A Self-Healing Distributed Pervasive Health System

Stefano Bromuri*, Michael I. Schumacher

University of Applied Sciences Western Switzerland,

Business Information Systems Department,

Rue de Technopole 3, CH3960, Switzerland

E-mail: stefano.bromuri,michael.schumacher@hevs.ch

*Corresponding author

Kostas Stathis

Royal Holloway University of London,

Computer Science Department,

Egham Hill, Egham, TW20 0EX, United Kingdom

E-mail: kostas.stathis@rhul.ac.uk

Abstract:

In the context of pervasive healthcare systems 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 50% of the system is not functioning due to external causes.

Keywords:
Pervasive Health, Distributed Agent environments, Self-Organisation, Personal Health Systems, Distributed Event Based Systems, Self-Healing

1 Introduction

Pervasive Healthcare [Varshney, 2009] focuses on bringing healthcare everywhere, breaking the boundaries of hospital healthcare. In [Varshney, 2009] Varshney defines Personal Health Systems (PHSs) as complex systems where multiple components interact to allow large scale monitoring of physiological data of heterogeneous patients. Two main limitations affect existing PHSs: (a) failures of the distributed system are never taken into consideration and (b) the system topology is statically defined. Consequently, PHSs need fault tolerance mechanisms as system downtimes may be dangerous for patients relying on them. In PHSs
centralised fault tolerance should be avoided as it may create a bottleneck due to the large amount of data processed in PHSs and because it proposes a single point of failure.
Also, fault-tolerance in PHSs should not be achieved by what Gartner defines in [Gartner, 1999] as space-based redundancy, which improves the
resilience of an infrastructure by replicating components, an expensive practice for distributed systems like PHSs that serve possibly many patients at once and that have already distributed nodes to load balance the traffic associated with such patients, requiring a trade-off between the number of patients served and the resiliance of the system.
In this paper we address PHSs fault tolerance via the self-healing paradigm [Ghosh et al., 2007] that focuses on what Gartner defines as time-based redundancy, which, instead of replicating components, replicates components behaviours to ensure that if a component fails an existing component can substitute it. Mikic-Rakic et al. in [Mikic-Rakic et al., 2002] identified the following properties as necessary for self-healing systems: adaptability, dynamicity, awareness, observability, autonomy, robustness, distributability, mobility and traceability.
Multi agent systems (MASs) [Wooldridge, 2002] represent a valid abstraction to model such systems and fulfill the self-healing paradigm requirements. In particular, the adoption of MASs facilitates the transition from a centralised computing model to a decentralised one where thousands of autonomous agents interact to achieve a common goal. The use of the concept of agent facilitate the modeling of such infrastructures with respect to lower level abstractions such as processes, threads or Web services, this because agents are statefull entities, programmed with a goal and a plan to achieve the goal proactively. Moreover, the concept of agent environment [Weyns et al., 2007] has been accepted as a useful abstraction to mediate the interaction between agents
and to model how the agents perceive resources and interfaces that they utilise in their interaction.
In [Bromuri et al., 2011b, Krampf et al., 2011, Bromuri et al., 2010a] we proposed a PHS based on a distributed agent environment built on the GOLEM platform [Bromuri and Stathis, 2009], to support intelligent agents monitoring pregnant women affected by gestational diabetes. Amongst the assumptions proposed in [Bromuri et al., 2011b, Krampf et al., 2011, Bromuri et al., 2010a], there is the mapping between the agent environment represented as a distributed rectangular grid and a real environment representing a city. In this paper we extend the system presented in [Bromuri et al., 2011b, Krampf et al., 2011, Bromuri et al., 2010a] by defining a novel coordination and planning algorithm that agents use to detect faults and recover the system functionalities.
The contributions of this paper are: a) we introduce a practical approach based on agents to handle fault tolerance in PHSs; b) we split responsibilities between the agents and the agent environment thus simplifying the behaviour of the agents to fulfill the requirements of self-healing systems; c) we illustrate how a compact declarative specification for the agents behaviour can deal with multiple failures of the agent environment in parallel. d)

We introduce a load balancing technique for persisted agents which allows the PHS to scale up better.
The reminder of this paper is structured as follows: Section 2 describes the background necessary to understand this paper; Section 3 describes our PHS self-healing capabilities; Section 4 evaluates our approach; Section 5 discusses relevant related work; finally Section 6 concludes this paper and presents future directions.

2 Personal Health System Definition and Background

In [Bromuri et al., 2010a, 2011b, Krampf et al., 2011] we presented a Personal Health System (PHS) to monitor Gestational Diabetes. The aim of a personal health system is to speed up the delivery of medical care, by changing the way in which patients and doctors interact. The logic architecture of our PHS is built in three layers shown in Fig. 1. At the Mobile Interface layer, the patient can produce her physiological values and symptoms by means of a mobile phone application.
Such data is then sent to an Agent Environment (AE) where intelligent expert agents receive the data to perform reasoning about the patient status. Such an agent environment can be distributed in multiple hosts, or even be contained in a single host, according to the amount of patients monitored.
The AE component is also subdivided in cells associated to a particular area of a city, as we imagine the patients to be uniformely distributed within the city. Finally the Patient Management System allows the doctors to visualise the patient's data, to modify its treatment and to visualise the alerts produced by the AE. Fig. 2 shows such an interface. In particular, through the interface the doctors can observe the evolution of the patient's physiological values in response to the treatments and react promptly.

