The role of indirect communication in emerging collective behaviours

Dimitrios Tsiridis[1] , Ioannis D. Zaharakis[2], and Achilles D. Kameas[1,2], *Member, IEEE*

***Abstract*--This work introduces a reactive agent architecture based on the subsumption scheme incorporating communication abilities as a basic behaviour block. The aim is to study and evaluate the emerging collective behaviours and the role of indirect communication in reactive agent societies situated in highly dynamic environments. A series of experiments are presented, focusing at the optimisation of the basic behaviours synthesis and the exploitation of the indirect communication in collectively performed tasks. The experiments are conducted with the simulation of swarm systems applied to a real-life scenario of the industrial segment.**

***Index terms*-- emergent behaviours, indirect communication, reactive agents, subsumption architecture.**

I. Introduction

Swarm Intelligence (SI) offers an alternative way of designing intelligent systems, in which autonomy, emergence, and distributed functioning replace control, pre-programming, and centralization. SI is defined as the emergent collective intelligence of groups of simple agents, or in more detail, SI is the property of a system whereby the collective behaviours of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge [1]. The SI approach emphasizes parallelism, distributedness, and exploitation of local interactions among relatively simple agents and the environment they belong to.

Communication is one of the most common means of interaction among intelligent agents. Direct communication is a purely communicative act, one with the sole purpose of transmitting information, such as a speech act, or a transmission of a radio message. Although direct communication is a natural way to communicate, it needs specific mechanisms and protocols to send, receive, and interpret any signals or messages, and in most cases it needs increased computational resources. On the other hand, in natural systems as insect societies any observable behaviour and its consequences can be interpreted as a form of communication, and could be considered as an indirect communication act. This type of communication is referred to as stigmergic in biology literature, as it refers to communication by altering the state of the environment in a way that will affect the behaviours of the others for whom the environment is a stimulus [4]. This means that the environment serves as a medium of communication.

This type of communication could be applied to an artificial swarm system consisting of micro scale, unsophisticated agents with limited computational capabilities. As Bonabeau et al. notice in [1], stigmergy contributes in a number of ways:

- In problem solving where the replacement of coordination through direct communications by indirect interactions helps the design of simple agents and reduced communication among agents.
- In incremental construction and optimization where a new solution is constructed from previous solutions.
- In flexibility where, because the changes of the environment are treated in a similar way no matter whether they come from physical or communication actions, the agents need not be reprogrammed to deal with any particular change.

The basic factor that influences the complexity of an agent is the type of the environment on which it is situated and interacts with in order to achieve its objectives. According to [5] the most complex environment is the one that is characterised as inaccessible, non-deterministic, non-episodic, dynamic, and continuous.

The agent architecture is a fundamental issue as its main role is to provide the necessary interaction between an agent and its environment: receive the environment sensory perception input and pass it to the agent's program, execute the program and activate the appropriate effectors to change the environment. Because the symbolic approach of agents in classic AI wasn't adequate when they had to operate in highly environments with time restrictions, the nouvelle AI researchers proposed alternative approaches for the construction of agents that respond in a timely fashion to their environment changes, so as their actions to have the expected results. These unsophisticated agents are referred to the literature as simple reflex agents [5], or purely reactive agents [7], thought the latter are more hardware oriented.

The "subsumption architecture" of reactive agents that was introduced by R. Brooks [2] [2], did not adopt symbolic representation, nor reasoning mechanisms. The two main characteristics of this architecture are: a) an agent's decision making procedure is realized by a set of task accomplishing behaviours, each one can be considered as an action function and b) more than one behaviours can be triggered simultaneously, so a selection mechanism of multiple action proposals must be present. Brooks proposed the arrangement of behaviours in a subsumption hierarchy, where

[1] Hellenic Open University, 23 Sahtouri str, 26222, Patras, Hellas email: tsir@otenet.gr, dimitris.tsiridis@geniki.gr.
[2] Research Academic Computer Technology Institute, Uni-versity Campus, N. Kazatzaki str, 26500, Patras, Hellas, email: {jzaharak, kameas}@cti.gr

lower behaviours in hierarchy can suppress the higher ones. The advantages of reactive architectures are: simplicity, computational tractability, economy, robustness against failure and elegance [7].

The aim of this work is to study and evaluate the role of indirect communication (a kind of stigmergy) in the performance of reactive agent societies. As the computational resources of such type of agents are limited, the overhead of the communication capability is a critical issue. So, we introduce a subsumption agent architecture that embodies the communication capability as a basic behaviour allowing the synthesis of complex behaviours and requires no additional mechanisms or protocols; instead, it deals with communication signals as changes of the environment the agent is situated in. A performance comparison between agent societies with and without communication ability is presented. Additionally, issues and situations that boost or reduce the performance of the indirect communication approach are noted. This is accomplished through several test case simulations for both approaches on a simulated multi agent environment.

