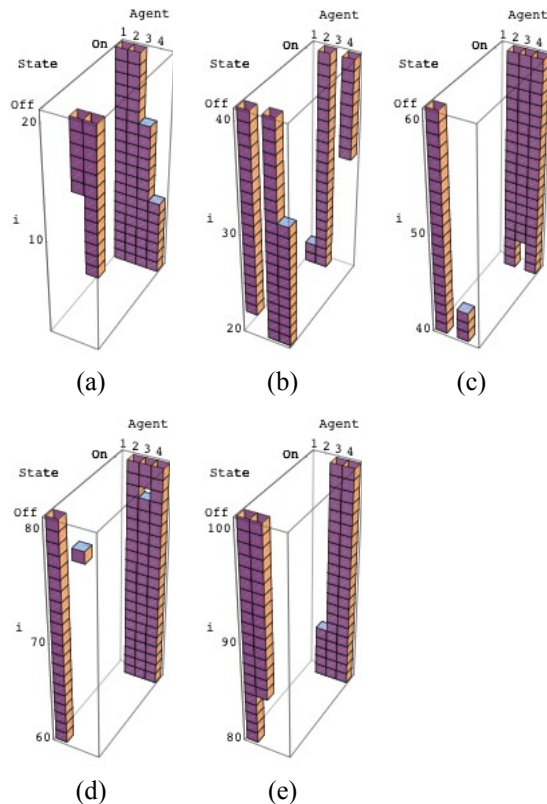


FIG. 22: MDM of the system when agent 2 is locked. The system is stable.

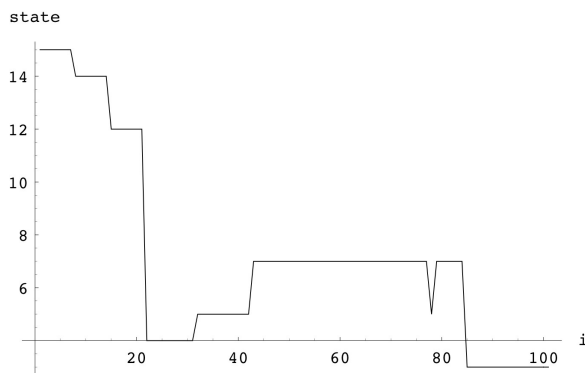


FIG. 23: Response of the system when agent 2 is locked.

### 5.2.3 Hybrid Solution: Automatic Locking using INPRES and User-based locking

As was shown, a member of a loop needs to be locked in order to have a stable system. Under some circumstances, the user may want a specific device on; however, as we have shown, if that device is not locked, the system could automatically turn it off. This is the case of Figure 20 (b) where the user turns on agent 3, but because of the rules, and the locking of agent 2, the system automatically turns off agent 3.

In order to prevent this situation arising, it is possible to refine our approach with a hybrid solution: In first instance the strategy defined by INPRES is used in order to prevent instabilities (locking agent  $n$ ); once the system is stable, the device the user has just interacted with is locked (if that device is part of a cycle):

```

automaticLocking();//node n is locked
if the user interacts with agent n':
    if agent n' is part of a Cycle :
        lock(agent n');
        unlock(agent n);
    fi;
fi;

```

In order to capture the user interaction with the system, the user is provided with an 8 button controller (on/off for each device - see Fig. 24).

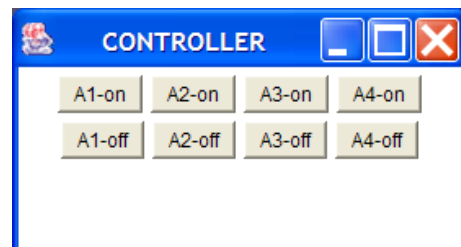


FIG. 24: Controller for user interaction.

Figure 25 and 26 shows the evolution of the system using the hybrid strategy. Agent 3 is locked automatically, being in state 'off'. The user then turns on agent 3, and agent 3 remains locked (Fig. 25 a). In Fig. 25 (b) the user turns off agent 4, but it is not locked as it was not part of the loop. At around iteration 30, agent 1 is turned off, and agent 2 is automatically turned off; agent 3 remains on because it is locked. In Fig. 25 (c) agent 2 is turned on and is locked, preventing any change (without the locking it would automatically be turned off); agent 3 is not locked anymore, and is automatically turned off.

