Creating Resilient Self-Healing Agent Environments

Stefano Bromuri and Michael Schumacher

University of Applied Sciences Western Switzerland,
Institute of Business Information Systems,
3960, Technopole 3, Sierre, Switzerland
{stefano.bromuri,michael.schumacher}@hevs.ch

Abstract. In the context of pervasive healthcare systems [15] there is a growing need of services that are constantly available to the patients accessing them. To address this issue, in this paper we present a distributed pervasive infrastructure that is capable of self-healing one or more of its parts when an external event causes a disruption of the service in the areas covered by the pervasive system. We utilise approaches from multi-agent systems (MASs) such as communication, coordination, planning and agent environments to create a distributed system whose emergent behaviour shows the capability to heal itself even if 60% of the system is not functioning due to external causes.

1 Introduction

In fields like pervasive computing and ambient intelligence there is a growing need of models that allow the system to be independent from human intervention. This is particularly true for the emerging discipline called *pervasive healthcare* [15]. Such a field is focused on bringing healthcare everywhere and for everyone, breaking the boundaries of hospital healthcare. Pervasive healthcare systems (PHSs) are intrinsically distributed and present the need of been extremely fault tolerant as a downtime of the system may be potentially dangerous for the life of the patients relying on them.

Due to this strict requirements, centralised approaches should be avoided as the number of mobile services (i.e. ambulances) and patients utilising the infrastructure may be very high. Also, due to the spatially-distributed nature of pervasive-healthcare applications, it is necessary to consider approaches that are more sophisticate than simple space-based redundancy. As defined by Gärtner in [7], space-based redundancy approaches assume the possibility of replicating the functionalities of a node/component to improve the resilience of an infrastructure, but this becomes expensive when the number of nodes/components participating in the system is already very high.

At the same time it is not acceptable to just leave the system fail, as a patient may rely on the pervasive healthcare system to deal with his or her physiological signals. Since in PHSs the fault tolerance model based on space-based redundancy it is of little or no help, it is necessary to consider a different schema,

such as the one proposed by self-organising systems, that rather focus in what Gärtner defines as time-based redundancy, that implies replicating behaviours and functions in the components of the system to allow the system to adapt at run-time to a disruption. Self-organising systems draw inspiration from natural systems [1], biological systems [10] and social interactions [11], where entities with self-similar behaviours interact to form complex interactions.

In this sense self-organising systems [19] can be a useful way to manage the complexity of applications as the ones proposed by pervasive healthcare. In particular, we are interested in self-healing systems [9], a particular branch of self-organising systems that focus on resilience and fault tolerence.

The adoption of abstractions such as agents and multi-agent systems [18] is facilitating the transition from a centralised model of computing to a decentralised and distributed model where possible thousands of entities, agents, interact to achieve some common goal. Moreover, the concept of agent environment [17] has been accepted as a useful abstraction to mediate the interaction between agents and to model the resources and interfaces that the agents utilise in their interaction.

Since agents are by definition autonomous entities, MASs represent a valid abstraction to model self-organising systems by having complex properties emerge from the interaction taking place between hundreds of agents.

Arguably, a combination of the following drawbacks affects nowadays pervasive healthcare systems: (a) faults of the distributed system are never taken into consideration; (b) the topology of the system is statically defined; (c) there is not a clear approach to embed intelligence in the distributed system. In particular, in the system that we presented in [2] we proposed a distributed agent environment capable to support abductive logic programmed agents to monitor pregnant women affected by gestational diabetes mellitus. Amongst the assumptions proposed by such a system, there is the mapping between the agent environment represented as a distributed cellular automaton and a real environment representing a city. In this paper we propose to extend the system presented in [2] with an approach based on planning, coordination, communication to tackle situations when cells of the agent environment fail due to external causes, thus making the agent environment capable to self-heal.

The contributions of the system proposed in this paper can be summarised as follows. First of all we introduce a possible approach to the complexity of fault tolerance in pervasive healthcare systems. Secondly, we show that the introduction of the agent environment as a medium of interaction allows to simplify the behaviour of the agents involved in the self-healing process. Finally we show that using simple planning agents allows us to deal with multiple faults of the distributed environment in parallel.

The reminder of this paper is structured as follows: in Section 2 we discuss the requirements of a pervasive healthcare system; Section 3 describes the self-healing system we developed in terms of its main components to handle the requirements previously described; Section 4 is an evaluation of our approach;

Section 5 discusses the main related work; finally Section 6 concludes this paper and draws the lines for future work.

