A New Concept of Flexible Organization for Distributed Robotized Systems

Peter Sapaty Anatoly Morozov
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

Robert Finkelstein
Robotic Technology Inc., 11424 Palatine Drive, Potomac Maryland 20854
Tel: 301-983-4194 Voice, Fax: 301-983-3921
RobertFinkelstein@compuserve.com

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

Dale Lambert
Fusion for Situation Awareness Initiative, Defence Science and Technology Organisation
PO Box 1500 Edinburgh South Australia 5111
Tel: (+61) 8 8259 7175, Fax: (+61) 8 8259 5589
Dale.Lambert@dsto.defence.gov.au

Abstract

The presented concept for management of distributed dynamic systems is based on installing in all system components of a universal intelligent module interpreting special high-level language, in which any centralized or distributed control can be expressed. The mission scenario in the language, starting from any interpreter, is collectively executed by their network. The interpreters perform appropriate operations in nodes, while passing other parts of the scenario, together with intermediate data, to other interpreters in a coordinated manner. This process covers the system at runtime and sets up distributed command and control infrastructures providing the overall integrity and goal orientation. The approach allows us to manage robotized systems in unpredictable and hostile environments, with possible failures of components. The language description and programming examples in it are provided.

Keywords: crisis management, cooperative engagement capability, ubiquitous command and control, distributed systems, high level language, mission scenario, interpretation network, robotized systems.

1 Introduction

In managing large systems, whether civil or military, we usually think of them as of something already existing, with proper expertise in parts (nodes) and relations between them. We also assume that there exists a sort of command and control infrastructure, usually hierarchical, covering all its parts, through which any external orders to these systems are received and both internal and outside impacts realized. This infrastructure should also support the overall system stability and integrity and guide its internal behavior in accordance with the existing rules and local and global goals pursued.

But in complex environments, the system organization may be indiscriminately damaged at any moment of time, with skills vanished in nodes, relations broken, and infrastructures destroyed. To put the system back into life, we may need to restore its parts, as well as the whole, at runtime [1].

Moreover, quite different philosophies, supported by new technologies, to the organization of dynamic emergent systems may be needed. First of all, the power and universality of the traditional command and control (or C2) may be questioned, as this already takes place in the area of crisis and disaster response [2,3].

The C2 approach is based on the idea that the right way to manage disasters is through centralized control and hierarchies. But actual community crisis response networks, as indicated by reaction on the recent well known disasters, looked nothing like the military-like C2 hierarchies [2]. They consisted of loosely-coupled collections of individuals, groups, and organizations that continually changed, having permeable boundaries.

Rather than being organized according to the principles of command and control, disaster response activities were undertaken through a complex and varied set of organizational arrangements characterized by a high degree of emergence and improvisation. New

networks formed that blended the activities of existing organization with those of emergent groups, which dealt with local problems as they emerged, using the existing resources in novel ways. The decentralized multiorganizational responses appeared to be a major strength, rather than weakness, and a source of resilience. Centralization and hierarchy only slowed down and hampered response efforts.

In this paper, we propose to automate the process of runtime composition of dynamic distributed systems from dissimilar, possibly casual, elements and creation of any infrastructures covering them -- by shifting the overall system organization to a higher, semantic level with orientation on both manned and unmanned systems dedicated to operate in emergency situations.

First, we briefly describe the two well-known organizational approaches to distributed dynamic systems, namely, Cooperative Engagement Capability (CEC) and Ubiquitous Command and Control (UC2) [6-8]. We then reveal a technology, based on a higher-level control language, which can effectively support both CEC and UC2, as well as any other, especially crisis management, systems. Related programming examples will be presented too.

The previous applications of the technology presented include intelligent network management, distributed interactive simulation, distributed knowledge bases, group behavior, and support of robotized infrastructures [4, 5].

2 Basic Tends in Organization of Distributed Systems

We outline here the CEC and UC2 approaches, originally with military orientation, but with obvious significance to any systems with limited local resources but oriented on global problems. The first one provides global awareness to the distributed system in any its local node, and the second one allows us restructure the system's command and control at runtime, without loss of global functionality and goal orientation. CEC and UC2 actually complement each other, rather than compete, with UC2, however, being potentially a broader approach, covering the CEC capabilities as well.

2.1 Cooperative Engagement Capability

The Cooperative Engagement Capability [6] brings a new power to distributed systems, not by adding new sensor and impact components, but by distributing data from them in a significantly different manner. CEC fuses high quality tracking data from participating sensors and distributes it to all other participants in a filtered and combined state, using identical algorithms to create a single, common picture. CEC has a robust communications system with considerable improvement to bandwidth and countermeasures, as well as the advantages offered by the global positioning system. CEC's main components and functions are as follows.