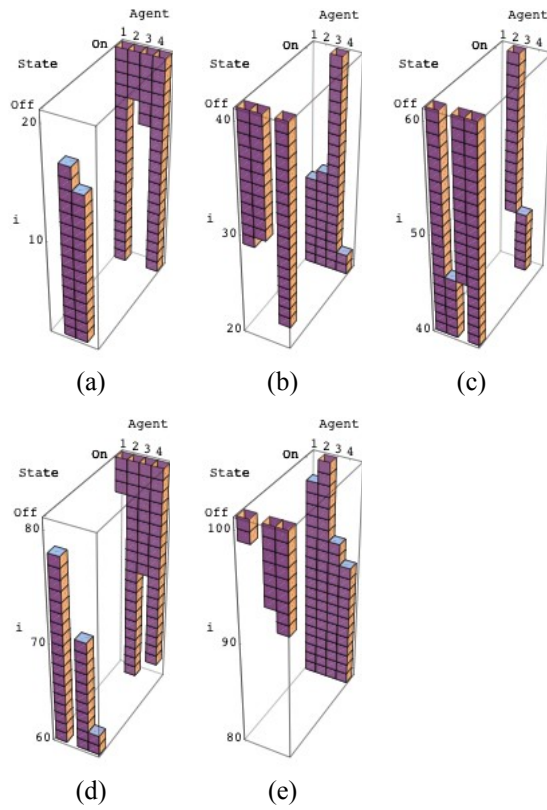


FIG. 25: MDM of the system showing the user based locking.

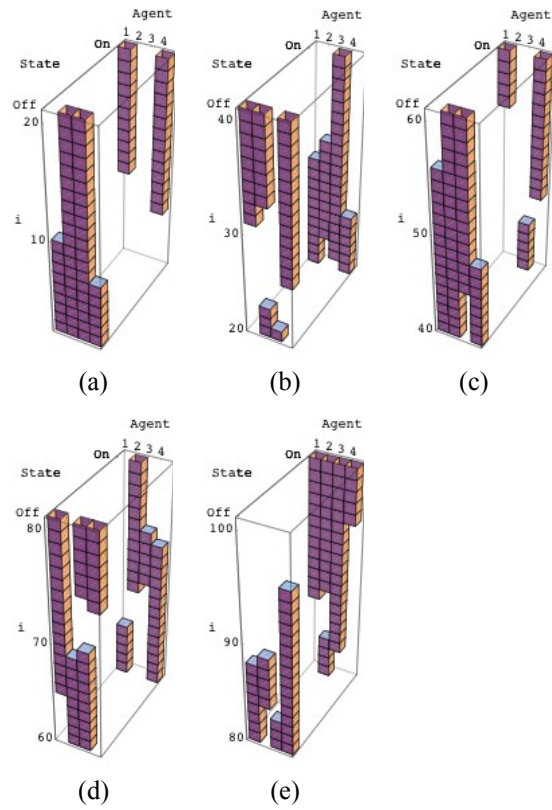


FIG. 27: MDM of the system.

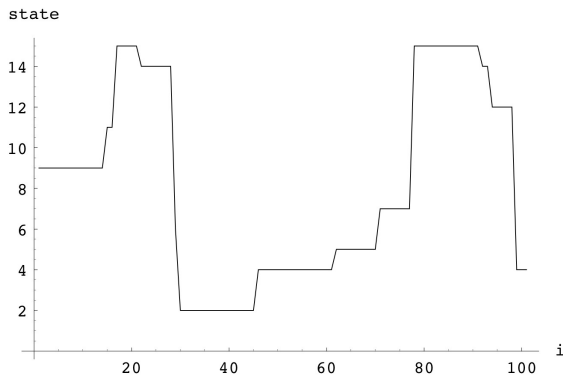


FIG. 26: Evolution of the system, showing stability due to the user-based locking.

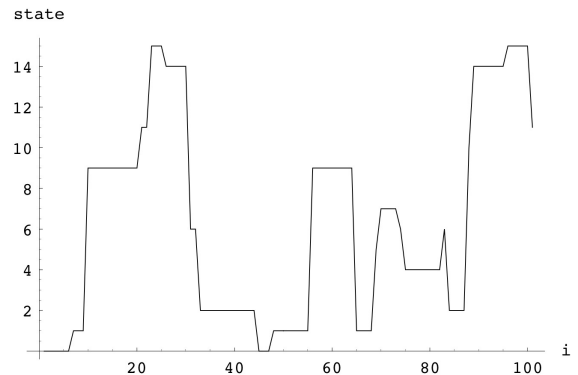



FIG. 28: Evolution of the system.