The next section introduces the case study we were based upon to develop our work and the approach we adopted. The section III describes the simulation environment with brief reference to user interface parameterization, including the agent behaviours, indirect communication protocol, agent's architecture, the simulation branch appearance description, with a special reference to important functioning details. The section IV details on the experiment scenarios, evaluation criteria and the results taken by relevant evaluation test cases. Finally, the section V discusses the results and the conclusions derived by the experiments and sets new issues for further investigation

II. Case Study

In order to study, evaluate and exploit the collective phenomena which emerge from local actions of large groups of elementary agents activated in non-deterministic and dynamic environments, we dealt with the following subject [6]:

"A society consisted of enough population of elementary autonomous robotic agents is activated in a dynamic environment having as an objective the diagnosis and repair some problematic situations. The agents have specific and limited capabilities of perceptions and actions, low reasoning calculating resources and memory. More specifically they perceive limited number of environment stimuli and react in predetermined way to that. Their capabilities coded by a total of basic behaviours. They do not communicate directly, but a potential collective action emerges through local interaction with the rest members of community."

The main objectives include the following:

- The basic behaviours encoding and their integration in a suitable architecture schema;
- The creation of an indirect communication mechanism;
- The development of a simulation environment (implemented using Java3D platform);
- The study of the emerging collective phenomena through a set of experiments; and
- The comparative evaluation of the society performance using test case scenarios.

To fulfil these objectives we dealt with a problem of the real world based on a hypothetic scenario [6]:

"In the process of secondary production of petroleum products, pipes are used for the regulation of level of water as well as a pumping system, using heater treatment units to separate the two liquids (water & petroleum). The technological problem it faces is the salt sediments which are accumulated on the pipe walls (Fig. 1) and they might cause obstructions and damages on the pumping system."

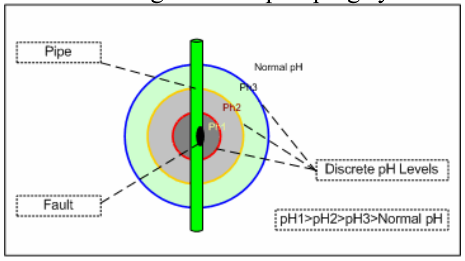


Figure 1: Fault Creation

The approach we adopted from Software Engineering point of view is based on "Swarm Intelligence" approach of emergent "social" intelligence and from Technical point of view based on multi-agent systems and the subsumption agent architecture. The reasons we adopted this approach are: simplicity, economy, robustness against failure, elegance, impressive results in many applications and suitability for dynamic & non-deterministic environments.

A solution to the case study we described is to place a large enough robotic homogenous agent population into the fluid environment: tiny sized agents, with low consumption and computational power and naturally low cost. Each agent is supplied with a perception system (infrared, pH changes, and static and moving obstacles recognition sensors), reactive subsumption architecture (with a basic behaviour hierarchy), a small memory and an actuation system (submarine navigation, transceiver system and a chemical compound load).

The agents are moving randomly in the fluid environment sensing the pH variations by their sensors trying to find pH disturbances, to reach stochastically the faults, dock and repair them. The environment can be supplied with refuel stations so as the agents with low vital status can be refuelled. The efficiency of system will be evaluated with or without indirect communication

III. Agent Architecture and Simulation Environment

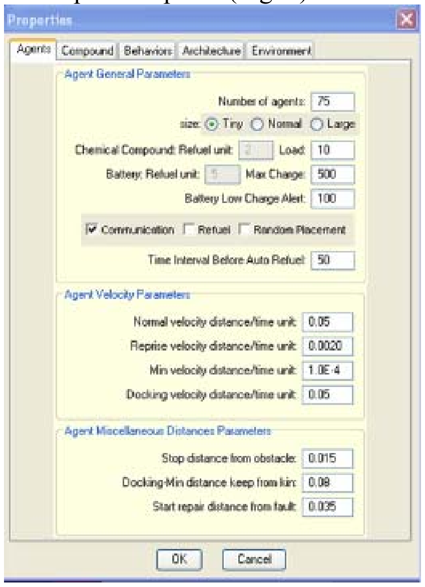
For the purposes of this study, we designed and developed a standalone multi-agent simulation application running under Windows systems, using GUI for user interface and Java3D platform for the simulation environment.

We analyzed and coded basic and complex behaviours and arranged them in a subsumption architecture scheme. We also implemented a realistic resource-bounded communication protocol. The main frame of simulation system (Fig. 7) consist of a set of buttons to start, stop a simulation, set the system properties and simulation scene appearance and a set of simulation statistics on a vertical bar on the left side of the main frame and the 3D simulation scene on the main body. Also we give a brief reference to interesting functioning details.

The user interface we developed covers a great deal of simulation parameterization concerning:

- Agents: population, agent's size, chemical compound load, battery charge, communication, etc.
- Compound: Fault sphere (range, fault units), Signal Sphere (range, life, hop threshold) and Station (range, known or unknown location indicator, etc) properties
- Behaviour properties: enabling distances, etc.
- Architecture construction (Fig. 5).
- Environment related information: tank & pipe dimensions and volume, free space, faults and signals environment percentage of occupation, etc.

All these parameters can be set in the corresponding tapped panes in the "Properties" panel (Fig. 2).