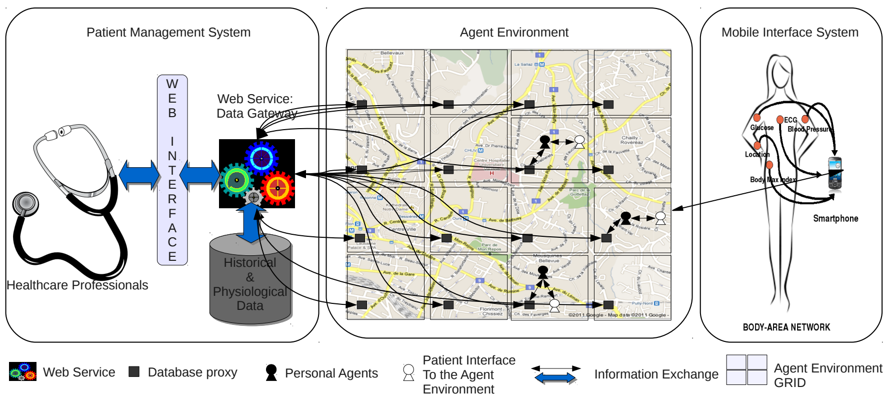
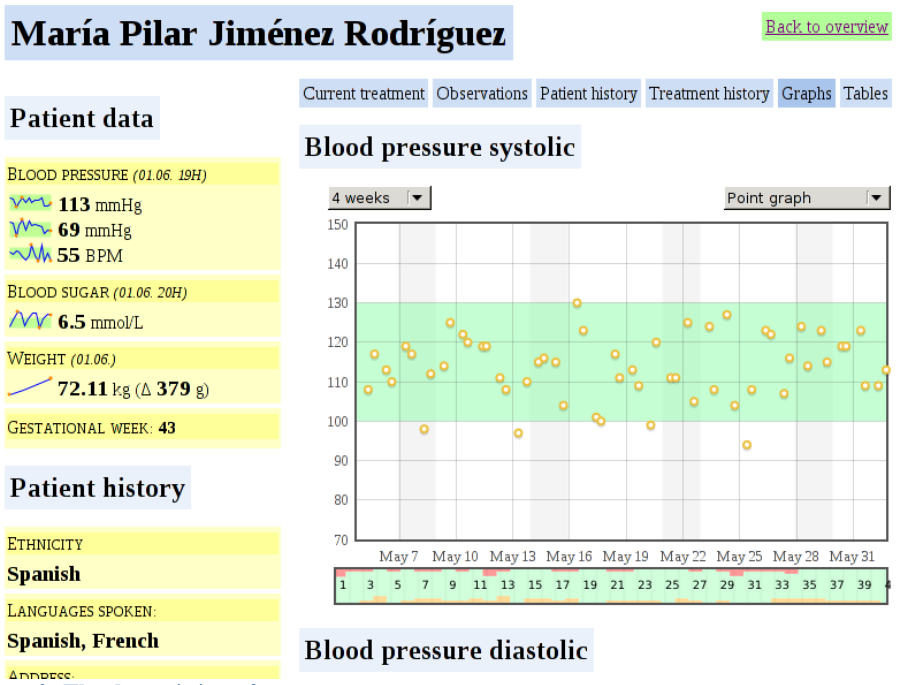


Figure 1: Personal Health System Architecture.

To create a self-healing pervasive healthcare agent environment, we decided to take our PHS and to extend it to deal with fault tolerance with respect to cell disruptions. To extend such a system, we made use of the GOLEM agent platform [Bromuri and Stathis, 2009, Bromuri et al., 2010b] 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. The GOLEM agent infrastructure was chosen because there have been already attempts

to use it in pervasive systems [Bromuri et al., 2010b,a] and because it considers the agent environment as a first class abstraction, thus allowing the agents and the objects residing in it to manipulate its state.
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 [Weyns et al., 2007]. 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.



María Pilar Jiménez Rodríguez

Patient data

BLOOD PRESSURE (01.06. 19H)
113 mmHg
69 mmHg
55 BPM

BLOOD SUGAR (01.06. 20H)
6.5 mmol/L

WEIGHT (01.06.)
72.11 kg (Δ 379 g)

GESTATIONAL WEEK: 43

Patient history

ETHNICITY
Spanish

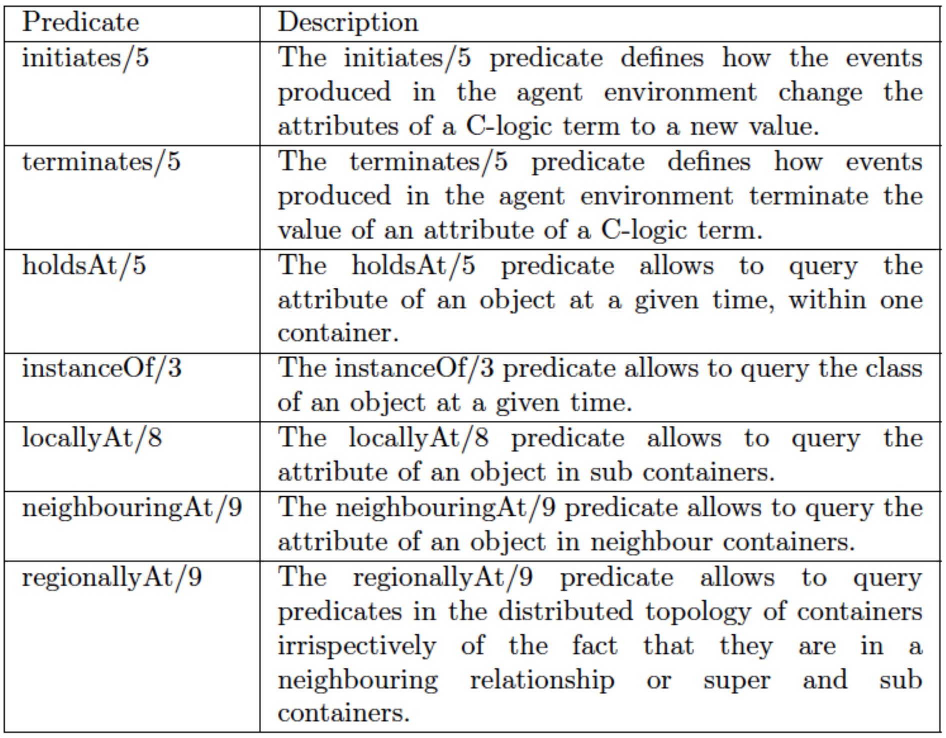
LANGUAGES SPOKEN:
Spanish, French

Blood pressure systolic

Blood pressure diastolic

Figure 2: The doctor's interface.

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 [Bromuri and Stathis, 2009], a particular dialect of the Event Calculus [Kowalski and Sergot, 1986] that can handle the interaction between distributed containers. Such a formalism allows containers to mediate the interaction in distributed settings. The main predicates of this formalism are discussed in Fig. 3, following a Prolog-like syntax where we represent predicates with an lower case letter, variables with an upper case letter. For example, the term **initiates/5** denotes a predicate with name initiates and arity 5.



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 the distributed topology of containers irrispectively of the fact that they are in a neighbouring relationship or super and sub containers.

Figure 3: The AEC Predicates.