2 PHSs Requirements

In this paper we are motivated by the requirements of pervasive healthcare systems (PHSs). As specified by Varshney in [15], PHSs are complex systems where multiple components interact to allow large scale monitoring of physiological data originated by a set of possibly heterogeneous patients using a Body-Area-Network (BAN). A typical topology of a PHS is shown on the top of Fig. 1. In such a system every cell represents a different location of the real environment. The patients have access to such cells by means of a mobile device that has Internet connectivity capabilities.

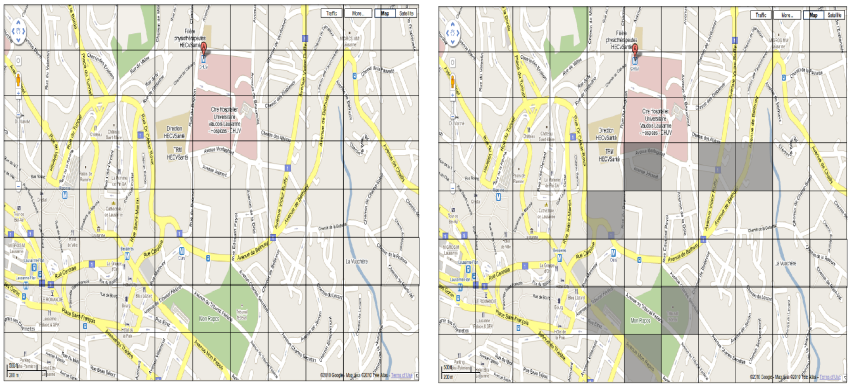

Fig. 1. Division in Cells of Pervasive Healthcare System (on the left) and Cell Disruption (on the right).

Every cell of the environment is defined by its boundaries in terms of its absolute GPS location and by its neighbourhood. When the system functions correctly, the patients can migrate from one cell to another by moving in the city, as it happens with a cellular network.

It is clear that PHSs systems have strong requirements from the standpoint of *availability* and from the standpoint of *fault tolerance*. In other words, if a fault happens to a cell of the system, like shown on the bottom of Fig. 1 (disruptions shown in gray), the system must be able to recover quickly, even if with degraded functions, to minimise the probability that an emergency occurs during the downtime. At the same time, the number of cells that cover a city can vary according to the city and according to the number of patients in the city. In general, the number of cells can be very high, as a consequence, to satisfy the availability and fault tolerance constraints, it is necessary to limit the use of centralised components as they offer a single point of failure. As Gärtner specifies in [7], redundancy is necessary for fault tolerance, but for PHSs it is important

to avoid solutions that are redundant in space as they would be very expensive for this kind of applications, but rather use solutions that are redundant in time, by replicating the behaviour in distributed self-similar cells.

In this paper we focus on creating a distributed agent environment that can support a PHS and at the same time satisfies the availability constraint previously discussed. In particular, we will focus on defining those behaviours of the distributed system that provide the necessary fault tolerance, while we will not discuss other components of the PHS, such as the BAN, how the hubs/smartphones are programmed and how the data is interpreted as these issues are out of the scope of this paper.

3 Pervasive Healthcare Systems

In this paper we extend the PHS presented in [2] to deal with cell disruptions. To extend such a system, we made use of the GOLEM agent platform [4] whose main abstractions are agents, cognitive entities, objects, reactive entities available to the agents as resources, and containers, declaratively programmed spaces where agents and objects are situated in possibly distributed settings, defining a distributed agent environment.

It is important to clarify that with the term *environment* we mean the world that is external to the agents and that the agents can inspect by using the *agent environment* [17]. On one hand we define the *agent environment* as an entity that mediates the interaction between the agents and resources deployed in the system, working as medium of interaction. On the other hand the *agent environment* hides to the agents the complexity of dealing with the state of the *environment*, by providing standard interfaces and standard descriptions to the resources in the external environment.

In the scope of this paper we use *environment* in terms of a place or a set of places delimited by borders defined in terms of longitude and latitude in the real environment. Longitude and latitude are mapped to a *distributed agent environment* for monitoring purposes, where every node of the distributed agent environment has an assigned area of the real environment. GOLEM is based on the Ambient Event Calculus [4], a particular dialect of the Event Calculus [12] that can handle the interaction between distributed containers. Such a formalism allows containers to mediate the interaction in distributed settings, and it will allow us to define procedures to deal with the disruption of cells in the distributed system. The main predicates of this formalism are discussed in Fig. 2, following the conventions of first-order logic that represent predicates with an upper case letter and variables with a lower case letter.