Properties

Agents | Compound | Behaviors | Architecture | Environment

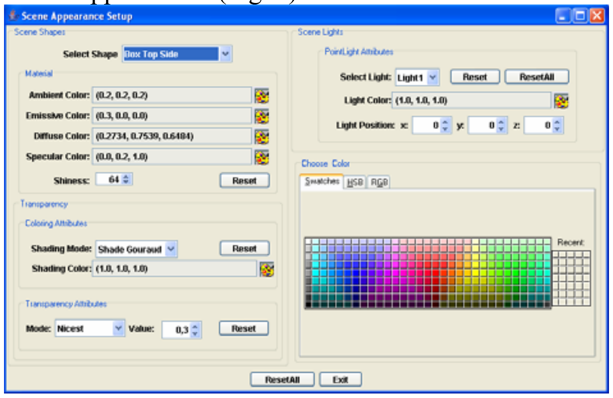
Agent General Parameters
Number of agents: 75
size: [✓ Tiny] Normal Large
Chemical Compound: Refuel unit: 2 Load: 10
Battery: Refuel unit: 5 Max Charge: 500
Battery Low Charge Alert: 100
[✓ Communication] Refuel Random Placement
Time Interval Before Auto Refuel: 50

Agent Velocity Parameters
Normal velocity distance/time unit: 0.05
Reprise velocity distance/time unit: 0.0020
Min velocity distance/time unit: 1.0E-4
Docking velocity distance/time unit: 0.05

Agent Miscellaneous Distances Parameters
Stop distance from obstacle: 0.015
Docking-Min distance keep from kin: 0.08
Start repair distance from fault: 0.035

Figure 2: Agent properties tapped pane

Furthermore, all simulation default object attributes can be changed from the UI of the application, by pressing the button "Appearance" (Fig. 3).


Scene Appearance Setup

Scene Shapes
Select Shape: Box Top Side
Material
Ambient Color: (0.2, 0.2, 0.2)
Emissive Color: (0.3, 0.0, 0.0)
Diffuse Color: (0.2734, 0.7539, 0.6484)
Specular Color: (0.0, 0.2, 1.0)
Shiness: 64
Transparency
Coloring Attributes
Shading Mode: Shade Gouraud
Shading Color: (1.0, 1.0, 1.0)
Transparency Attributes
Mode: Nicest Value: 0,3

Scene Lights
PointLight Attributes
Select Light: Light1
Light Color: (1.0, 1.0, 1.0)
Light Position: x: 0 y: 0 z: 0

Figure 3: Simulation objects appearance change panel

A. *Behaviours*

Initially, we analyzed and coded the basic and complex behaviours of the individual agent. These are related to movement, task accomplishment, communication, and emergent behaviours.

1) *Movement behaviours*

These behaviours are responsible for the navigation of the agent either as an individual or as a member of a broader group and are the following.

Wander: No perception input needed. Calculate a random position to move in 3D space. It is activated if no other behaviour gives output (suppressed by all others). The velocity is calculated randomly between the min and max defined.

Avoid Obstacles: Allows agents to avoid collisions with static obstacles (walls or pipes). It is activated when static obstacles are sensed within behaviour enabling distance and gives as an output a turn upon nearest obstacle position to move. The velocity is adjusted depending of the distance from the nearest obstacle (slow down or accelerate).

Avoid Kin: Allows agents to avoid collisions with other agents. It is activated when nearby agents are sensed and are within behaviour enabling distance. The behaviour gives as an output a turn upon nearest kin position to move. The velocity is adjusted depending on the distance from the nearest agent (slow down or accelerate).

Follow Kin: Allows agents to follow other agents in the local surrounding. It is activated when nearby agents are sensed within behaviour enabling distance and the "eye" angle is less than 90^o. The behaviour gives as an output the next position to move. The velocity is adjusted trying to keep a safe distance from leader (slow down or accelerate).

Align: Allows agents to align their direction vector with other agents in their local environment. It gives as an output a direction vector which is the normalization of sum of all vectors of agents within the behaviour enabling distance:

$$\vec{V}' = normalize\left(\sum_{n=1}^{k} \vec{V}_n + \vec{V}, p\right). \tag{1}$$

Aggregate: Allows an agent to move to a direction so as to be part of a group. It gives as an output a position to move which is the centroid of positions of agents within the behaviour enabling distance:

$$local\ centroid = \frac{\sum_{n=1}^{k} p_k}{k} \tag{2}$$

The velocity is adjusted depending on the distance from the centroid (slow down while reaching the centroid).

Disperse: Allows agents to move to a direction so as groups are split and agents spread all over the environment. It's the opposite behaviour of aggregate and agents try to avoid the centroid. The velocity is adjusted depending on the distance from the centroid (accelerate while going away from centroid).

All these behaviours obey to the constraints that the position proposes to an agent to move:

- Must not exceed a turning angle (angle between current and new direction vector) of 45^o and
- The velocity distance (distance between current and new position to move) must not exceed the maximum velocity distance defined in UI.

2) *Task accomplishment behaviours*

These behaviours are related to the tasks an agent individual can perform and they are placed in a higher priority than the movement behaviours in the subsumption scheme, as they reflect the reason the agent has been constructed for as well as the "self-preservation instincts" of the agent.