Furthermore, GOLEM entities are specified as complex C-logic structures [Chen and Warren, 1989] 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. We can express and describe the state of a container at a given time by means of the following C-logic structure:

```
container:c1[
            latitude ⇒Lat,
            longitude ⇒ Lon,
            side ⇒ 50,
            state ⇒ up,
            neighbours ⇒fcontainer:c2,container:c3g
]
```

which means that a container represent a location, or cell, of the real environment, it is associated with a latitude Lat and a longitude Lon , its state is up , and it covers a square that has a side of 50 meters. The following two AEC rules (see Bromuri and Stathis [2009] for a more detailed description):

```
happens(Event,T)  ←attempt(Event,T), possible(Event,T).
happens(Event, T) ← attempt(Event, T), necessary(Event, T).
```

specify that an action in the GOLEM agent environment happens only if it has been attempted and it is possible or necessary, where **possible/2** and **necessary/2** rules are application dependent rules. In other words, possible/2 rules specify what are the actions that is possible to perform in the environment given its current (possibly distributed

state), while the necessary/2 rules specify what are the actions that happens as a consequence to previous events.
Moreover, to provide the mediation necessary to handle events in the distributed setting in Bromuri and Stathis [2009] we presented the **locally_at/8**, **neighbouring_at/9** and **regionally_at/9** primitive predicates to link the state of distributed containers, following a logic programming approach. Briefly, the definition of locally at is as follows:

```
locally_at(CId, Path, Path_, Id, Cls, Att, V, T) ←     locally_at(CId, Path, Path_, Id, Cls, Att, V, T) ←
holds_at(CId, container, entity of, Id, T),                instance_of(SCId, container, T),
holds_at(Id, Cls, Att, V, T),                               holds_at(SCId, container, super, CId, T),
append(Path, [CId], Path_).                                 append(Path, [CId], NewPath),
                                                            locally_at(SCId, NewPath, Path_, Id, Cls, Att,V,T).
```

The definition of **locally_at/8** states that the state of an entity can be inferred either from the top-level container or from a sub-container. If the states is inferred in the top-level container, then the predicate **holds_at/5** is applied to infer the attribute **Att** of value **V** of an entity of class **Cls** and identifier **Id**. If the first predicate fails, then the second predicate moves the computation in a sub-container. In this way containers can be recursively embedded inside other containers as objects, according to the topology needed, and deployed on different hosts. The **neighbouring_at/9** and the **regionally_at/9** predicates have a similar behaviour but allow to query adjacent and super-containers respectively. Finally, to specify how the state of an entity modifies over time, we utilise **initiates/5** and **terminates/5** rules. For example, the following **initiates/5** rule specifies when the position of an agent changes to the one the agent moves to:

```
initiates(E, avatar, A, position, Pos) ← do:E [actor ⇒ A, act ⇒ move:M [destination) Pos]].
```

The complete description of the event's effects also requires to terminate the attribute holding the old position of the agent by means of a terminate/5 rule.
In the current prototype the agent environment takes care of pairing agents and avatars as well as defining the mobility rules (i.e. what are the conditions that move an agent from one container to another). We define the following rules for mobility purposes:

```
possible(E,T) ←                                   possible(E,T) ←
        move:E[actor⇒avatar:A, move ) Pos],             instance of(Id,topology,T),
        instance_of(Id,topology,T),                     holds_at(Id,topology,borders,Borders,T),
        holds_at(Id,topology,borders,Bdr,T),            outside_borders(Bdr, Pos),
        inside_borders(Bdr, Pos).                       neighbouring_at(this, [], [C], 1, Id,
                                                         topology,  borders, Bdr, T),
                                                         inside_borders(Bdr,Pos).
```

the first one states that it is possible to move in the space represented by a container only if this space is within the borders controlled by the container. Otherwise, the second rule specifies that it is possible to move outside the borders only if there is another container that is responsible for a certain area where the patient is currently moving. The following AEC rules:

```
necessary(E, T) ←                                               necessary(E, T) ←
          happens(E*, T),                                                 happens(E*, T),
          deploy:E_[deploy ⇒avatar:Av],                                   disconnect:E_[actor ⇒A, new container  ⇒ C],
          not neighbouring_at(this, [], [C], 1, Av, caretaker, , T),      holds_at(A,avatar,caretaker,Id,T),
          deploy:E[agent ⇒caretaker:A].                                   physical_act:E[move_to ⇒ C, agent ⇒Id].
```

state respectively that whenever an avatar is deployed in the agent environment (event **E***), also its caretaker agent is deployed (event **E**), and that whenever an avatar disconnects from the agent environment to connect to a new container, the agent associated to the avatar is also serialised and moved to the new container.

Finally, a further **necessary/2** rule defines that an avatar is moved to a different container when outside the boundaries of the current container, but we omit the details as it is simpler than the ones presented above.
The main issue of the system described in [Bromuri et al., 2011b, Krampf et al., 2011, Bromuri et al., 2010a], as many other PHSs, is that it is statically defined and assumed to work, meaning that if a container fails, there is no procedure in place to deal with the failure 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 an infrastructure agent for every container as we will discuss in the next Section.

3 Extending GOLEM with Self-Healing Procedures

In this paper we extend the PHS presented in [Bromuri et al., 2011a,b, Krampf et al., 2011, Bromuri et al., 2010a] with self-healing procedures. The reason to perform this extension resides principally in the fact that a PHS should be almost constantly available to the patients accessing it. With the particular case of patients affected by diabetes, gestational diabetes or cardiovascular diseases, a downtime of several hours can have a serious impact on the effectivity of a treatment based on telemedicine. For example, if the downtime lasts hours, a patient with gestational diabetes may enter in a phase of poor glycemic control and consequently the baby may experience macrosomia, defeating the whole purpose
of a telemedicine approach based on a PHS.

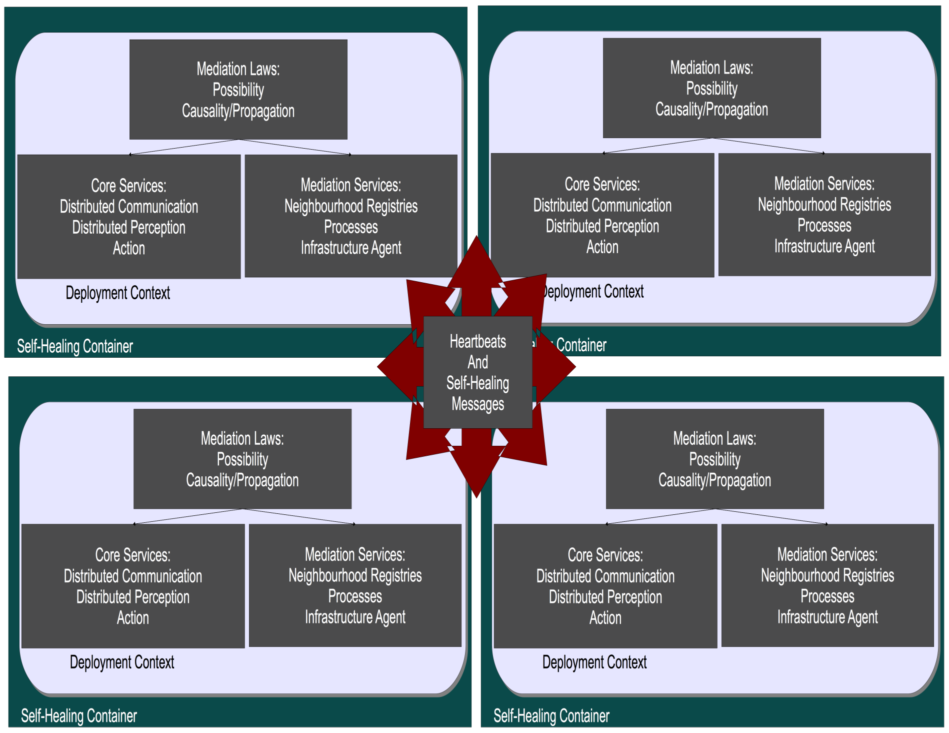


Figure 4: Extensions to the System Architecture.