Furthermore, GOLEM entities are specified in terms of their observable state, called affordances [3]. The affordances in GOLEM are complex C-logic structures [5] that evolve over time. C-logic is a formalism that has a direct translation to first order logic and it is convenient to describe the structure of complex entities such as agents, objects and containers. For example we can express the state of a container at a given time by means of the following C-logic structure:

Predicate	Description
Initiates/5	The Initiates/5 predicate defines how the events produced in the agent environment change the attributes of a C-logic term to a new value.
Terminates/5	The Terminates/5 predicate defines how events produced in the agent environment terminate the value of an attribute of a C-logic term.
HoldsAt/5	The HoldsAt/5 predicate allows to query the attribute of an object at a given time, within one container.
InstanceOf/3	The InstanceOf/3 predicate allows to query the class of an object at a given time.
LocallyAt/8	The LocallyAt/8 predicate allows to query the attribute of an object in sub containers.
NeighbouringAt/9	The NeighbouringAt/9 predicate allows to query the attribute of an object in neighbour containers.
RegionallyAt/9	The RegionallyAt/9 predicate allows to query predicates in distributed regions of super/sub and neighbour containers.

Fig. 2. AEC Predicates

```
Container:C1[Latitude ⇒ lat,Longitude ⇒ lon,Side ⇒ 50, State ⇒ Up,
             Neighbours ⇒ {Container:C2,Container:C3}]
```

which means that a container represent a location, or cell, of the real environment, it is associated with a latitude Lat and and a longitude Lon, its state is up, and it covers a square that has a side of 50 meters.

The main issue of the system described in [2], is that there is no procedure in place to deal with the failure of a container and the area assigned to the container is not covered until an operator amends the issue. To provide a solution to this problem we introduced a *infrastructure agent* for every container.

The infrastructure agents observe and perceive changes in the topology of the agent environment and they coordinate with neighbour agents to cover for the fault of one or more containers. The plans are instantiated and executed in parallel to minimise the downtime, while the agent communication allows to minimise the excessive expansion of the containers ensuring that the workload is equally distributed. Moreover, we modified the behaviour of the distributed containers to re-route the messages exchanged between the agents according to the neighbourhood known in previous interaction, splitting in this way the responsibility to recover from the fault from the responsibility of dealing with the topology. The overall result is a system that is capable to handle the failure of a cluster of containers in the distributed agent environment and continue functioning even if with degraded functions until the normal behaviour of the system is restored.

3.1 The Containers Behaviour

In this paper, we extended the behaviour and architecture of the GOLEM containers to handle self-healing procedures. Fig. 3 shows the architecture of PHS GOLEM containers used in this paper. In particular we modified the existing GOLEM containers by loading them with rules to handle self-healing procedures, communication during self-healing and by adding an infrastructure agent as a permanent service deployed in every container.

We assume that a newly deployed container in the distributed agent environment is created with direct knowledge of its direct neighbours. We also assume that the containers cover exactly one location of the real environment.

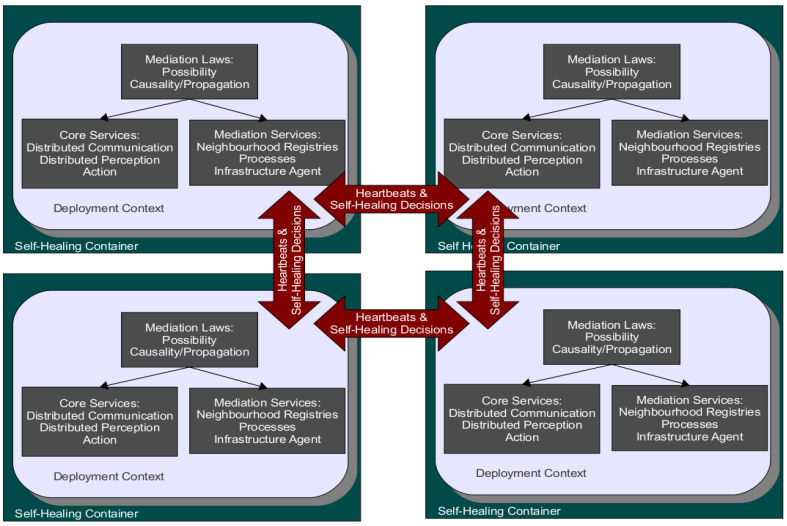

Fig. 3. The Architecture of a PHS GOLEM Container.

For the moment, we do not focus on how a container can join an existing network, we assume that this task is performed by a human actor. We also assume that once a container is put down due to external causes, this can be redeployed only by a human actor. These two problems are out of the scope of this paper, but they will be addressed in future publications.

Given the aforementioned assumptions, we extended the behaviour of the GOLEM containers and of the AEC, introducing the PropagateAt/3 predicate to handle the propagation of messages produced by agents in the distributed settings. The PropagateAt/3 predicate is specified as follows:

R1) PropagateAt(Ack:event,$ContainersList$,t)←
Happens(event,t), event[Actor ⇒ agent, Receivers ⇒ $ContainersList$],
(∀ x ∈ $ContainersList$ | HoldsAt(x,Neighbour,this,t)).