Search Station: Activated only when an agent's vital status is low. Agent tries to find a refuel station. Refuel stations positions might be known or transmit a specific signal. Includes embedded behaviours:

Docking Align: Activated when a predefined distance from a station reached (if station positions are known) or a predefined intensity of station signal sensed (if station positions are unknown). It gives as an output a direction vector so as the agent to align its back vertically to the station, using the agent's current direction vector, the nearest obstacle position and station position (if known) or station signal (if unknown). A predefined maximum rotation angle per activation allowed.

Docking Approach: Activated when the docking alignment to a station completed. It gives as an output a position to move backwards, using the distance from nearest obstacle and the distance from the station (known position) or the station signal (unknown position) until the final dock.

Refuel: When the docking procedure is completed, the agent starts to recharge its batteries and to reload with the chemical compound it carries for repairing. This behaviour gives output until batteries are fully recharged and chemical compound is fully reloaded.

Search Fault: Activated when pH disturbance is sensed. The agent tries to move stochastically to the fault by sensing the pH variations. If the current pH value is greater than previous one sensed (means that an agent is closer to a fault) then the agent maintains course otherwise it tries to return to previous position.

Docking Align: Activated when a predefined pH disturbance is sensed. It gives as an output a direction vector, so as the agent to align its back vertically to the fault, using the agent's current direction vector and the nearest obstacle position. A predefined maximum rotation angle per activation allowed.

Docking Approach: Activated when the docking alignment to a fault completed. It gives as an output a position to move backwards, using the pH variations sensed and the distance from the nearest obstacle until a predefined minimum dock distance from nearest obstacle reached.

Repair: When the docking procedure is completed, the agent starts to release a chemical substance which decomposes the salt sediments. This behaviour gives output until a fault completely repaired or the agent's load becomes empty.

3) *Indirect communication behaviour*

This type of behaviour is essentially a movement behaviour; however, it is triggered by different type of input signals, i.e., by signals perceived be the RF sensors.

Stigmergic Move. It is activated when a fault reported signal is sensed by an agent and gives as an output the position to move so as the agent will move towards the most important signal in its effort to reach faults.

4) *Complex behaviour*

This behaviour is the result of the synthesis of other basic behaviours (Fig. 6) and it is placed in the highest priority in the subsumption scheme, as it is responsible for the agent safety during navigation.

Safe Wander: It is a complex behaviour which combines the basic behaviours *Avoid Obstacles* and *Avoid Kin*. If both of them give output then *Safe Wander* gives as an output a turn upon the centroid of outputs position to move.

B. Indirect communication protocol

We designed an algorithm and implemented a non-demanding indirect communication protocol due to agent resource limitations. This protocol is based on the wireless protocol IEEE 802.15.4 (ZigBee). In our model we consider that an agent may be in one of three distinct states (Fig. 4):

- ***Wandering***: An agent is wandering (not with the mean of the *Wander* behaviour) in the environment without sensing any pH disturbance.
- ***Fault_Sensed or Station_Sensed***: An agent has sensed a fault via its pH sensors or a station signal via its transceiver.
- ***Fault_Reported or Station_Reported***: An agent has sensed via its transceiver a fault or station reported signal.

Each signal has the following parameters:

- ***HopCount***: Indicates the importance of the signal. A zero value indicates that an agent directly sensed a fault or station; values so indicate how far an agent is from a fault or station and
- ***HopThreshold***: An agent which receives a signal with *HopCount* value greater than *HopThreshold* value, does not retransmit the signal (namely the maximum level of signal retransmission).

We have distinguished two cases depending on the vital status of the agents. An agent's vital status is "Active" when its battery charge is above "low battery alert" and has enough chemical substance cargo, otherwise is "Hungry".

Case 1: The vital status of the agent is "Active"

Initially an agent is in the *Wandering* state. If it directly senses a fault, then enters in *Fault_Sensed* state and starts to transmit signal with *HopCount=0*. If it stops sensing a fault, then it returns to *Wandering* state. If it senses fault reporting signals then enters in *Fault_Reported* state and tries to follow the most important signal (with the less *HopCount* and if there are more than one, the one with the highest intensity) while retransmits it increasing the received *HopCount* by one. If while it is in *Fault_Reported* state and directly senses a fault, then enters in *Fault_Sensed* state and if it stops sensing any fault reported signals returns to *Wandering* state.

The *Fault_Sensed* state is implemented by the *SearchFault* behaviour, which when it gives output (the agent sensed fault) except the position to move propose the agent to transmit a *Fault_Sensed* signal (*HopCount*=0).

The *Fault_Reported* state is implemented by the *StigmergicMove* behaviour, which when it gives output (the agent sensed fault reported signal) except the position to move propose the agent to retransmit the *Fault_Reported* signal received increasing its *HopCount* by one, only if the received signal *HopCount* is less or equal from *HopThreshold*.

