Intelligent Management of Distributed Dynamic Sensor Networks

Peter Sapaty
Institute of Mathematical Machines and Systems, National Academy of Sciences
Glushkova Ave 42, 03187 Kiev Ukraine
Tel: +380-44-5265023, Fax: +380-44-5266457, sapaty@immsp.kiev.ua

Masanori Sugisaka
Department of Electrical and Electronic Engineering, Oita University
700 Oaza Dannoharu 870-1192 Japan
Tel: 097-554-7831, Fax: 097-554-7841, msugi@cc.oita-u.ac.jp

Jose Delgado-Frias
School of Electrical Engineering & Computer Science,
Washington State University, Pullman, WA 99164-2752, USA
Tel: (509)335-1156, Fax: (509)335-3818, jdelgado@eecs.wsu.edu

Joaquim Filipe
Departamento Sistemas e Informática,
Escola Superior de Tecnologia de Setúbal, 2910-761 Setúbal, Portugal
+351 265 790 040, +351 265 721 869 (fax), j.filipe@est.ips.pt

Nikolay Mirenkov
University of Aizu, Aizu-Wakamatsu, Fukushima-ken 965-8580, Japan
+81-242-37-2500, +81-242-37-2528 (fax), nikmir@u-aizu.ac.jp

Abstract

A universal solution for management of dynamic sensor networks will be presented, covering both networking and application layers. A network of intelligent modules, overlaying the sensor network, collectively interprets mission scenarios in a special high-level language, which can start from any nodes and cover the network at runtime. The spreading scenarios are extremely compact, which may be useful for energy saving communications. The code will be exhibited for distributed collection and fusion of sensor data, also for tracking mobile targets by scattered and communicating sensors.

Keywords: sensor networks, intelligent management, distributed scenario language, distributed interpreter, tracking objects, hierarchical data fusion.

1 Introduction

Sensor networks are a sensing, computing and communication infrastructure that allows us to instrument, observe, and respond to phenomena in the natural environment, and in our physical and cyber infrastructure [1, 2]. The sensors themselves can range from small passive microsensors to larger scale, controllable platforms. Their computation and communication infrastructure will be radically different from that found in today's Internet-based systems, reflecting the device- and application-driven nature of these systems.

Of particular interest are wireless sensor networks (WSN) [3,4], consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations. WSN, however, have many additional problems in comparison to the wired ones. The individual devices in WSN are inherently resource constrained--they have limited processing speed, storage capacity, and communication bandwidth. These devices have substantial processing capability in the aggregate, but not individually, so we must combine their many vantage points on the physical phenomena within the network itself.

In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. The size a single sensor node can vary from shoebox-sized nodes down to devices the size of grain of dust.

Typical applications of WSNs include monitoring, tracking, and controlling. Some of the specific applications are habitat monitoring, object tracking, nuclear reactor controlling, fire detection, traffic monitoring, etc. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes. They could be deployed in wilderness areas, where they would remain for many years (monitoring some environmental variable) without the need to recharge/replace their power supplies. They could form a perimeter about a property and monitor the progression of intruders (passing information from one node to the next). At present, there are many uses for WSNs throughout the world.

In a wired network like the Internet, each router

connects to a specific set of other routers, forming a routing graph. In WSNs, each node has a radio that provides a set of communication links to nearby nodes. By exchanging information, nodes can discover their neighbors and perform a distributed algorithm to determine how to route data according to the application's needs. Although physical placement primarily determines connectivity, variables such as obstructions, interference, environmental factors, antenna orientation, and mobility make determining connectivity a priori difficult. Instead, the network discovers and adapts to whatever connectivity is present.

Fig. 1 shows what we will mean as a sensor network for the rest of this paper. It will hypothetically consist of (many) usual sensors with local communication capabilities, and (a limited number of) those that can additionally transmit collected information outside the area (say, via satellite channels). Individual sensors can be on a move, some may be destroyed while others added at runtime (say, dropped from the air) to join the existing ones in solving cooperatively distributed problems.

The aim of this paper is to show how any imaginable distributed problems can be solved by dynamic self-organized sensor networks, if to increase their intelligence with a novel distributed processing and control ideology and technology effectively operating in computer networks.