### 5.3 Results Discussion

Figure 27 and 28 shows another example of the strategy. In this case agent 3 is locked automatically. In Figure 27 (b) all the agents 1, 2, and 4 are off, while agent 3 (locked) is on. In Figure 27 (d) the user turns on agent 2 (now locked), preventing any automatic change.

We have implemented and tested a strategy to prevent cyclic behaviour using both computer simulations and real networked devices. The simulator was programmed in Mathematica® 6, using 64 agents distributed over a grid of  (see Fig. 4). Consistent with our taxonomy, we used 4 topologies: non-coupled cycles, coupled cycles (1 point), coupled cycles (multiple point), and finally a randomly

generated system. Our algorithm to find cycles correctly identified 16, 20, 40 and 81 simple cycles respectively. In order to visualize the evolution of the system over time, we used the decimal representation of the binary state of the system.

As part of our experimental procedures we first calibrated the systems in order to have a set of rules that could induce cyclic behaviour. These rules were generated randomly and for each case we showed the instabilities. In the cases of non coupled system and the system coupled in 2 points, the random perturbations (playing the role of a user interacting with the environment) drive the system to show different modes of oscillations. In all the cases our INPRES system proved able to prevent instabilities satisfactorily.

In the case of systems with 40 and 81 cycles, INPRES locked 28 and 9 agents respectively. As our algorithm was based on locking one agent in each cycle, clearly INPRES chose the same agent to be locked for multiple cycles.

In order to test our strategy with real networked devices, we used system with 4 agents (see Fig. 17). As mentioned previously, time delays can play an important role in the instabilities [6]. For instance, in Figure 19 we have oscillations together with delays. In all the cases INPRES removed satisfactorily all the instabilities, as these delays can be interpreted as equivalent to perturbations generating new initial conditions to the system.

The Multidimensional Model MDM proved to be a very useful tool to analyze locally the evolution of the system. In particular, it helped to interpret the spikes present in the iteration-state graph of the evolution of the system.

We also used a hybrid strategy combining automatic and user based locking. We used INPRES to automatically lock the system and, after that, we adapted the locking using user interaction. If a device was directly interacted with by the user, we locked that device if it was part of a cycle. This approach proved more satisfactory to the user as the system, due to the rules, could reverse the user's last interaction. This was the case shown in Figure 20 (b) where agent 2 is locked, and the user turns on agent 3, however due to the rules and the locking of agent 2 the system automatically turns agent 3 off.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper we have addressed the problem of cyclic instability that occurs in pervasive computing systems composed of multiple distributed interacting pervasive computing devices (rule based agents). Earlier systems, such as smart homes, were based mainly on centralized control servers where such problems do not exist. The move to distributed models has exposed this issue which is rooted in

interacting rules and delays between interdependent devices. We have discussed how this issue relates to other engineering domains such as dynamic systems and distributed computing.

Dynamics systems are closely related to our research, because they show complex behaviour that depends on the interaction of the members of the system. In order to solve a dynamic system, it is necessary to find the attractors and the basin. The evolution of the system depends on the initial conditions, and in general, it is not possible to solve it theoretically. This is a very important result, because it shows that periodic behaviour depends on the rules of interaction between agents.

In the domain of distributed system, the problem of deadlocks has some similarities to our problem: devices and users are waiting for further interaction, but in our problem the agents are interacting indefinitely. A deadlock can be found if there is a loop in the WAITFOR graph, which can be solved aborting a process in the loop.

We have presented an Interaction Network, which is a mathematical model based on a directed graph that lets us reason about the rules and the interaction of the devices, showing the interdependencies of rules. These dependencies could be cyclic, leading our system to an oscillatory state. We have also presented our *Instability Prevention System* INPRES, which is a strategy based on finding the cycles in the IN associated, in order to lock a device with the minimum functionality, ie, the minimum impact on the network. We tested successfully this strategy using not only computer simulation but an implementation in the iDorm, our test bed.