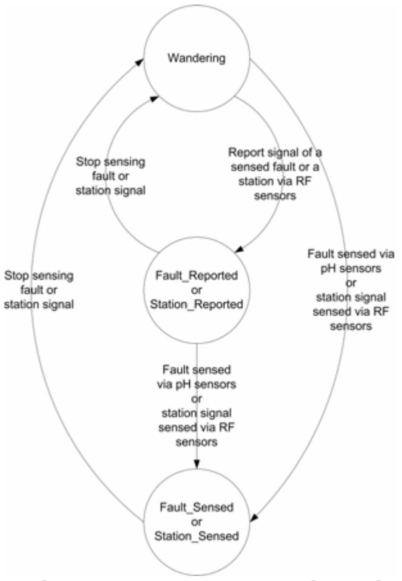

Wandering
Stop sensing fault or station signal
Report signal of a sensed fault or a station via RF sensors
Stop sensing fault or station signal
Fault_Reported or Station_Reported
Fault sensed via pH sensors or station signal sensed via RF sensors
Fault sensed via pH sensors or station signal sensed via RF sensors
Fault_Sensed or Station_Sensed

Figure 4: Indirect communication protocol state diagram

The *SearchFault* behaviour will be higher in the subsumption architecture, so as to suppress the *StigmergicMove* behaviour, because *Fault_Sensed* state is more important than *Fault_Reported*.

When an agent's vital status is "Active", it ignores all the possible station signals may perceive (*Station_Sensed, Station_Reported*).

Communication algorithm of *SearchFault* behaviour:

```
SearchFault:
    if (inpHValue > NORMAL_WATER_pH) {
        …
        signal.setType() = Fault_Signal
        signal.setHop() = 0
        RFTransmit(signal)
    }
```

Communication algorithm of *StigmergicMove* behaviour:

```
StigmergicMove:
    if (signals sensed) {
      Select most import Fault_Reported signal (less hop count
    and max intensity)
      …
      If (selectedSignal.getHop() <=
        HOP_THRESHOLD)) {
        signal.setType() = Fault_Signal
        signal.setHop() = selectedSignal.getHop() + 1
        RFTransmit(signal)
        }
    }
```

Case 2: The vital status of the agent is "Hungry"

The difference from the first case is that an agent with vital status "Hungry" stops searching for faults and starts to search for station signals. The states *Fault_Sensed* and *Fault_Reported* are replaced correspondingly by *Station_Sensed* and *Station_Reported* and the signal types are now different.

The *Station_Sensed* state is implemented by the *SearchStation* behaviour, which when it gives output (agents sensed directly a station's signal) except the position to move propose the agent to transmit a *Station_Sensed* signal (*HopCount*=0).

The *Station_Reported* state is implemented by the *StigmergicMove* behaviour, which when it gives output (agents sensed station reported signals) except the position to move propose the agent to retransmit the *Station_Reported* signal received increasing its *HopCount* by one, only if the input *HopCount* is less or equal from *HopThreshold*.

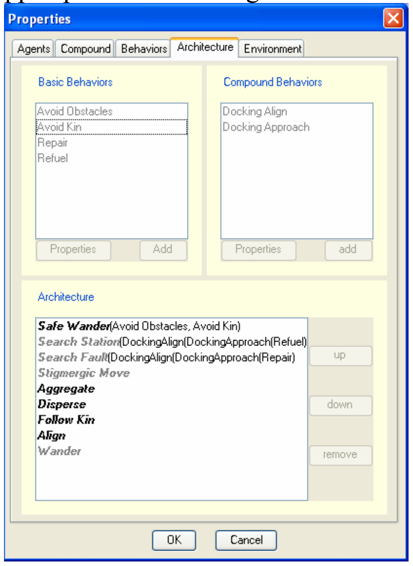
The *SearchStation* behaviour will be higher in the subsumption architecture so as to suppress the *SearchFault* and *StigmergicMove* behaviours, because "self-preservation instinct" is stronger than others and *Station_Sensed* state is more important than *Station_Reported*.

When an agent's vital status is "Hungry", it ignores all possible faults and all possible *Fault_Reported* signals may perceive.

The activation of indirect communication can be done by checking the "Communication" check box included in the "Agents" tapped pane in "Properties" panel of UI.

C. *Agent Architecture Construction*

The subsumption architecture can be constructed dynamically by the user, who can select any combination of basic and/or complex behaviours and to define the suppression order. This can be done using the UI "Architecture" construction tapped pane shown in Fig. 5.



Avoid Obstacles
Avoid Kin
Repair
Refuel

Docking Align
Docking Approach

Safe Wander(Avoid Obstacles, Avoid Kin)
Search Station(DockingAlign(DockingApproach(Refuel)
Search Fault(DockingAlign(DockingApproach(Repair)
Stigmergic Move
Aggregate
Disperse
Follow Kin
Align
Wander

Figure 5: Architecture construction tapped pane

The reason we used this approach was that we wanted the simulation system to allow free experimentation on a great variety of architecture combinations, so as finally to choose the best synthesis that gives the best robustness, in our effort to solve the problem we dealt with.

The only behaviour that is default in the agent architecture (and cannot be removed by the user) is the *Wander* behaviour, which gives output when no other behaviour gives output (suppressed by all others in higher levels).

There are some architecture constraints encoded in the application that automatically include or exclude some behaviour from the architecture depending on the parameters set by the user.

After large enough number of experiments (more than 200) we constructed the prototype synthesis of subsumption architecture shown in Fig. 6, which seems to give the best subsumption schema that exploits the aspects of all the behaviours described and allow the emergency of more complex behaviour.