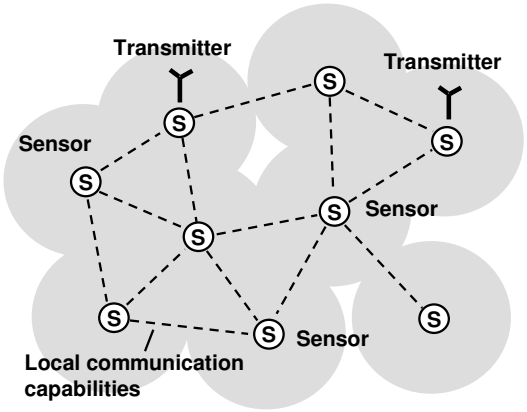

Figure 1. Distributed sensors and their emergent network.

2 The Distributed Management Model

The distributed information technology we are using here is based on a special Distributed Scenario Language (DSL) describing parallel solutions in computer networks as a seamless spatial process rather than traditional collection and interaction of parts (agents).

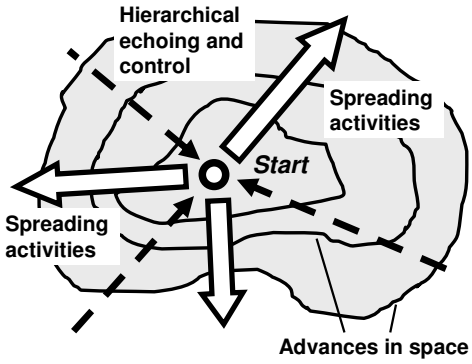

Figure 2. Runtime coverage of space.

Parallel scenarios can start from any interpreter of the language, and then spread and cover the distributed space at runtime, as shown in Fig. 2. The overall management of the evolving scenarios is accomplished via the distributed track system providing hierarchical command and control for the execution of scenarios, with a variety of special echo messages. We will mention here only key features of DSL, as the language details can be found elsewhere in the current proceedings [5] and from its previous versions described in [6-8].

A DSL program, or *wave*, is represented as one or more constructs called *moves* (separated by a comma), embraced by a *rule*, as follows:

wave → *rule* ({ *move* , })

Rules may serve as various supervisory, regulatory, coordinating, integrating, navigating, and data processing functions, operations or constraints over moves. A move can be a *constant* or *variable*, or (recursively) an arbitrary wave itself:

move → *constant* | *variable* | *wave*

Variables classify as *nodal*, associated with space positions and shared by different waves, *frontal*, moving in space with program control, and *environmental*, accessing the environment navigated. Constants may reflect both information and physical matter.

Wave, being applied in a certain position of the distributed world, can perform certain actions in a distributed space, terminating in the same or in other positions. It provides final result that unites local results in the positions (nodes) reached, and also produces resultant control state. The (distributed) result and the state can be subsequently used for further data processing and decision making on higher program levels. Parallel waves can start from different nodes in parallel, possibly intersecting in the common distributed space when evolving in it independently.

If moves are ordered to advance in space one after the other (which is defined by a proper rule), each new move is applied in parallel in all the nodes reached by the previous move. Different moves (by other rules) can also apply independently from the same node, reaching new nodes in parallel.

The functional style syntax shown above can express any program in DSL, but if useful, other notations can be used, like infix one. For example, an advancement in space can use period as operator (separator) between successive steps, whereas parallel actions starting from same node can be separated by semicolon. For improving readability, spaces can be inserted in any places of the programs--they will be automatically removed before execution (except when embraced by quotes).

The interpreter may have its own physical body (say, in the form of mobile or humanoid robot), or can be mounted on humans (mobile phones). A network of the interpreters can be mobile and open, changing its volume and structure, as robots or humans can move at runtime. We will be assuming for the rest of this paper that every sensor has the DSL interpreter installed, which may have a software implementation or can be a special hardware chip.

In the following sections we will show and explain the DSL code for a number of important problems to be

solved by advanced sensor networks, which confirms the efficiency of the proposed distributed computational and control model.

3 Collecting Events throughout the Region

Starting from all transmitter nodes, the following program regularly (with interval of 60 sec.) covers stepwise, through local communications between sensors, the whole sensor network with a spanning forest, lifting information about observable events in each node reached. Through this forest, by the internal interpretation infrastructure, the lifted data in nodes is moved and fused upwards the spanning trees, with final results collected in transmitter nodes and sent in parallel outside the system using rule `Transmit` (See Fig.3).

```
Hop (all transmitters).
Loop (
 Sleep (20).
 IDENTITY = TIME.
 Transmit (
  Fuse (
   Repeat (
    Free (observe (events));
    Hop (directly reachable, first come)
   )
  )
 )
)
```