R2) PropagateAt(Inform:event, $Containers$, time)←
event[Actor ⇒ agent[Container ⇒ sendercontainer], Cover ⇒ cbroken, RandomValue ⇒ diceroll]
(∀ x ∈ $Containers$, $Containers \subseteq KnownContainers$ |
HoldsAt(x,Neighbour, cbroken,time),HoldsAt(x,State, Up,time)).

R3) Initiates(cid,Container,Neighbour, sendercontainer, ev) ←
cover:ev[Actor ⇒ agent[Container ⇒ sendercontainer], Container ⇒ cbroken]
Time(ev,T), HoldsAt(cbroken,Neighbour,this,T).

R4) Initiates(cid, Container,State, Down, ev) ← notify_failure:ev[Actor ⇒ agent, Down ⇒ cid].

The interpretation of the PropagateAt/3 is as follows. The R1 rule is triggered whenever there is an acknowledgment message submitted by a infrastructure

agent to the neighbour containers. The idea here is that at deployment time the containers have information only about the direct neighbourhood, while the information about the other containers in the network is learned by message exchange. The message exchange takes the form of an *acknowledgement message* that is issued by the agents at regular intervals to check for the health of the distributed system. On one hand, for practical reasons and to avoid global knowledge of the network, we have chosen to have agents that know only which are their neighbours and tha can learn by message exhange which are the neighbours of their neighbours. On the other hand, the containers can modify their knowledge of the topology and modify their boundaries to cover for a disruption. When this happens, the neighbourhood of the container changes their knowledge about the neighbourhood to adapt to the new topology using R3 and R4. The R2 rule propagates the information that an agent wants to cover for a container that has been detected to be down by broadcating the information to all the neighbours of the dead container that are known to be alive. The introduction of R2 simplifies the behaviour of the agent that does not need to keep a routing table to know which is the current neighbour of a certain container. Rules R3 and R4 specify respectively that the current container is neighbour of another container if its infrastructure agent is covering for a dead neighbour of the current container. In particular, the event described in R4 takes place when an agent does not receive an inform event from a neighbour container at the right interval of time or if the communication with a container fails resulting in an notification of failure event. The routing table of the dead or alive containers in the neighbourhood of the container is handled by means of C-logic terms and it can be queried by the agents by means of the predicates of the AEC in the declarative context of the GOLEM container. To conclude, rules R3 and R4 have two correspondent Terminates/5 rules to specify when a container ceases to be a Neighbour or when it ceases to be dead.

3.2 The Infrastructure Agents

In GOLEM, agents are situated entities that have a body with a set of sensors and effectors that can perceive and act in the agent environment and a declarative mind connected to the body by a brain interface. What is important to state is that the declarative agent mind is a separated module with respect to the agent environment, which means that, despite the fact that the body is constrained by the rules of the agent environment, the agent preserves its autonomy in the reasoning process, and that the body and the mind work as coroutines that share queues for percepts from the environment and actions to the environment. We extended GOLEM and we introduced an infrastructure agent that is situated in every container of the distributed agent environment and that is capable to interact with other infrastructure agents to *heal* the agent environment when a disruption takes place.

The agent cognitive model is based on two cycles, one to observe the environment and the other one to plan and act in the environment. The agent mind for the self-healing process is also based on the Event Calculus, but the theory

that represent the reasoning process is quite complex, as a consequence we report here the pseudo code of the two cycles that represent the agent mind (CSP stands for Conditional-STRIPS-Planner):

```
procedure ACTING-Cycle(time)
    static:KB, a knowledge base
           ACTION-QUEUE, a queue of actions
           accessible by the agent body,
           p_1, p_2 ... p_k, where ∀p_i, p_i ∈ Plans ⊂ KB, a set of plans

    currentstate ← STATE-DESCRIPTION(KB,time),
    goal ← NEXT-GOAL(currentstate,time),
    ADD(Plans,p_1),
    if ∄p_i|p_i.goal = goal then
           p_k ← CSP(currentstate,goal), ADD(Plans, p_k)

    p_exec ← NEXT-EXECUTABLE-PLAN(Plans,time),

    if(p_exec = nil)then
                    NOW(timenew),
                    ACTING-Cycle(timenew)
    else
        currentaction = p_exec.nextaction,
        if(CONDITIONAL?(currentaction)) then
              if(CHECK-KB(KB,IF-PART[currentaction]))
                    then p_exec ← THEN-PART[currentaction],
                                  NOW(timenew),
                                  ACTING-Cycle(timenew)
              else p_exec ← ELSE-PART[currentaction]
                            NOW(timenew),
                            ACTING-Cycle(timenew)
        else
            ADD-ACTION(ACTION-QUEUE, currentaction),
            NOW(timenew),
            ACTING-Cycle(timenew)

procedure PERCEPTION-Cycle(time)
    static:KB,
           PERCEPTION-QUEUE,
           p_1, p_2 ... p_k, where ∀p_i, p_i ∈ Plans ⊂ KB
    percept ← NEXT-PERCEPT(PERCEPTION-QUEUE, time),
    UPDATE-KB(KB, percept, time),
    NOW(timenew),
    PERCEPTION-Cycle(timenew)
```