To extend our system with self-healing procedures, we introduced an infrastructure agent in every container of the distributed agent environment. Fig. 4 shows the details of the conceptual architecture of the GOLEM containers extended with the self-healing functionalities in relationship to their distributed topology.
The infrastructure agents are capable to observe and perceive changes in the topology of the agent environment due to a failure and they are capable to devise simple plans to coordinate with neighbour agents in order to cover for the fault of one or more containers. The plans are instantiated and executed in parallel, to minimise the downtime, while the communication process taking place between the agents allows to minimise the excessive expansion of the cells ensuring that
the workload is distributed amongst the neighbours of the dead cells. 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. The rationale is a system that fulfills the requirements of self-healing systems: our system is adaptable to changes in the topology, it is observable as we are using a declarative approach to describe our entities, it is autonomous as we are utilising agents to deal with the failures and it is aware as we are using agents that plan and coordinate in a distributed environment. To handle a big number of connections per area, we further extended the GOLEM with load balancing features as shown in Fig. 5.

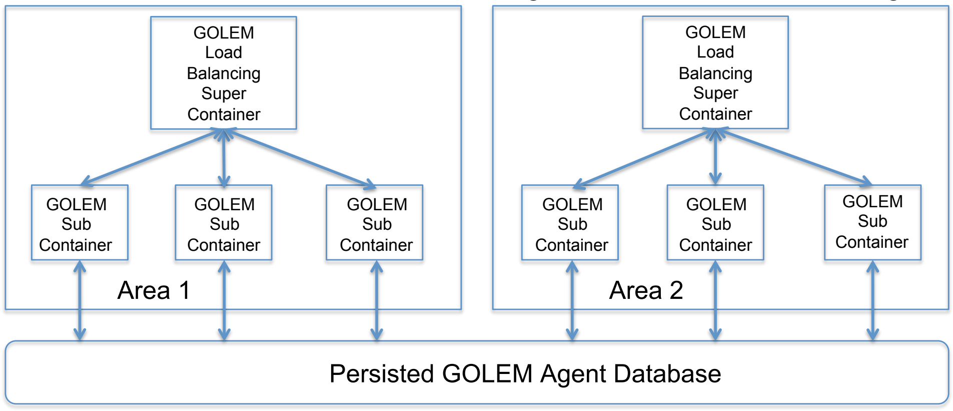
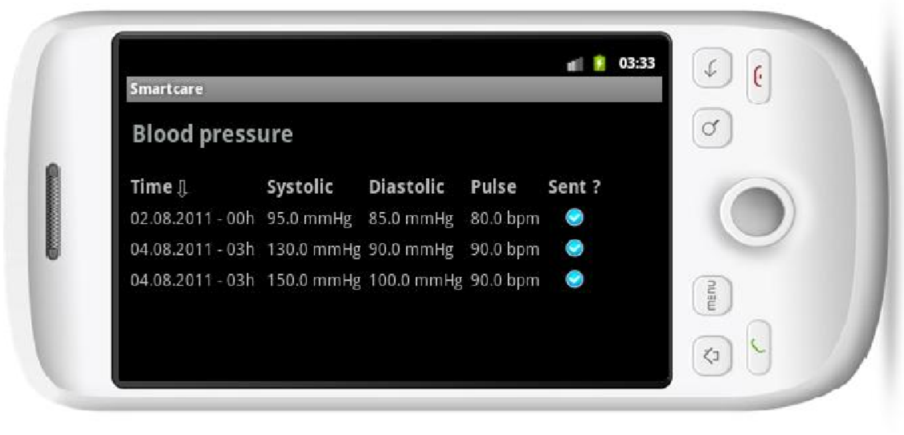


Figure 5: Load Balancing with GOLEM Containers.

Such an architecture allows a super-container to split the loads incoming from multiple patient's requests in sub-containers. The sub-containers have then access to a resource which is a NO-SQL database containing the monitoring agents for the Personal Health System shown in Fig. 1. Such a resource is available to the agents in the agent environment in form of a GOLEM object which allows to retrieve serialized agents, put them into execution when the patient requires their services

3.1 Physiological Values Storage in the Smartphone

The patient's smartphone plays a role to ensure that the data are not lost when dealing with the self-healing agent environment. All physical values and symptoms are acquired though a mobile application (see Fig. 6).



Blood pressure

Time ⇩	Systolic	Diastolic	Pulse	Sent ?
02.08.2011 - 00h	95.0 mmHg	85.0 mmHg	80.0 bpm	✓
04.08.2011 - 03h	130.0 mmHg	90.0 mmHg	90.0 bpm	✓
04.08.2011 - 03h	150.0 mmHg	100.0 mmHg	90.0 bpm	✓

Figure 6: The Patient's Mobile Interface.

Typically, the patient can insert up to six times per day the glucose values,
twice per day the blood pressure and pulse values, as many times per day as preferred the symptoms experienced and the weight once per week. Current smartphone technology is capable of storing several gigabytes of data, consequently there is not a storage problem in the mobile phone.
As a consequence, we utilize the patient's smartphone capabilities to store
information by modelling a SQLite database for the physiological values and symptoms of the patients. This allows our patients to work on the smartphone also when there is not an active internet connection or when the agent environment is down due to maintenance issues. As shown in Fig. 7 the smartphone implements a protocol of interaction with the agent environment where the physiological values of the patient are deleted from the smartphone only when the agent environment acknowledge the reception of the data, meaning that the data has been successfully stored in the Patient Management System.

3.2 The Containers Behaviour

GOLEM containers are programmed declaratively, using the Ambient Event Calculus (AEC) [Bromuri and Stathis, 2009] formalism, an extension of the Event Calculus (EC) [Kowalski and Sergot, 1986], that handles the evolution of C-logic.