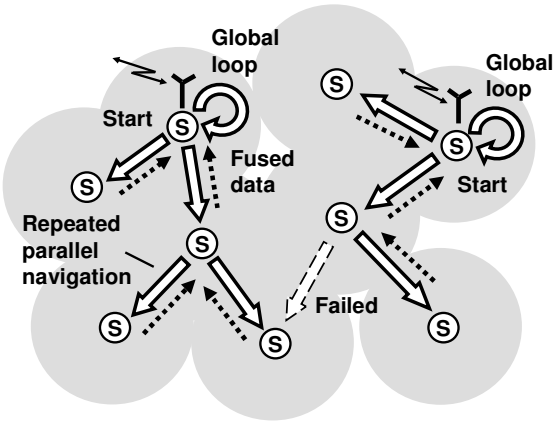

Figure 3. Parallel navigation and data collection.

Globally looping in each transmitter node (rule `Loop`), the program repeatedly navigates (rule `Repeat`) the sensor set (possibly, in competition with navigation started from other transmitters), activating local space observation facilities in parallel with the further navigation. The resultant forest-like coverage is guaranteed by allowing sensor nodes to be visited only once, on the first arrival in them. The hierarchical fusion rule `Fuse`, collecting the scattered results, also removes record duplicates, as the same event can be detected by different sensors, leaving only most credible in the final result.

To distinguish each new global navigation process from the previous one, it always spreads with a new identity for which, for example, current system time may be used (using environmental variables `IDENTITY` and `TIME` of the language).

4 Regular Creation of Hierarchical Infrastructures

In the previous program, we created the whole spanning forest for each global data collection loop, which may be costly. To optimize this process, we may first create a persistent forest infrastructure, remembering which nodes were linked to which, and then use it for a frequent regular collection and fusion of the scattered data.

As the sensor neighborhood network may change over time, we can make this persistent infrastructure changeable too, updating it with some time interval (much larger, however, than the data collection one), after removing the previous infrastructure version. This can be done by the following program, which regularly creates top-down oriented links named `infra` starting from the transmitter nodes (as shown in Fig. 4).

```
Hop (all transmitters).
Loop (
 Sleep (120).
 IDENTITY = TIME.
 Repeat (
  Hop (directly reachable, first come).
  Remove links (all).
  Stay (create link (-infra, BACK))
 )
)
```

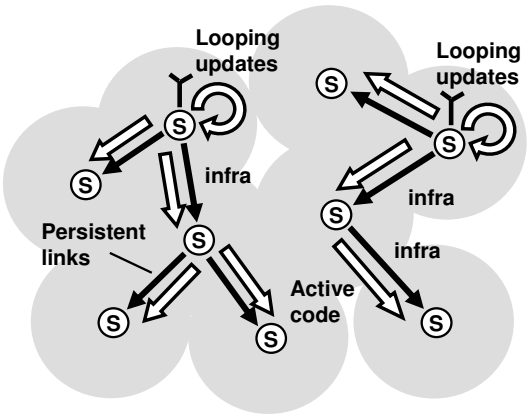

Figure 4. Runtime creation of hierarchical infrastructure.

This infrastructure creation program provides competitive asynchronous spatial processes, so each time, even if the sensors did not change their positions, the resultant infrastructure may differ, as shown in Fig. 5.

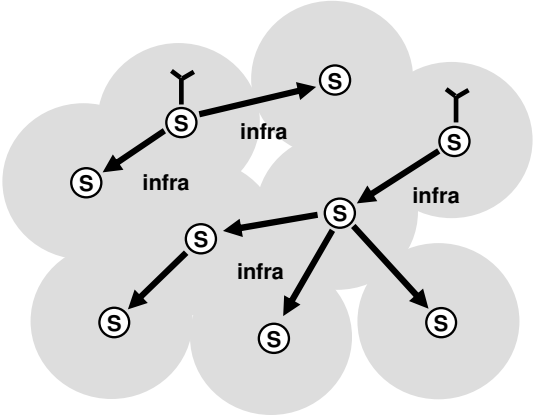

Figure 5. Another possible infrastructure.

Having created the persistent infrastructure, we can use it frequently by the event collection program which can be simplified and updated now as follows:

```
Hop (all transmitters).
Loop (
 Sleep (20).
 Transmit (
  Fuse (
   Repeat (
    Free (observe (events));
    Hop (+infra)
   )
  )
 )
)
```

The global infrastructure creation program (looping slowly) and the event collection and fusion one (looping fast) can operate simultaneously, with the first one guiding the latter on the data collection routes, which may change over time.