On one hand the ACTING-Cycle is in charge to instantiate plans and execute the actions within the plan by pushing the action in the ACTION-QUEUE and consequently in the agent body and effectors. Notice that if the plans are of different nature, or they have a different goal, they can be instantiated and executed in parallel. Due to the nature of the problem of self-healing, this does not represent a problem, because there are not conflicting plans, as they are associated to different cells of the environment. The agent mind is capable to deal with plans that are conditional, as a consequence when a conditional action arises, the agent will check the condition within the knowledge base and execute the subplan whose condition results to be true at a certain time. The PERCEPTION-Cycle reads the PERCEPTION-QUEUE for percepts coming from the agent body sensors. Such percepts are used to update the knowledge base KB about the state of the containers that are known in the agent environment.

At the same time the updating process can destroy plans in the case these are not reachable any more. During the normal functioning of the network, the agents

observe the environment at regular intervals, updating their knowledge about their neighbours. One one hand, the agents do not have global knowledge of the environment as this would cause severe inconsistencies every time the network changes. On the other hand, the agents have knowledge about the neighbourhood of its direct neighbourhood as this helps in situations when the disruption in the distributed agent environment hits cluster of neighbour cells.

The agent planning capabilities are based on a modification of STRIPS [6] to handle conditional plans. In particular, the plan to handle the disruption of neighbour cells is composed by means of the following atomic principles:

```
Op(ACTION:Broadcast(Inform(Cover(aid,containerid,diceroll))),
  PRECONDITIONS:[Down(containerid), IAM(aid), Diceroll(diceroll)],
  EFFECTS:[Add(PreviousAction(Inform(Cover(aid,containerid,diceroll)))),
           Add(Diceroll(diceroll))]).

Op(ACTION:Timeout(Time),
  PRECONDITIONS:[PreviousAction(Cover(aid,continerid,direroll))],
  EFFECTS:[Add(KnowsWhether(winner(aid,containerid,value))),
           Del(PreviousAction(Cover(aid,continerid,diceroll)))]).

Op(ACTION:end_plan,
  PRECONDITIONS:[Down(containerid), IAM(aid),
                 KnowsWheter(winner(aid,containerid,False)))],
  EFFECTS:[add(End)]).

Op(ACTION:ModifyTopology(aid,containerid),
  PRECONDITIONS:[KnowsWhether(winner(aid,containerid,True)))],
  EFFECTS:[Add(Covers(aid, containerid))]).
```

The actions specified above represent the atomic actions of the plan in Fig. 4.

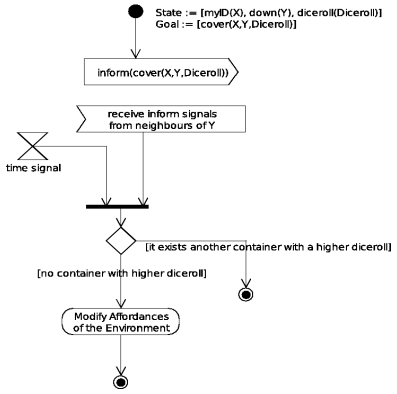

Fig. 4. Plan to Cover a Dead Cell of the Environment

Such a plan is triggered in the agent mind when a neighbour container is perceived to be down at a certain time due to a missing acknowledgement repeated three times or due to a failed communication with a neighbour container. Whenever one of these situations arise, the infrastructure agent submits an inform message to the agent environment.

The inform message contains the informations about the sender, the dead cell and a diceroll value. Such a value is a random value used to compete with the neighbour agents for covering for the dead neighbour.