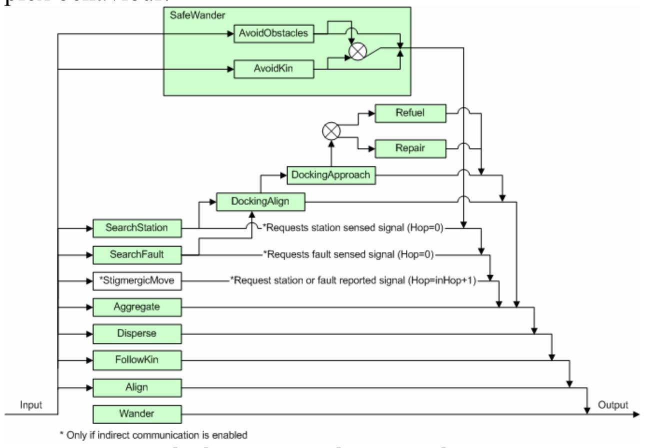

SafeWander
AvoidObstacles
AvoidKin
Refuel
Repair
DockingApproach
DockingAlign
SearchStation
SearchFault
*StigmergicMove
Aggregate
Disperse
FollowKin
Align
Wander
*Requests station sensed signal (Hop=0)
*Requests fault sensed signal (Hop=0)
*Request station or fault reported signal (Hop=inHop+1)
Input
Output
* Only if indirect communication is enabled

Figure 6: Proposed subsumption architecture schema

It includes all the behaviours described above. When the agents indirect communication is disabled the behaviour "*StigmergicMove*" removed from the architecture schema (because is dedicated to indirect communication and as we mentioned before implements the "Fault_Sensed" or "Station_Sensed" state of the communication protocol) and the "SearchStation" and "SearchFault" behaviours output does not request "Station_Sensed" or "Fault_Sensed" signal transmission correspondingly.

D. *Simulation Branch*

In the body of main frame of the application (Fig. 7) we constructed the simulation scene, which includes the static objects representing the environment and the dynamic objects representing the agents and the changes occurred in the environment (faults, signals, etc).

The environment consists of six boxes representing the side walls of a cubic petroleum tank and a vertical pipe on the centre of the tank where faults can be created by clicking with the mouse.

The agents are represented as cone objects, faults with red spheres, fault sensed signals with blue spheres, fault reported signals with green spheres and stations with light blue small cubes places on all environment sides in virtual grids (which calculated automatically depending on the agent population defined). The simulation scene is rotateable and can be seen by many different points of view.

Once a simulation is started (by pressing the "Start" button, after the desired system properties were set up), the agents population is placed in the virtual environment (randomly or starting from stations depending on the value of the corresponding parameter) and they start to implement the predefined subsumption architecture reacting with the environment (perceive environment and change it implementing the enabled behaviour output proposal).

The most important module of the application is the "*Sensor Simulator*", which simulates all the possible environment senses that an agent can perceive (and not only that).

In time fashion sends to each agent packages with all the possible senses of its local environment:

- Nearest static obstacle,
- Nearest kin positions & directions,
- Current fluid pH,
- Most important RF and
- Nearest station position or most important station signal.

Furthermore "*Sensor Simulator*" undertakes the mission to:

- Implement the algorithm of linear reduce of all signals life;
- Clear all the faults that have less than one fault units;
- Reduce agents battery charge and change agents vital status;
- Auto revive agents when the corresponding indicator is on, and
- Update simulation statistics.

Another important module of application is that which handles the phenomena of agent collision with the walls or kin.

We implemented a method that reduces fault sphere volume (at the repair procedure) depending on a volume reduce unit (*VRU*) per chemical compound unit (CCU) released by an agent. Each fault sphere is characterized by its radius (*r*) and the fault units it includes (*FU*), which can be defined in the "Compound" tapped pane of "Properties" of UI. So the volume reduce unit is given by:

$$V = \frac{4\pi r^3}{3} \quad (3) \qquad \text{and} \qquad VRU = \frac{V}{FU} \quad (4)$$

So, after each chemical compound unit is released for repairing a fault by an agent, a volume reduce unit subtracted from current fault volume and the changed fault sphere is re-drawn if the remaining volume is greater than zero or else the fault disappears.

We made the assumption that the maximum pH concentration (14) appears to the centre of a fault sphere with range of 0.4 (which volume corresponds to 6.55% of environments net volume), so all the fault sphere range pH concentrations are calculated using this as a base).

Also, we implemented a method that linearly reduces all signals intensity per time interval. Each signal sphere characterized by its radius (*r*) and life (l), which can be defined in the "Compound" tapped pane of "Properties" of UI. So the linear reduce unit (*LRU*) is given by: $LRU = \frac{r}{l}$ (5)

So, after each "*Sensor Simulator*" wakeup each signal radius is reduced by one LRU and the signal sphere is redrawn if the radius is greater than zero or else the signal disappears.