5 Routing Local Events to Transmitters

We have considered above the collection of distributed events in the top-down and bottom-up mode, always with the initiative stemming from root nodes of the hierarchy, the latter serving as parallel and distributed tree-structured computer.

In this section, we will show quite an opposite, fully distributed solution, where each sensor node, being an initiator itself, is regularly observing the vicinity for the case an event of interest might occur. Having discovered an event, each node independently from others launches a spatial cyclic self-routing process, via the infrastructure links built before, which eventually comes to the transmitter node, bringing with it the event information. The data brought to transmitters should be fused with the already existing there. The corresponding program will be as follows.

```
Hop (all nodes).
Frontal (Transfer). Nodal (Result).
Loop unconditional (
 Sleep (5).
 Nonempty(Transfer = observe (events)).
 Repeat (hop (-infra)).
 Fuse and assign (Result, Transfer)
)
```

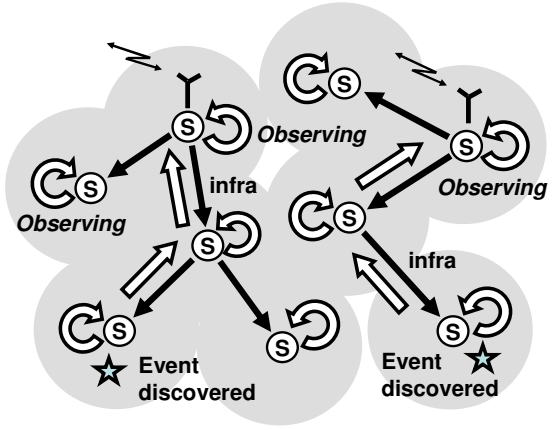

Figure 6. Routing scattered events to transmitters.

The transmitter nodes, accumulating and fusing local events, arriving from sensor nodes independently, can send them outside the system. Different strategies can be used here. For example, one could be waiting until there are enough event records collected in the transmitter before sending them, and the other one waiting for some threshold time and only then sending what was accumulated (if any at all). The following program combines these two cases within one solution, where arriving data from sensors is accumulated in nodal variable `Result`.

```
Hop (all transmitters).
Loop unconditional (
 Or (
  Quantity (nodal ((Result)) >= 100,
  (sleep 60. Result != nil)
 ).
 Transmit and clear (Result).
)
```

This program in every transmitter can work in parallel with the previous program collecting events and looping in every sensor (in transmitters as well, as they are assumed to be sensors too), and also with the earlier program, starting in transmitters, for the regular infrastructure updates.

6 Tracking Mobile Objects

Let us consider some basics of using DSL for tracking mobile (ground or aerial) objects moving through a region controlled by scattered but communicating sensors. Each sensor can handle only limited part of space, so to keep the whole observation continuous, the object seen should be handed over between the neighboring sensors during its movement, along with the data accumulated during its tracking and analysis.

The space-navigating power of the model discussed here can catch each object and accompany it individually, while moving between the interpreters in different sensors, just accompanying the movement in physical space by an active mobile intelligence spreading in a computer network [6]. This allows us to have an extremely compact and integral solution unattainable by other approaches based on communicating agents. The following program, starting in all sensors, catches an object it sees, and follows it wherever it should go if not seen at this point any more (more correctly: if its visibility becomes lower that the given threshold).

```
Hop (all nodes).
Frontal (Threshold) = 0.1.
Frontal (Object) = search (aerial).
Visibility (Object) > Threshold.
Repeat (
 Loop (
  visibility (Object) > Threshold
 ).
 Maximum destination (
   Hop (all directly reachable).
   Visibility (Object) > Threshold
 )
)
```

The program investigates the object's visibility in all

neighboring sensors in parallel, and moves control along with the program code and accumulated data (the latter not shown in this simplified program) to the neighboring sensor seeing it best -- again, if its visibility exceeds the threshold given (see Fig. 7).

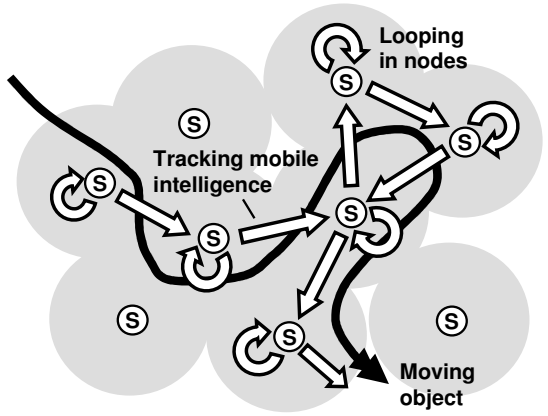

Figure 7. Active tracking of a mobile object.