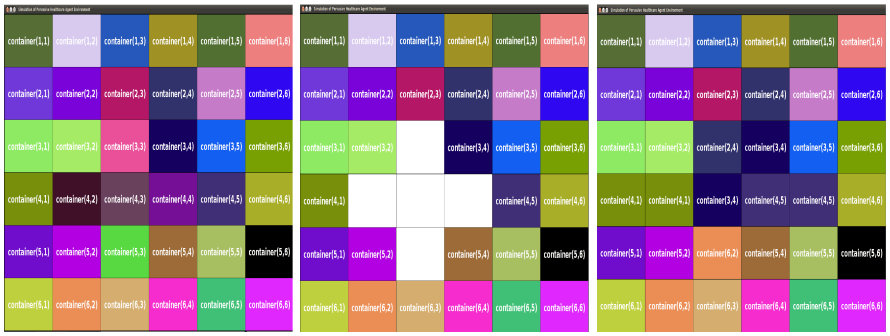

Fig. 5. Simulation of Interaction of a 6x6 Pervasive Agent Environment with service disruptions

Such an inform message is propagated by the environment by means of the PropagateAt/3 predicate to the neighbours of the dead cell that are known by the infrastructure agent to be alive. After the submission of the inform message, the agent waits for inform signals coming from the other agents until a timeout is internally triggered in the agent mind. After the timeout, the agent checks the received signals and if it is the winner of the competition, it proceeds to cover for the dead cells, otherwise it ends the plan as it is not reachable any more, due to the fact that there is another container with a higher diceroll that is going to cover for the disruption.

It is worth noting that this simple behaviour could be handled using purely reactive agents. We argue here that, despite the fact that reactive agents could handle the disruptions of containers, at the same time defining termination procedures for unreachable actions would require updating the state of the agent in every reactive rules defined to handle such an action. Instead, we prefer to have agents capable to reason about their actions, reifying the concept of plan as structures on which the agent can reason about, as this allows for a more concise and clean definition of the agent mind. Another solution different than planning could be to span a different agent whenever a neighbour container dies and then destroy the agent as soon as the solution is covered. Although current hardware architectures support more and more multi-threading, such a practice in a PHS would be a lot more resource demanding than the current approach proposed in this paper as an agent is understood to be an active entity with its own control flow and its own allocated resources. To illustrate how the plan in Fig. 4 works from the perspective of the agents interacting in the neighbourhood, let us take into consideration the example shown in Fig. 5. The panel on the left in Fig. 5 shows a set of GOLEM containers represented by a different color in the cellular automaton. The central panel in Fig. 5 shows that four of containers

have undergone a service disruption. The panel on the right in Fig. 5 shows that the area of the containers that have undergone a service disruption have been covered by their neighbours.

The introduction of parallel plans help the agents to minimise the downtime for the coverage of a dead cell as whenever a neighbour is recognised to be dead, a new plan is instantiated within the agent mind.

At the same time the introduction of agent communication, aided and mediated by the agent environment, allows to minimise a) the number of messages exchanged in the environment b) the uncontrolled growth of the area controlled by a cell and c) the conflicts arising between cells covering for a dead neighbour at the same time.

In particular, it is important to say that, despite the communication taking place between the agents, two cells may end up covering the same broken cell. In this case, when the inconsistency is perceived, the interested agents start a resolution protocol similar to the one in Fig. 4 to decide who should give up the covered area, but we omit this from the description as the protocol itself is very simple.

4 Evaluation

In a pervasive healthcare system like the one presented in this paper there are two measures that are critical. One is the downtime of the system due to a failure. The higher is the downtime the higher is the probability of an emergency happening while the system cannot cover it. Secondly the number of messages exchanged in the system to recover from a downtime. This is a sensible measure as if the number of messages exchanged is very high, this could have impact on the performances of the healthy part of the system, even causing further disruption.

We evaluated the overall behaviour of our system by deploying a 10x10 Grid of GOLEM containers with self-healing capabilities, taking an average of a set of 10 repeated tests for every curve showed in Fig. 6. For the evaluation we used a dual core Intel Centrino 2, 2.66 Ghz per core with total of 3Gb of RAM.

Despite the fact that we deployed the real system within a single machine, this evaluation only emulate what could be the behaviour of the system in real settings. In real settings, the system should be distributed on a GRID of distributed machines that all have their own resources. In particular, we imagine that in a real PHS application, some of the containers would require more resources as they represent hot spots in the real environment (i.e. super markets, schools or hospitals) while some other areas require less resources and as a consquence multiple containers could be deployed on the same machine or even one container could just be assigned to a bigger unpopulated area. For the moment this is still matter of future work, but we recognise the need to study the system with the presence of patients in it.

As evaluated by Urovi et al. in [14], a GOLEM container can support up to 50 avatar agents with acceptable performances, which means that a GRID of

100 containers can support up to 4000-5000 users, that is a meaningfull scale for a real system in a city where there are about 200.000 people such as the city of Lausanne where the number of patients that need constant monitoring regimes are on the order of some hundreds.