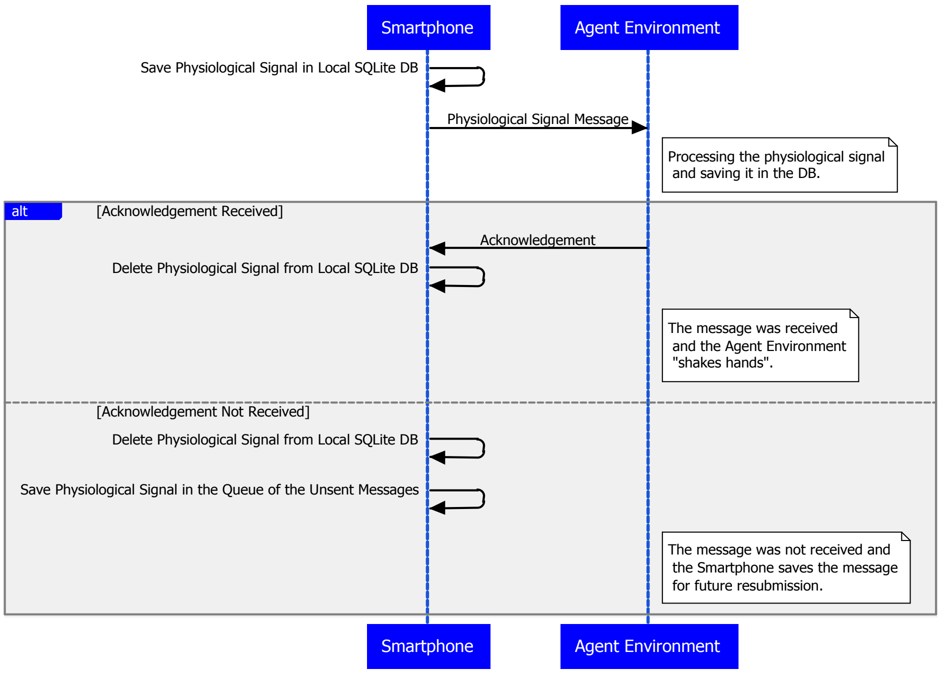

Smartphone
Agent Environment
Save Physiological Signal in Local SQLite DB
Physiological Signal Message
Processing the physiological signal and saving it in the DB.
alt
[Acknowledgement Received]
Acknowledgement
Delete Physiological Signal from Local SQLite DB
The message was received and the Agent Environment "shakes hands".
[Acknowledgement Not Received]
Delete Physiological Signal from Local SQLite DB
Save Physiological Signal in the Queue of the Unsent Messages
The message was not received and the Smartphone saves the message for future resubmission.
Smartphone
Agent Environment

Figure 7: The Smartphone Interaction with the Agent Environment.

structures by means of events in distributed containers. GOLEM agents use a subscriber/publisher pattern for sensing the events happening in the containers.
We extended the behaviour of the GOLEM containers with the **propagate_at/2** predicate to handle the dispatching and propagation of events produced by agents in the distributed settings. The infrastructure agents become subscribers and publishers of ack and inform events, while the events dispatching is handled using the **propagate_at/2** predicates. The **propagate_at/2** predicate is specified as follows:

```
propagate_at(ack:Event,T) ←
            happens(Event,T), Event[actor ) Agent, receivers ) Containers, known neighbours ⇒  NeighbourList],
            foreach(member(X,Containers),holds_at(X,neighbour,this,T)).

propagate_at(inform:Event, T) ←
            Event[actor ⇒ Agent, receivers ⇒SubList, cover ⇒CBroken, randomvalue ⇒ Diceroll],
            Holds_at(this,neighbourhood list, List,T), subset(List,SubList),
            foreach(member(X,SubList), holds at(X,neighbour, CBroken,T)).
            initiates(Cid,container,neighbour, Origin, Ev) ,
            cover:Ev[actor ⇒ Agent[container ⇒ Origin], container ⇒ CBroken]
            time(Ev,T), holds_at(CBroken,neighbour,this,T).

initiates(Cid, container,state, down, Ev) ) ←  notify_failure:Ev[actor ⇒ Agent, down ⇒  Cid].
```

where the **happens/2** is an AEC predicate stating that an event has happened in the agent environment and **holds_at/5** is an AEC predicate that provides an attribute value given an entity identifier (in this case this represents the current container), the attribute name (in this case neighbourhood list) and the time. The first rule mediates an **ack** event produced by an agent during the PHS normal behaviour and it states that whenever such an event happens, then this is propagated to all the receivers in the Containers list. Inside

the ack message, there is also the known neighbour containers list at a given time, so that every agent in the distributed topology can have knowledge of the neighbours of their direct neighbours. This is similar to the successor list approach of the CHORD P2P algorithm [Stoica et al., 2003], where given a successors list of lenght **r** , and a probability of disruption **p** for a single node, then the probability of disruption for the CHORD ring is **p^r**, meaning that the ring resilience can be improved by increasing the successors list length. In our case, the probability that the PHSs cannot restore the area covered by a node is **p^8**, when keeping a list of neighbours of neighbours in a grid like topology, where if needed the resiliency of our PHS can be improved by increasing the neighbourhood knowledge.
The second rule mediates disruptions happening in the distributed settings.
Once an agent does not perceive ack events from a neighbour container, the agent sends an inform event to all the containers that have a neighbouring relationship with the unresponsive container and it starts the healing procedure that we will discuss later.
The containers behaviour has been modified with additional predicates to update the neighbours list, with the third and fourth rules specified as initiates/5 predicates, which respectively state that the current container is neighbour of another container if its infrastructure agent is covering for a dead neighbour and that when an agent does not receive an inform event from a neighbour container at the right interval of time the neighbour is considered to be down when a notify failure event is produced by the infrastructure agent. For the moment, the tasks of joining a network and redeploying a failed container are handled by a human actor who needs only to propagate manually one single message to the neighbourhood of the failed container in order to reset the network to its initial behaviour.

3.3 The Infrastructure Agents

A GOLEM agent consists of a declarative module embedded in an agent body, which is situated in a container to perceive the events happening in it. The infrastructure agent cognitive model is based on two cycles, one to process the events sensed by the body and one to plan and act in the environment. The pseudo code for the two agent mind cycles is reported in Figure 8 (CSP stands for Conditional-STRIPS-Planner, an extension of the STRIP planner [Fikes and Nilsson, 1971] to handle conditional plans).

```
procedure PERCEPTION-Cycle($time$)
    static:KB; PERCEPTION-QUEUE; $p_1, p_2 \dots p_k where \forall p_i, p_i \in Plans \subset KB$;
    $percept \leftarrow$ NEXT-PERCEPT(PERCEPTION-QUEUE, $time$);
    UPDATE-KB(KB, $percept$, $time$); NOW($timenew$), PERCEPTION-Cycle($timenew$)

procedure ACTING-Cycle(time)
    static:KB, a knowledge base; ACTION-QUEUE, a queue of actions accessible by the agent body;
        $p_1, p_2 \dots p_k, where \forall p_i, p_i \in Plans \subset KB$, a set of plans;
    $currentstate \leftarrow$ STATE-DESCRIPTION(KB,time); $goal \leftarrow$ NEXT-GOAL($currentstate$,$time$);
    if $\nexists p_i | p_i.goal = goal$ then $p_k \leftarrow$ CSP($currentstate$,$goal$); ADD(Plans, $p_k$);

    $p_{exec} \leftarrow$ NEXT-EXECUTABLE-PLAN(Plans,$time$),

    if($p_{exec} = nil$) then NOW(timenew); ACTING-Cycle(timenew);
    else $currentact = p_{exec}.nextact$,
        if(CONDITIONAL?($currentact$)) then
          if(CHECK-KB(KB,IF-PART[$currentact$]))
              then $p_{exec} \leftarrow$THEN-PART[$currentact$];
                           ADD(Plans, $p_{exec}$);
                           NOW($tnew$);
                           ACTING-Cycle($tnew$);
          else $p_{exec} \leftarrow$ ELSE-PART[$currentact$];
                           ADD(Plans, $p_{exec}$);
                           NOW($tnew$);
                           ACTING-Cycle($tnew$);
        else ADD-ACTION(ACTION-QUEUE, $currentaction$);
                    NOW($tnew$);
                    ACTING-Cycle($tnew$);
```