IV. SCENARIOS & EXPERIMENTS

We used general experiment scenarios to perform, in order to select the appropriate composition of:

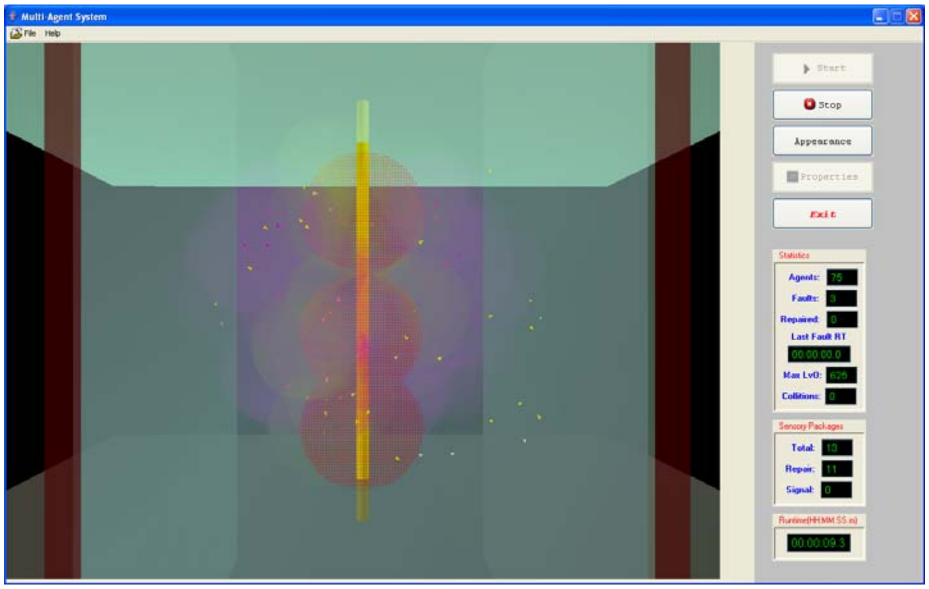
- Agent population,
- Subsumption architecture behaviour hierarchy and
- System parameterization.

In order to achieve a MAS system that repairs faster faults using a minimum agent population.

The basic axis of the scenarios is centralized on the issue of the use or not indirect communication and the parameters that improve the system performance with communicating agents, in order to minimize the randomness of agent's movement so as more agents to participate in repairing pro-

cedure which might reduce agent population and/or agent's chemical substance cargo.
Before we continue let's introduce some environment metrics (which is also given to "Environment" info pane of UI). The "time unit" is relative to wakeups per frame elapsed number defined in the CPU behaviour is given to each agent.
Each side of the "tank" is 1.6 **L**ength **U**nits (LU), so an agent with a velocity 0.05 LU per time unit needs 32 "time units" to cross from one side to the opposite.

units in combination with agent cargo allow as evaluating experiments with faults which needs a number (5, 6, 10) agents cooperation to be repaired. And the last one to evaluation of agent population in different numbers of environment faults presence.
As far as the signal parameters concern we used:
1. The *Fault_Sensed* and *Fault_Reported* sphere ranges (0.3, 0.25, 0.2),
2. The signal *life* (time units) (15, 10) and
3. *Hop_Threshold (*levels of signal retransmission*)* (3, 4).



Agents: 75
Faults: 3
Repaired: 0
Last Fault RT: 00:00:00.0
Max Lv0: 626
Collitions: 0
Total: 13
Repair: 11
Signal: 0
Runtime(HH:MM:SS.m): 00:00:09.3

Figure 7: A simulation example

The net fluid environment volume is approximately 4.0915 $(LU)^3$, so a fault or signal sphere with range 0.4 LU corresponds to 6.55%, with range 0.3 LU to 2.76%, with range 0.25 LU to 1.6% and with range 0.2 LU to 0.8% of net environment volume.
The "life" parameter of signal corresponds to the "time units" needed so as to completely fade.
Now we can introduce the set of parameters we used.
As far as the agent parameters concern we used:
1. The MAS population (50, 75, 100),
2. The chemical load units cargo (5, 10),
3. The with or without communication indicator and
4. The velocity (LU per time unit) (0.05, 0.06, 0.07).

We used these parameters because one of our goals is to reduce agent population, and this might be done by increasing the agent's cargo, or by using indirect communication or by increasing their speed.
We used the tiny sized cone shaped agent, with height=0.01 LU, base radius$\approx$0.0033 LU and volume=1,16355E-07 $(LU)^3$ (which corresponds to 0.000028‰ of environments net volume).
As far as the fault parameters concern we used:
1. The fault sphere range (0.4, 0.3, 0.25),
2. The max fault sphere fault units (80, 34, 20) and
3. The number of faults (3, 5, 7).

We use these parameters to evaluate systems with or without communication using large fault spheres (which might be in favor of systems without communication). The fault

We use this set of parameters to evaluate the best communication parameters synthesis, which does not distract the agents and allow the system to achieve the goals mention above.
Because all this range of parameters introduce a very large number of experiment combinations (3.888), we tried to reduce this number by keeping a standard velocity (0.5, which evaluated in previous set experiments as a good one, which reduce the possibility of agent collisions) and creating standard five faults per experiment (which will not harm the generality). So, we reduced the combinations to 432 (Fig. 8).