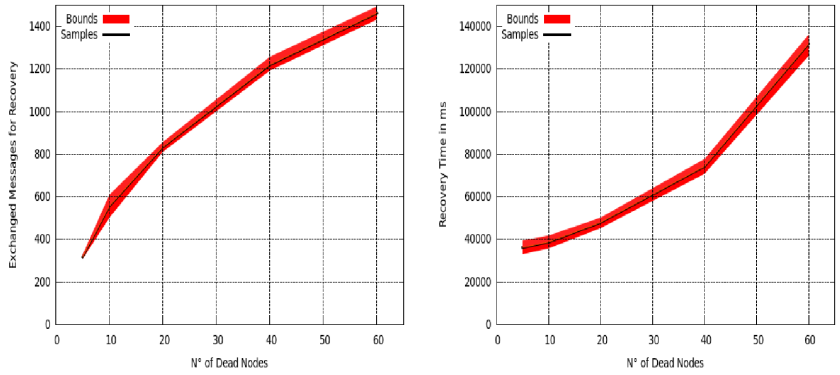

Fig. 6. Performance Evaluation of a 10x10 Pervasive Agent Environment

When producing the curves in Fig. 6 we made the following set of assumptions. First of all, since the system is deployed in a single machine, the delays of a real network are not taken into consideration. We assumed for both of the graphs that the cells die all at the same time, although this is not realistic, it is a pessimistic assumption. A third assumption is that the system presents failures after every cells had the time to learn about its neighbours and their immediate neighbourhood. The last assumption is that the failures are random and do not depend on a particular pattern, although we can handle small cluster of dead cells, for practical reasons our algorithm cannot handle the failures of very big clusters of cells, meaning that we assume the failure to be distributed in the network and that the agents have not a global knowledge. The last assumption is not a big limitation as it is related to catastrophic events happening in the network that in many cases are impossible to handle without an operator reparing the physical hardware in an area of the pervasive system.

The part on the top of Fig. 6 shows the number of dead cells with respect to the number of messages that the containers and the agents within them need to exchange to cover for the missing containers. The behaviour of the curve is logarithmic. This happens because the more nodes in a neighbourhood die, the less nodes take part to the competition for covering a dead node, as a consequence there are less messages to exchange in the neighbourhood. The direct drawback is that the agents still alive have to instantiate and execute more plans to cover for the dead cells. The effect of this can be seen on the part on the bottom of Fig. 6. The time to recover from a set of faults in the network behaves as a mildly quadratic curve as far as the percentage of dead nodes in the network is less than 40%, but when percentage of dead nodes goes over 50%, the downtime has a peak. This happens because the agents have to cover for multiple cells in

their neighbourhood, resulting in the instantiation of multiple plans which is a time expensive operation.

To conclude the evaluation, the two graphs show that the healing agent environment can scale up as the message exchange is kept local to a neighbourhood thanks to a combination of interaction between the agents and the rules of their environment, and that the limit for recovering from a failure is that at least 50% of the cells of the distributed agent environment are still alive.

5 Related Work

The pervasive health care field is relatively new, but it has its roots in the distributed computing area, where multiple solution have been proposed in the past to deal with distributed resiliency.

Mikic-Rakic et al. in [13] defined a set of properties that is necessary to address to have true self-healing systems. In particular these properties are identified as: adaptability, dynamicity, awareness, observability, autonomy, robustness, distributability, mobility and traceability. The authors then propose the PRISM model as the answer to the above properties. Such a model is based on having a set of components connected by means of communication ports that exchange synchronous and asynchronous events and rely on metal-level components for the self-healing procedures. With respect to PRISM, the infrastructure agents can be thought as meta-level components with knowledge about the topology of the distributed system, while the communication between the components is done by means of message exchange amongst the infrastructure agents. In particular, to foster the observability property of the topology of our distributed system we use a declarative approach based on planning and on extending the rules of the Ambient Event Calculus [4].

From the stand point of self-healing systems, in [8] Selvin et al. in propose to utilise a bio-inspired approach to rebuild geometric shapes and then they apply this approach for a distributed wireless file service. Selvin et al. demonstrate that using nature inspired models like cell division and morphogenesis and wound healing allows to have a very resiliant service that is capable to self-heal despite the fact that 99% of the network is dead.

Our approach is similar to the one proposed by Selvin et al. except that we have a further constraint on the number of messages exchanged and on the time to recover from the failures. In this sense, with respect to the work in [8], the fact that we have planning agents that interact producing simple plans in parallel helps to minimise the downtime, while the communication process, aided by the rules of the environment, helps to minimise an uncontrolled expansion of the cells in their neighbourhood.

The work of Kondacs in [10] is another example on how using bio-inspired models allows to create robust systems. In particular Kondacs proposes a self-healing systems where the general principles are formalized as a programming language with explicit primitives, basing the self-organisation on processes such as morphogenesis and developmental biology. Kondacs demonstrates that, de-

spite having cells in the system dieing at a high rate, the system is capable to self-regenerate the missing part. Similarly to Kondacs, we utilize a declarative approach to define the rules for with the healing process takes place, but differently from what proposed by Kondacs, we have a further constraint related to the mapping of the system to a real environment which does not allow us to have an uncontrolled growth of the cells to cover for the dead cells.