Whilst it would be possible to use any network topology to evaluate our system it is preferable to have an arrangement that is both easy to reason about and for others to duplicate as part of verification or benchmarking results. Therefore we have proposed the use of a 3D grid to allocate the agents and allow the complexity of the tests be grown in an ordered and intuitive way. We refer to this as the *Interaction Benchmark* (IB). In the IB, the complexity of the topology can be controlled using 4 independent but identical layers of interconnected devices and by increasing the number of edges progressively for subsequent trials (which, as a consequence results in, additional cycles in each tested topology). This independence is evident to the observer, but not to the search and locking algorithms used by INPRES. We hope that this IB will, in itself, be a useful model for other researchers.

In our tests, for the IB, we have adopted a random topology and rule set based on a symmetric 4x4x4 array (ie 64 nodes or agents). Whilst some nodes were disconnected (for example, node 4 and 29 in figure 14), our algorithm found 81 cycles, with 9 nodes locked (to provide a stable system). These 9 nodes covered all the possible instability cycles, indicating there was a high degree of overlap between the cycles (otherwise more nodes would have needed to be locked).

It is important to note that cycles having exactly the same elements, disregarding their connectivity or order, are considered to be indistinguishable by INPRES, and do not need to be considered nor locked separately.

It is evident from examining the results presented that the algorithm defined by INPRES proved to be successful on removing oscillations in all the cases.

In the experiments performed in the iDorm we use the MDM, which proved to be a very useful for analyzing individual behaviour due to its expressiveness and simplicity. This model could be used in larger systems focusing on a small subset of the whole system, ie, in a convenient neighbourhood in the device-time-state space (see Figure 39).

Using the MDM, we realized that event though INPRES prevents successfully the cyclic behaviour, it has a drawback: due to the rules and the state of the system, the device that the user has just interacted with (changing its state), could be reversed. This can be possible because of several conditions: state of the system, rules, and the device locked. The strategy can be improved using a hybrid approach: using an automatic locking using INPRES, and after that adjust the locking using the user interaction. This hybrid approach was tested successfully in terms of the prevention of oscillations; however, more research is needed in this direction using more realistic scenarios.

In our experiments with coupled systems, we found out that sometimes it is not so easy to find instabilities, compared to systems with less coupling or less number of cycles. We postulate that high coupling and high number of cycles contributes dramatically to this self-locking; however, more research is needed. We look forward to reporting on our progress in this direction in future publications.

Finally, before this work there was no framework for analysing and eliminating problems of unwanted cyclic behaviour related to the interaction of rule-based autonomous agents in pervasive and intelligent environments. The *Multidimensional Model*, *Interaction Benchmark*, *Interaction Network Theory*, *Instability Prevention System* and intelligent locking techniques described in this paper offer a practical solution to this problem.

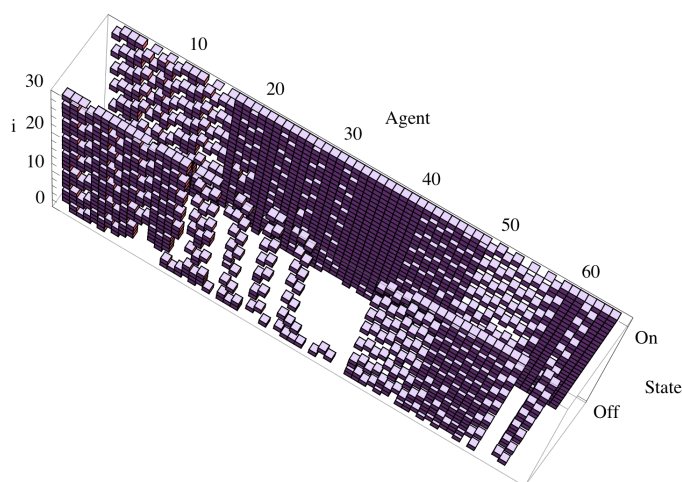


FIG. 29: This graph shows instability using the MDM for the case of 64 nodes. Focusing on a small part of the information sometimes could be useful in order to process and analyze the dynamics of the systems.