Agents			Fault Spheres		Signal Spheres		
Population	Chemical Load Units	Comm	Range	Fault Units	Range	Life	Hop Threshold
50	**5**	**Yes**	**0,4**	**80**	**0,3**	**15**	**3**
75	**10**	**No**	**0,3**	**34**	**0,25**	**10**	**4**
100			**0,25**	**20**	**0,2**		
Combinations			Combinations		Combinations		
3	2	2	3	1	3	2	2
3	6	**12**	3	**3**	3	6	**12**
		12		36			432

Figure 8: Experimentation scenarios parameters

The first general scenario is to find the best two of the twelve combinations of communications parameters, using a population of 75 agents and fault ranges 0.4 and 0.25 (24 experiments). Each experiment is executed four times and the evaluation is done by using the average of the results.
The second general scenario concerns the comparison of the same set of experiments with and without communication,

using the best two communication set of parameters. Each experiment without communications is executed four times and each experiment with communications is executed two times (54 experiments) and the evaluation is done by using the average of the results.

As a first criterion for the evaluation we selected the time needed for all the faults to be repaired. However, after the first experiments we found that this isn't reliable, due to the time threshold introduced by the large amount of "live" simulation objects (especially in experiments with communication). So, we decided to search for new criteria and we found the following:

- The number of sensory packages the system needs for the full repair of all faults and
- The number of sensory packages needed to fade all the communication signals after the repair of the last fault (the last criterion concerns the evaluation of experiments with communication).

After the execution and the analysis of the first general scenario set of experiments and having a light reserve happen by from the randomness of the experiments, we selected the below combination of indirect communication parameters (Fig. 9), which gives best performance (fasted repair of all faults) to a system with communication:

Signal Spheres		
Range	Life	Hop Threshold
0,3	15	3
0,25	15	4

Figure 9: First general scenario-best two communication parameters

Using the above best two combinations of indirect communication parameters we executed and analyzed the second general scenario set of experiments and we present the below table (Fig. 10) with the average results of corresponding experiments without and with indirect communication and evaluation. In the last two columns we present the percentage and new population needed so as a system without communication to have the same performance with a corresponding system with indirect communication.

In each experiment we made a partial evaluation and reached to conclusions and finally we evaluated the whole amount of experiments which were driven us to the final conclusions.

Parameters			Avg Sensory Packages Needed		Analysis	
Population	Agents Chemical load	Fault Sphere Range	No Comm	Comm	%	New Population
50	5	0,3	**1257,3**	**424,3**	296,35%	**148**
75	5	0,3	**427,5**	**232,3**	184,07%	**138**
100	5	0,3	**274,5**	**197,5**	138,99%	**139**
50	10	0,3	**392,3**	**299,8**	130,86%	**65**
50	5	0,4	**1692,8**	**1185,3**	142,82%	**71**
75	5	0,4	**902,0**	**521,3**	173,05%	**130**
50	10	0,4	**657,5**	**414,3**	158,72%	**79**
75	10	0,4	**355,5**	**250,0**	142,20%	**107**
100	10	0,4	**275,0**	**234,0**	117,52%	**118**

Figure 10: Second general scenario-table with average results of corresponding experiments without and with indirect communication and evaluation.

V. Conclusions

In the experiments of this paper we focused on the evaluation of a multi-agent system with indirect communication against systems without communication but from the whole procedure and other interesting conclusions were also emerged. Some of them include the following:

- The validation of robustness against failure (a small amount of collisions observed didn't affect considerably the general performance of the system),
- The validation of what we expected, which is that the systems with communication gives a great boost to the efficiency, but a suitable selection of communication parameters was needed to do so,
- To cover the lack of efficiency the systems without communication, needs a great increase of agents population,
- There is an upper limit on the population that close the difference in the efficiency between systems with and without communication and increases the collisions,
- The increase of *Hop_Thresold* might balance the reduce of signal *range* and/or *life*,
- The chemical load of agents combined with the number of faults that may be created in the environment, might create the need of increase or decrease of the agent's population,
- The systems with communication can trace easily small range faults,
- Indirect communications reduces the randomness of agent's movements to achieve their objectives.

The communication generally overloads the system concerning the extra equipment, increase of computational power, memory, energy consumption and cost and place constraints in the effort of agent's diminution.

On the other hand communication allow the reduction of agent population and even the agent's chemical load, because limits the randomness of their movement and increases the number of agents that participating in the task has been assigned.

Surely, there are intermediate solutions, like the reduction of signal range and/or life parallel with the increase of the *Hop_Threshold*, which might need less powered transceivers but with higher power consumption.

What is sure, is that the large number of parameters and behaviours increases the complexity of evaluation and construction of such systems and requires very big labour, research and experimentation.

References

[1] Bonabeau, E., Dorigo, M., and Theraulaz, G. Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, 1999.

[2] Brooks, R, A. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 2(1): 14–23, 1986.

[3] Brooks, R. A. *Robot: The future of flesh and machines.* London: Penguin Books, 2002

[4] Grasse, P-P. La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la Stigmergie: Essai d'interpretation du Comportement des Termites Constructeurs. Insectes Sociaux, 6:41 81, 1959.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003

[6] SOCIAL Project, IST-2001-38911, http://www.socialspike.net/

[7] Wooldridge, M. Intelligent Agents. In G. Weiss (Ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence (pp. 27-77). MIT Press, 1999.