In [16] Haesevoets et al. present the MACODO system for self-healing and self-adaptive agent organisations. In particular the MACODO system is based on splitting the responsibilities of the agents in terms of roles and define a set of laws in the agent environment to handle the consistency of the roles. Similarly to the work in [16], we separate the concerns of handling self-adaptation between the agent environment and the agents, but differently from [16], we use cognitive agents to deal with the changes of the environment, which allows us to deal with its failure and inconsistencies in parallel.

6 Conclusion and Future Works

In this paper we presented a pervasive healthcare system where agents reorganise the agent environment to self-heal from a fault of one more of the cells composing it. The approach makes use of planning agents that can reason in parallel about multiple faults and that produce plans and interact to cover for the missing cells in the environment, and it makes use of distributed agent environments programmed to propagate the messages produced by the agents according to the alive neighbourhood. The novelty of the approach resides in the use of planning agents combined with a complex declarative agent environment that simplifies the interaction between the agents controlling the distributed system. We have evaluated the behaviour of the system from the point of view of the downtime and number of message exchanges required to recover from a growing number of dead cells, discovering that the system as actually conceived scales up but it can recover in usefull time even more than 50% of the cells is down. We also compared our work with the existing literature.

Future works include deploying the system in real setting and testing it with real users as well as extending the algorithm to define the topology of the environment dynamically at deployment time. Another issue that we will take into consideration in future work is how to deal with patients roaming in a distributed network that is self-healing from a disruption. Other possible directions imply improving the current results further, avoiding broadcasting in the neighbourhood for every single dead cells, as well as considering more complex distributed agent environments that include indoor environments, using a hierarchical topology rather than a cellular automaton.

References

1. P. Balaprakash, M. Birattari, T. Stützle, Z. Yuan, and M. Dorigo. Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem. *Swarm Intelligence*, 3(3):223–242, 2009.

2. S. Bromuri, M. I. Schumacher, and K. Stathis. Towards distributed agent environments for pervasive healthcare. In *Proceedings of the Eighth German Conference on Multi Agents System Technologies (MATES '10)*, 2010.
3. S. Bromuri and K. Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multiagent Systems (EEMMAS'07)*. Springer, Oct 2007.
4. S. Bromuri and K. Stathis. Distributed Agent Environments in the Ambient Event Calculus. In *DEBS '09: Proceedings of the third international conference on Distributed event-based systems*, New York, NY, USA, 2009. ACM.
5. W. Chen and D. S. Warren. C-logic of Complex Objects. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 369–378, New York, NY, USA, 1989. ACM Press.
6. R. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *IJCAI*, pages 608–620, 1971.
7. F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31:1–26, March 1999.
8. S. George, D. Evans, and L. Davidson. A biologically inspired programming model for self-healing systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 102–104, New York, NY, USA, 2002. ACM.
9. D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya. Self-healing systems – survey and synthesis. *Decision Support Systems*, 42(4):2164–2185, 2007. Decision Support Systems in Emerging Economies.
10. A. Kondacs. Biologically-inspired self-assembly of two-dimensional shapes using global-to-local compilation. In *IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence*, pages 633–638, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.
11. R. Kota, N. Gibbins, and N. R. Jennings. Self-organising agent organisations. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *AAMAS (2)*, pages 797–804. IFAAMAS, 2009.
12. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.
13. M. Mikic-Rakic, N. Mehta, and N. Medvidovic. Architectural style requirements for self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, WOSS '02, pages 49–54, New York, NY, USA, 2002. ACM.
14. V. Urovi, S. Bromuri, K. Stathis, and A. Artikis. Towards runtime support for norm-governed multi-agent systems. In F. Lin, U. Sattler, and M. Truszczynski, editors, *KR*. AAAI Press, 2010.
15. U. Varshney. *Pervasive Healthcare Computing: EMR/EHR, Wireless and Health Monitoring*. Springer Publishing Company, Incorporated, 2009.
16. D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, and W. Joosen. The macodo middleware for context-driven dynamic agent organzations. *ACM Transactions on Autonomous and Adaptive Systems*, 5(1):3.1–3.29, February 2010.
17. D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.
18. M. Wooldridge. *MultiAgent Systems*. John Wiley and Sons, 2002.
19. F. Zambonelli and M. Viroli. Architecture and metaphors for eternally adaptive service ecosystems. In C. Badica, G. Mangioni, V. Carchiolo, and D. D. Burdescu, editors, *IDC*, volume 162 of *Studies in Computational Intelligence*, pages 23–32. Springer, 2008.