Figure 8: The Agent Perception and Acting Cycles.

The **PERCEPTION-Cycle/1** reads the **PERCEPTION-QUEUE/1** for percepts coming from the environment. Such percepts are used to update the knowledge base KB about the state of the containers that are known in the agent environment. The **PERCEPTION-Cycle/1** is thus a passive cycle which only reads events coming from the environment. A plan is represented in the agent mind as a C-logic object. For example, the following plan expressed in AEC:

```
plan:p1[goal ⇒ cover:g1[container⇒ c2], diceroll ) 3000,next action ⇒ ac1, delay_action ⇒ 0,
sequence ⇒ {inform:ev4[cover ⇒ c2, diceroll ⇒ 3000],wait:ev5[delay⇒ 6], if_then_else:if1 }].

if_then_else:if1[if ⇒ check winner⇒ag1,p1,3000),
                then ⇒ {modify_topology:ev6[cover⇒ c2], end_plan:ev8}, else ⇒ { end_plan:ev9} ]

check_winner(A,P, Diceroll*) ← now(Time),
              not (holds_at(P,competitor, Comp,Time),
              holds_at(Comp, diceroll, Diceroll*,Time), Diceroll* >Diceroll).
```

specifies a plan **p1**, with a goal **g1** to cover for a container **c2**. In this plan the actions performed are: an inform event **ev4**, then a wait **ev5** action with a 6 seconds delay, to attend for messages incoming from other agents. These actions are pushed by the **ACTING-Cycle/1** in the **ACTION-QUEUE/2** of the agent body, that produces them in the container, which then mediates the actions according to the rules previously defined. As opposed to the **PERCEPTION-Cycle/1**, the **ACTING-Cycle/1** also produces events in the agent environment, conditionally to the state of the agent's knowledge base.
The inform event is sent to the neighbourhood of the dead container. Inside this event there is a random value diceroll that is used by the agents to compete for the coverage of

a container (in this case c2). After waiting for the delay, the planner finds an **if1** object as the next action to perform. This is a C-logic object representing an if-then-else structure, handled by the **CONDITIONAL?/1** predicate in the planner, that checks for the if condition **check_winner/3** in the **if-then-else** structure. The planner then executes either the **then** sequence of actions or the **else** sequence of actions according to the result of the **check_winner/3** condition. If agent **ag1** is the competition winner, it covers for the dead container and ends the plan, otherwise the plan is destroyed. Furthermore the **NEXT-EXECUTABLE-PLAN/2** predicate distinguishes between plans that have been frozen due to the introduction of a delay, and plans whose execution can continue, allowing for plans with different goals to be executed in parallel.
Fig. 9 shows, from left to right, a complete example on how the covering of dead containers happens in self-healing agent environment. The panel on the left in Fig. 9 shows a set of GOLEM containers represented by a different color. The central panel in Fig. 9 shows that four of containers have undergone a service disruption.
The panel on the right in Fig. 9 shows that the area of the containers that have undergone a service disruption have been covered by their neighbours. It is important to state that to perform the covering of the neighbours, the coordination takes place only amongst neighbour containers and not globally.
Despite the communication taking place between the agents, two or more containers may end up covering the same area. When the inconsistency is perceived, the interested agents start a resolution protocol similar to the one just discussed. The advantage in using a high level description in terms of plans rather than using reactive agents is that it allows us to have a more compact definition of the agent behaviour by making the plans structures that can be instantiated and destroyed according to the interaction state. Achieving the same result with reactive rules would require a more verbose and low level agent mind specification, that is difficult to debug when dealing with hundreds of agents in distributed settings.

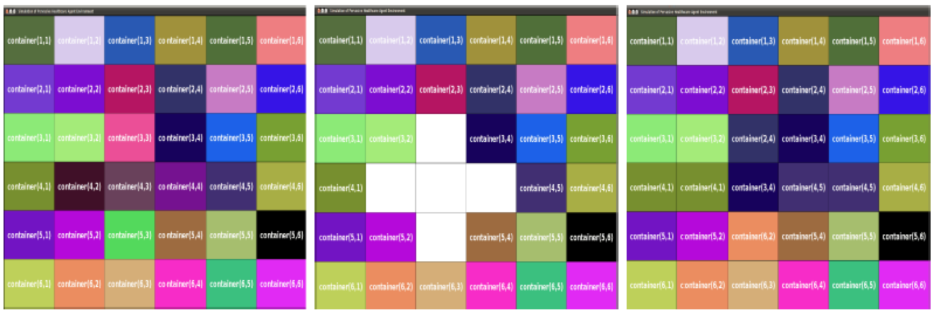


Figure 9: Simulation of Interaction of a 6X6 Pervasive Agent Environment with Service Disruptions.

4 Evaluation

In this Section we evaluate the behaviour of our self-healing agents. In particular we want to evaluate the time to recovery of our network and the messages exchanged by the various containers. Furthermore, we also need to evaluate how one or multiple containers would respond to the load of dealing with multiple patients at the same time. We evaluated our system by deploying a 10x10 grid of self-healing GOLEM containers. A

10x10 grid of GOLEM containers is usually enough to cover a large area of a city, where the load expected is around 4000-5000 patients.
To perform our evaluation, we used a dual core Intel Centrino 2, 2.66 Ghz per core with a total of 3Gb of RAM. We are aware that a PHS should be distributed on a grid and that deploying it on a single host gives a partial view of the real performances of the system.
In real settings we foresee that some of the containers will require more resources when representing hot spots (i.e. super markets or hospitals), while unpopulated areas will require less resources. For this purpose, as explained in Section 2, we enhanced GOLEM with load balancing techniques. Fig. 10 shows how the load is distributed amongst multiple GOLEM containers, distributed in different hosts within the same network, with a variable number of patients, going from 1 to 50, who experience a problem with their glucose levels, rising an alert of hypoglycemia to be reported to their doctors.