This was only a skeleton program in DSL, showing the space tracing techniques for controlling single physical objects. It can be extended to follow collectively behaving groups of physical objects as a whole (say, flocks of animals, mobile robots, or troops). The spreading individual object-tracing intelligences can cooperate in the distributed sensor space, and can be optimized jointly for the pursuit of global mission goals.

7 Averaging Parameters from a Region

Let us consider how it can be possible to asses the generalized situation in a distributed region given, say, by a set of its border coordinates, in a fully distributed way, where sensors located in the region can communicate with direct neighbors only. Assume, for example, that the parameter of interest is maximum pollution level throughout the whole region (it may also be temperature, pressure, radiation level, etc.) together with coordinates of the location showing this maximum.

The following program, starting in all sensors located in the region, regularly measures the pollution level in its vicinity, updates local maximum, and by communication with direct neighbors attempts to increase maximum there too, if this is possible. Eventually, after some expected time of such local communication activity, all sensors will have the same maximum value registered in them, which will be the one of whole region too (see the overall organization in Fig. 8).

```
Nodal (Level, Max, Region).
Frontal (Transfer).
Region = region definition.
Hop (all nodes, Region).
Loop unconditional (
 Or parallel (
  Loop unconditional (
   Sleep (5).
   Level = measure (pollution).
   Stay (Level > Max. Max = Level).
   Transfer = Max.
   Hop (directly reachable, Region).
   Transfer > Max. Max = Transfer
  ),
  Sleep (120)
 ).
 Level == Max.
 Transfer = Max & WHERE.
 Repeat (hop (- infra)).
 Transmit (Transfer)
)
```

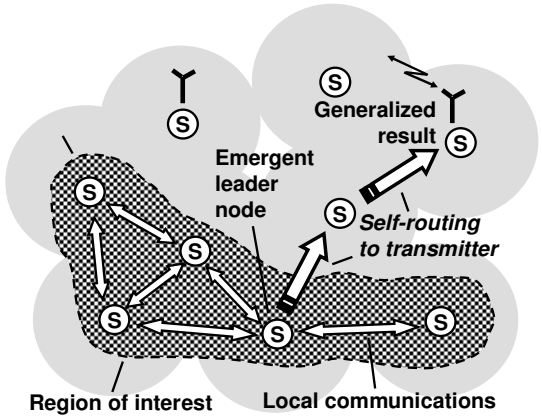

Figure 8. Distributed averaging with active routing.

As there may be many sensors located in the region of interest, we will need forwarding only a single copy of this resultant maximum value to a transmitter for an output. This can be achieved by delegating this only to the sensor whose measured local value is equal to the accumulated maximum in it, which will correspond to the overall region's maximum after some time. Having understood that it is the leader (after a proper time delay), such a sensor organizes repeated movement to the nearest transmitter via the earlier created infrastructure, carrying the resultant maximum value in frontal variable `Transfer`, outputting it outside the system in the transmitter reached, as shown in Fig. 8.

Similar organization may be introduced for finding averaged values, or even for assembling the global picture of the whole region with any details collected by individual sensors (the latter may be costly, however, with more practical solution shown in the next section).

8 Assembling Full Picture of a Region

To collect details from some region via local sensors and merge them into the whole picture could, in principle, be possible via local single-level exchanges only, as in the previous section, but the amount of communications and data transfer, as well as the time needed, may be unacceptably high. We were finding only a single value (maximum) via frequent internode communications, with minimal length exchanges. But for obtaining the detailed global picture of the region or some distributed phenomenon, we may have to gradually grow this picture in every sensor node (or in many of them) simultaneously, with high communication intensity between nodes. Also, there may be difficult to determine the completeness of this picture staying in local sensors only.

It is clear that much higher integrity and process structuring, especially with the use of hierarchies, may be required in order to see the whole distributed picture

via dispersed sensors with limited capabilities in nodes and emergent communications. Different higher-level approaches can be proposed in DSL for solving such classes of problems.

We will show only a skeleton here of how to make spanning tree coverage of the distributed phenomenon, and then hierarchically collect, merge, and fuse partial results from sensors into the global picture. The latter to be forwarded to the nearest transmitter via the previously created infrastructure using links `infra`, as shown in Fig. 9.