#### ACKNOWLEDGEMENTS

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

#### REFERENCES

- [1] 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.
- [2] V. Callaghan, M. Colley, H. Hagraas, J. Chin, F. Doctor, G. Clark. "Programming iSpaces: A Tale of Two Paradigms", in *iSpaces*. Springer Verlag, 2005, Chapter 24.
- [3] H. Hagraas, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish and H. Duman, Creating an ambient-intelligence environment using embedded agents. *IEEE on Intelligent Systems*, Volume 19, Issue 6, Nov-Dec 2004 Page(s):12 – 20
- [4] V. Callaghan, G. Clarke, J. Chin, "Some Socio-Technical Aspects Of Intelligent Buildings and Pervasive Computing Research", *Intelligent Buildings International Journal*, Vol 1 No 1, 2008
- [5] M. Wilson, E. Magill, M. Colberg. "An Online Approach for the Service Interaction Problem in Home Automation". *Consumer Communications and Networking Conference 2005 CCNC Second IEEE*, 3-6, Jan. 2005, pp. 251-256.
- [6] V. Zamudio and V. Callaghan. Unwanted Periodic Behaviour in Pervasive Computing Environments, *The IEEE International Conference on Pervasive Services*, Lyon, France, 26-29 June 2006
- [7] V. Callaghan, M. Colley, G. Clarke and H. Hagraas, *The Cognitive Disappearance of the Computer: Intelligent*

- Artefacts and Embedded Agents, Proceedings of the i3 2001, workshop WS4 on Cognitive Versus Physical Disappearance, Porto, Portugal, April 2001
- [8] D. Estrin, D. Culler, K. Pister and G. Sukhatme, Connecting the physical world with pervasive networks. IEEE on Pervasive Computing, Jan-March 2002, Volume: 1, Issue: 1, pages: 59-69
- [9] The CUSTODIAN Project [Online]. Available at <http://www.rgu.ac.uk/sss/research/page.cfm?pge=33336>
- [10] D. Paulson, C. Nicolle, M. Galley. Review of the current status of research on ‘Smart Homes’ and other domestic assistive technologies in support of TAHI trials-Prepared for The Department of Trade and Industry. Ergonomics and Safety Research Institute (ESRI), Loughborough University October 2002. Available on [http://dspace.lboro.ac.uk/dspace/bitstream/2134/1030/1/A\\_R2320.pdf](http://dspace.lboro.ac.uk/dspace/bitstream/2134/1030/1/A_R2320.pdf)
- [11] J. M. Martins Ferreira, T. Amaral, D. Santos, A. Agiannidis and M. Edge, The Custodian Tool: Simple Design of Home Automation Systems for People with Special Needs. Presented at the EIB Scientific Conference. Munich, October 2000.
- [12] M. Kolberg, E. Magill, D. Marples and S. Tsang, Feature interactions in services for Internet personal appliances. IEEE International Conference on Communications, 2002. ICC 2002. Volume 4, 28 April-2 May 2002 Page(s):2613 – 2618
- [13] M. Wilson, E. Magill and M. Colberg, An Online Approach for the Service Interaction Problem in Home Automation. Consumer Communications and Networking Conference 2005 CCNC Second IEEE, 3-6, Jan. 2005, pp. 251-256.
- [14] M. Wilson and E. Magill, A Model for Service Interaction Avoidance in Home Networks. In Proc. Of the 5<sup>th</sup> Annual Postgraduate Symposium on the Converge of Telecommunications, Networking and Broadcasting. Liverpool. June 2005.
- [15] M. Mowbray, M. Williamson. “Resilience for Autonomous Agents” Technical Report HPL-2003-210. Internet Systems and Storage Laboratory, HP Laboratories Bristol. October 17<sup>th</sup> 2003.
- [16] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems. Concepts and Design. 4<sup>th</sup> Ed. Addison Wesley. 2005.
- [17] G Weisbuch. Complex Systems. Lecture Notes Volume II. Santa Fe Institute Studies In the Sciences of Complexity. 1991.
- [18] G. Haggard, J. Schlipf and S. Whitesides. Discrete Mathematics for Computer Science. Thomson 2006.
- [19] V Zamudio, V Callaghan and J Chin, A Multi-Dimensional Model for Task Representation and Allocation in Intelligent Environments Proceedings of The Second International Symposium on Ubiquitous Intelligence and Smart Worlds (UISW2005). Nagasaki, Japan. December 6-7, 2005.
- [20] S. Wolfram. The Mathematica Book, 5<sup>th</sup> ed. Wolfram Media, 2003.
- [21] S. Pemmaraju and S. Skiena. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica™. Cambridge University Press 2003.
- [22] A. Holmes, H. Duman and A. Pounds-Cornish. The iDorm: Gateway to Heterogeneous Networking Environments. Proc. Int’l Test and Evaluation Association (ITEA) Workshop Virtual Home Environments, ITEA Press, 2002, pp. 30-37.