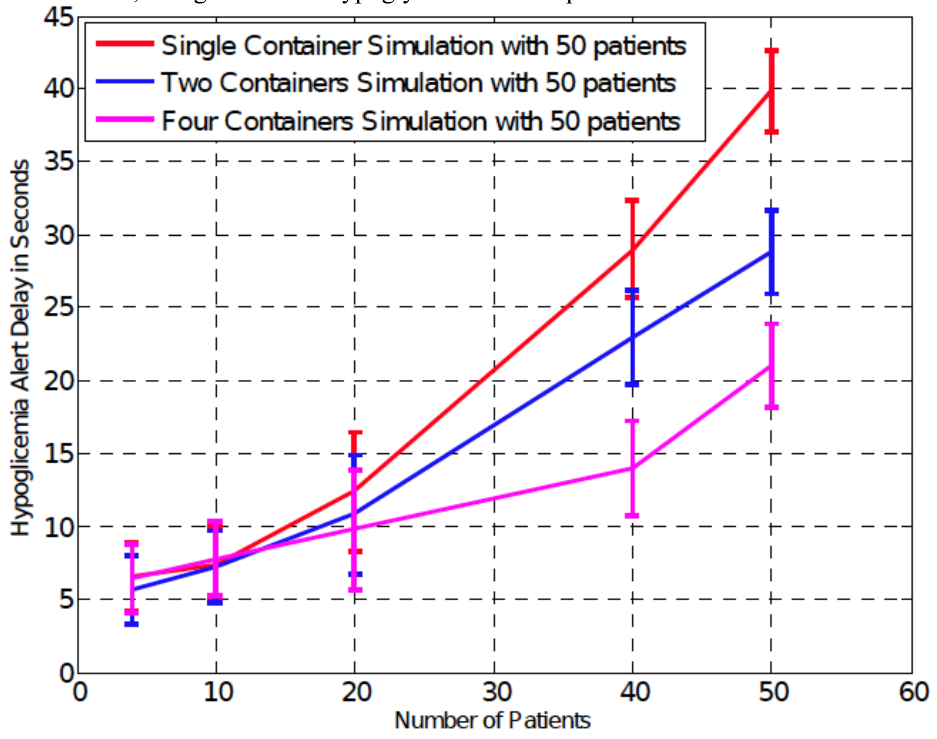
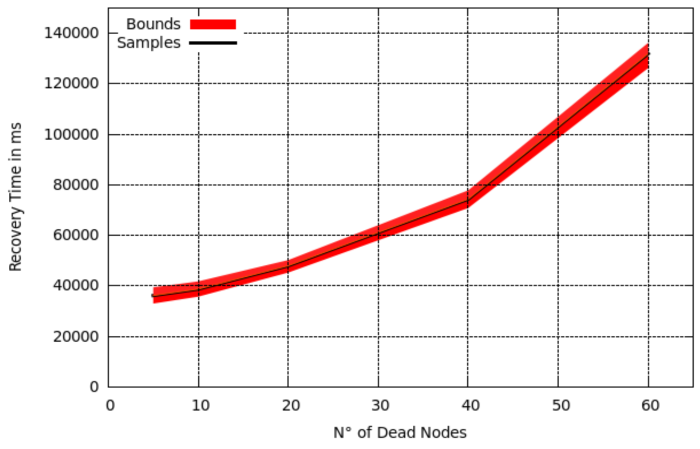


Figure 10: Load Balancing with Distributed GOLEM containers.

As shown in Fig. 10 the time required to answer to a situation of alert depends on the number of patients monitored by a GOLEM container. In particular, the introduction of a load balancing mechanism allows our container to avoid situations in which a container is so overloaded that it cannot answer a patient.
The tests showing the time to recovery and the number of messages exchanged by the containers for the recovery is shown in Fig. 11. When producing the curves in Fig. 11 we made the following assumptions: a) since the system is deployed in a single host, the delays of a real network are not taken into consideration b) we assumed that the containers die all at the same time, that is a pessimistic assumption as discussed in Section 3.2; c) the system presents failures after every

containers had the time to learn about its neighbours. The curves in Fig. 11



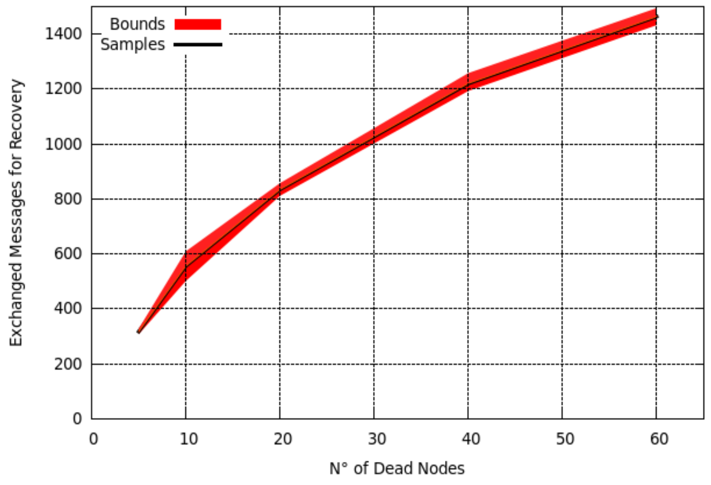


Figure 11: Performance Evaluation of a 10x10 Grid Pervasive Agent Environment.

evaluate respectively the number of messages to recover from a failure and the total downtime with respect to the number of dead containers. These are two critical parameters: if too many messages are exchanged this can impact the PHS performances, and if there are long downtimes, emergencies may happen when the system is unavailable.

Fig. 11 on the top shows the number of dead containers with respect to the number of messages that the agents need to exchange to cover for the missing containers. The curve behaves logarithmically because the more nodes in a neighbourhood die, the less nodes take part to the competition for covering a dead node, producing less messages. Consequently, the curve on the bottom of Fig. 11 behaves like a quadratic curve as the alive infrastructure agents have to execute more plans to cover a bigger area. The introduction of parallel plans helps agents to minimise the downtime as whenever a dead neighbour is detected, a new plan is instantiated to cover it. Finally, it is important to state that the network is going to recover completely only in the case in which the neighbourhood is not completly disrupted. Usually, if more than 60% of the network is down, only a partial recovery is possible as the containers may not know enough dead

neighbours to cover for all of them.
Also, introducing agent communication, mediated by the agent environment in terms of the propagation rules previously presented, allows to minimise a) the number of messages exchanged in the environment b) the uncontrolled growth of the area controlled by a container and c) the conflicts arising in the healing protocol.