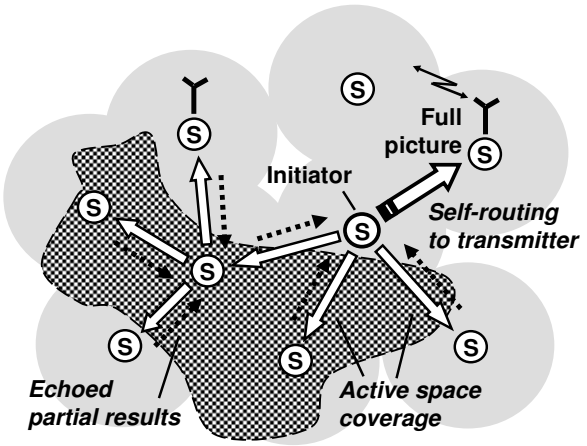

Figure 9. Space coverage with global picture fusion.

```
Hop (
 random, all nodes,
 detected (phenomenon)
).
Loop (
 Frontal (Full) = fuse (
  Repeat (
   Free (collect (phenomenon));
   Hop (
    directly reachable, first come,
    detected (phenomenon)
   )
  )
 ).
 Repeat (hop (-infra)).
 Transmit (Full)
)
```

In more complex situations, which can be effectively programmed in DSL too, we may have a number of simultaneously existing phenomena, which can intersect in a distributed space; we may also face combined phenomena integrating features of different ones. The phenomena (like flocks of birds, manned or unmanned groups or armies, spreading fire or flooding) covering certain regions may change in size and shape, they may also move as a whole having internal organization, etc.

In the previous versions of this language [6, 7], a variety of complex topological problems in computer networks had been investigated and successfully programmed in a fully distributed and parallel manner, which included connectivity, matching with graph patterns, weak and strong components like articulation points and cliques, also diameter and radius, optimum routing tables, and the like, as well as self-recovery after indiscriminate damages (see [6] especially).

9 Conclusions

We have presented a universal and flexible approach of how to convert distributed sensor networks with limited resources in nodes and casual communications into a ***universal spatial machine*** capable of not only collecting and forwarding data but also solving complex computational and logical problems as well as making autonomous decisions in distributed environments.

The approach is based on quite a different type of a high-level language allowing us to represent system solutions in the form of ***integral seamless spatial processes*** navigating and covering distributed worlds at runtime. This makes parallel and distributed application programs extremely short, which may be especially useful for the energy saving communications between sensors.

The code compactness and simplicity are achieved because most of traditional synchronization and data or agent exchanges (which are also on a high level, with minimum code sent) are shifted to efficient automatic implementation, allowing us concentrate on ***global strategies and solutions*** instead.

References

[1] D. Culler, D. Estrin, and M. Srivastava, “Overview of Sensor Networks” , Computer, August 2004, pp.41-49. Published by the IEEE Computer Society.

[2] C-Y Chong, and S. P. Kumar, “Sensor Networks: Evolution, Opportunities, and Challenges”, Proc. of the IEEE, Vol. 91, No. 8, August 2003, pp.1247-1256.

[3] Wireless sensor network--Wikipedia, the free encyclopedia, www.wikipedia.org.

[4] F. Zhao and L. Guibas, Wireless Sensor Networks: An Information Processing Approach (The Morgan Kaufmann Series in Networking), Morgan Kaufmann, 2004, 376p.

[5] P. Sapaty, A. Morozov, R. Finkelstein, M. Sugisaka, D. Lambert, “A New Concept of Flexible Organization for Distributed Robotized Systems”, Proc. Twelfth International Symposium on Artificial Life and Robotics (AROB 12th’07), Beppu, Japan, Jan 25-27, 2007. 8p.

[6] P. S. Sapaty, Mobile Processing in Distributed and Open Environments, John Wiley & Sons, ISBN: 0471195723, New York, February 1999, 436p. (www.amazon.com).

[7] P. S. Sapaty, Ruling Distributed Dynamic Worlds, John Wiley & Sons, New York, May 2005, 256p, ISBN 0-471-65575-9 (www.amazon.com).

[8] P. Sapaty, M. Sugisaka, R. Finkelstein, J. Delgado-Frias, N. Mirenkov, “Advanced IT Support of Crisis Relief Missions”, Journal of Emergency Management, Vol.4, No.4, ISSN 1543-5865, July/August 2006, pp.29-36 (www.emergencyjournal.com).