5 Related Work

The personal health systems and the pervasive health care field are relatively new, consequently there are not too many attempts to deal with self-healing problems.
Nevertheless, there have been already attempts to deal with the problem of self-healing a PHS when a disruption occurs, by using standard approaches to fault tolerance, as opposed to our approach based on planning and agent environments.
In [Liao et al., 2008] Cheng-Feng et al. presented a self-healing and self-organising protocol for smart houses. Cheng-Feng et al. propose a message-based pervasive middleware where the nodes represent devices within the smart house and are atomic components of the system. The nodes in the system then can constitute pervasive communities. The failure detection in this system is done by means of heartbeat messages submitted by the devices to a Pervasive Service Manager that checks periodically if they are alive.
Our approach differs from the one presented by Cheng-Feng et al. as we introduce the concept of agent environment to deal with the failure of the system rather than having a centralised entity and that we assume that agents in the distributed environment can reason about its topology and coordinate to heal it.
Schaeffer-Filho et al. [Schaeffer-Filho et al., 2009] define the concept of Self Managed Cell (SMC) as a recursive structure that goes from the body-area network for health monitoring of the patient to the SMC to handle the household of the patient to the SMC of the healthcare professionals in charge of the patient. The Body Area Network (BAN) is modelled as a virtual complex node that abstracts a set of sensors and publish events in the form of health records in the upper level SMCs. The upper level SMCs are controlled by doctors in charge of the patients that can use the SMC to check for the physiological signals of the patient.
From a certain perspective we can relate the concept of SMC to the concept of agent environment, although the SMCs are not thought to work in indoor environments. SMCs also define policies to handle the events happening in the environment and produce further events with a declarative approach that is similar to ours. The main difference between our work and SMCs is that we have cognitive agents reasoning about the properties of the agent environment, which allows us to have the different cells to coordinate when a disruption happens, while in the case of SMCs, the issue is left to the single SMC policies.
From the stand point of self-healing systems, in [George et al., 2002] 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 [George et al., 2002], 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 [Kondacs, 2003] is another example on how using bioinspired 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, despite having cells in the system dieing at a high rate, the system is capable to self-regenerate the missing part. Similarly to Kondcas, 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 [Haesevoets et al., 2009] 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 [Haesevoets et al., 2009], we separate the concerns of handling self-adaptation between the agent environment and the agents, but differently from [Haesevoets et al., 2009], 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 reorganize the agent environment to self-heal from a fault of one or more of the containers composing it. We utilise planning agents that can reason in parallel about multiple faults and that produce plans and interact to cover for missing containers in the environment. The novelty of the approach resides in using planning agents combined with a complex declarative agent environment that simplifies the interaction between the agents controlling the distributed system. We have evaluated the system from the point of view of the downtime and number of message exchanges required to recover from a growing number of dead containers, discovering that the system scales up and it can recover in useful time even when more than 50% of the system is down. 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.

Acknowledgements
This work has been partially funded by the Hasler Stiftung grant MONDAINE, by the Nano-Tera proeject G-DEMANDE and it was partially supported by the FP7 287841 COMMODITY12 project.

References

Stefano Bromuri and Kostas 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.

Stefano Bromuri, Michael Ignaz Schumacher, and Kostas Stathis. Towards distributed agent environments for pervasive healthcare. In Jurgen Dix and Cees Witteveen, editors, MATES, volume 6251 of Lecture Notes in Computer Science, pages 125-137. Springer, 2010a. ISBN 978-3-642-16177-3.

Stefano Bromuri, Visara Urovi, and Kostas Stathis. iCampus: A Connected Campus in the Ambient Event Calculus. International Journal of Ambient Computing and Intelligence, 2(1):59-65, 2010b.

Stefano Bromuri, Michael Schumacher, and Kostas Stathis. Pervasive healthcare using self-healing agent environments. In 9th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'11), Advances in Intelligent and Soft-Computing. Springer Verlag, 2011a.

Stefano Bromuri, Michael Schumacher, Kostas Stathis, and Juan Ruiz. Monitoring gestational diabetes mellitus with cognitive agents and agent environments. In Proceedings of the 2011th IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2011), August 2011b.

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. ISBN 0-89791-308-6.

Richard Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In IJCAI, pages 608-620, 1971.

Felix C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Comput. Surv., 31:1-26, March 1999. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/311531.311532. URL http://doi.acm.org/10.1145/311531.311532.

Selvin George, David Evans, and Lance 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. ISBN 1-58113-609-9. doi: http://doi.acm.org/10.1145/582128.582149.

Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems - survey and synthesis. Decision Support Systems, 42(4):2164-2185, 2007. ISSN 0167-9236. doi: DOI: 10.1016/j.dss.2006.06.011. Decision Support Systems in Emerging Economies.

Robrecht Haesevoets, Danny Weyns, Tom Holvoet, and Wouter Joosen. A formal model for self-adaptive and self-healing organizations. Software Engineering for Adaptive and Self-Managing Systems, International Workshop on, 0:116-125, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/SEAMS.2009.5069080.

Attila 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.

R Kowalski and M Sergot. A logic-based calculus of events. New Gen. Comput., 4(1): 67-95, 1986. ISSN 0288-3635. doi: http://dx.doi.org/10.1007/BF03037383.

Johannes Krampf, Stefano Bromuri, Michael Schumacher, and Juan Ruiz. An agent based pervasive healthcare system: a first scalability study. In Proceedings of the 4th ICST International Conference on eHealth (eHealth 2011), ICST Lecture Notes (LNICST). Springer Verlag, November 2011.

Chun-Feng Liao, Ya-Wen Jong, and Li-Chen Fu. Psmp: A fast self-healing and self-organizing pervasive service management protocol for smart home environments. In APSCC, pages 574-579, 2008.

Marija Mikic-Rakic, Nikunj Mehta, and Nenad Medvidovic. Architectural style requirements for self-healing systems. In Proceedings of the first workshop on Self-healing systems, WOSS '02, pages 49-54, 2002. ISBN 1-58113-609-9.

Alberto Schaeffer-Filho, Emil Lupu, and Morris Sloman. Realising management and composition of self-managed cells in pervasive healthcare. pages 1-8, Apr 2009. doi: 10.4108/ICST.PERVASIVEHEALTH2009.5979.

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw., 11:17-32, February 2003. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2002.808407. URL http://dx.doi.org/10.1109/TNET.2002.808407.

Visara Urovi, Stefano Bromuri, Kostas Stathis, and Alexander Artikis. Towards runtime support for norm-governed multi-agent systems. In KR, 2010.

Upkar Varshney. Pervasive Healthcare Computing: EMR/EHR, Wireless and Health Monitoring. Springer Publishing Company, Incorporated, 2009. ISBN 1441902147, 9781441902146.

Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems, 14(1):5-30, 2007.

M. Wooldridge. MultiAgent Systems. John Wiley and Sons, 2002. ISBN ISBN 0 47149691X.