The Data Distribution System provides line-of-sight (LOS) communication with other units. The Cooperative Engagement Processor provides processing of data received from other units and incorporates it with own platform data to form a single composite data, forming the same picture on each unit. The Data Distribution Function provides real time data transfer in LOS and over-the-horizon. The Command/Display Support Function performs doctrine management and distribution, also group operations in defending against threats. The Sensor Cooperation Function provides increased detection and track performance by using composite track data from active sensors. The Engage Decision Function provides the capability for decisions to be made by an automated process based on doctrine entered by the Net Control Unit. The Engagement Execution Function supports the control process for impact of designated targets and responds to command directions and decisions.

2.2 Ubiquitous Command and Control

A Ubiquitous Command and Control system [7, 8] is a system of assets, all of which possess a similar C2 capability. UC2 systems represent devolution of decision making power from C2 centers to platforms which are designed to provide alternative functionality. Under this philosophy, command and control becomes an additional function performed in any manned or unmanned units, and ubiquitous C2 systems are so named because they advocate a C2 capability on every platform. Automation is the primary mechanism for acquiring a similar C2 capability in any unit, and some decision making can be fully automated. Other aspects will perform better with human interaction, with the choice between the two being mediated empirically.

The automated and human decision making are fully integrated, and this includes the option of allowing the machine to override or substitute the human. UC2 systems primarily endorse a distributed and decentralized management structure. It also introduces a command fusion problem, as each decision maker may fuse requests for its resources from multiple sources. In the information age, C2 centers may often become the prime targets for surgical impacts. In defending against the latter, one approach is to build duplicate C2 centers. Supporting this, UC2 also enables C2 functionality to reconfigure as necessary, offering greater sustainability, as well as quality of system performance, with graceful, rather then instantaneous, degradation.

3 Flexible Distributed Management Model

The distributed computation and control model and technology, previously known as WAVE [4] and WAVE-WP (or World Processing) [5], is based on a higher-level language describing parallel distributed solutions in computer networks as a single seamless spatial process rather than traditional collection and interaction of parts (agents). Communicating copies of the language interpreter (as universal control modules U in Fig. 1) should be installed in important system resources (like internet hosts, mobile robots, or mobile phones), which may be emergent. Parallel spatial scenarios written in the language can start from any

interpreter, covering the network at runtime, cooperating or competing with each other in the distributed space.

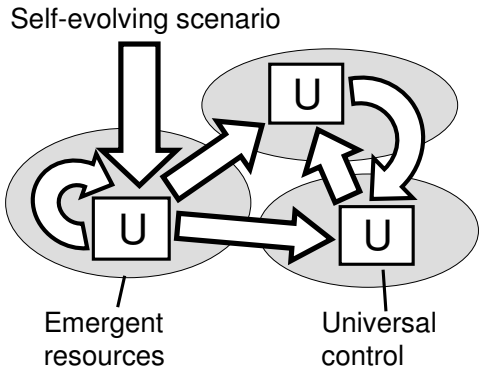

Figure 1. The universal control network.

The spreading scenarios can create dynamic knowledge infrastructures arbitrarily distributed between computers (robots). Subsequently or simultaneously navigated by same or other scenarios, these infrastructures can effectively support distributed databases, command and control, global situation awareness, parallel inference, and autonomous decisions. It is possible to operate in this seamless virtual world fully ignoring its physical distribution, whereas virtual networks can migrate (partially or as a whole) in physical networks while being processed. The distributed virtual world can optimize and guide movement and operations in the physical world, say, by robotic groups (armies).

The system mission in the language sets what the system should do in a distributed space rather than how to do this, which resources to use, or how C2 should be organized. The main burden on actual composition of the system, its internal organization, and runtime recovery is shifted to efficient distributed implementation of the scenario language. This allows us to continuously support the development of missions and global goal orientation regardless of the state of system resources, keeping the system operable if at least a single node remains functional. Any top down, bottom up and combined solutions are available within this spatial programming paradigm. The approach often provides hundreds of times application code reduction and simplification, allowing us to concentrate on efficient global solutions rather than implementation details.

4 Distributed Scenario Language

Let us symbolically call it within the current application context as Distributed Scenario Language (or DSL), whereas the earlier versions carried names WAVE [4], WAVE-WP [5], and WPL [9]. The DSL syntax in the most general form can be represented as follows:

wave → *rule* ({ *move* , })
move → *constant* | *variable* | *wave*
variable → *nodal* | *frontal* | *environmental*

The program, or *wave*, is represented as one or more constructs called *moves*, which are separated by a comma and embraced altogether by another construct called *rule* (in the functional style, using parentheses). The name "wave" reflects the general space navigation ideology of the approach, where the program code can cooperatively cover the distributed system in parallel wavelike steps. And similarly, "move" highlights the potential mobility of all language constructs.

Rules serve as various supervisory, regulatory, coordinating, integrating, navigating, and data processing functions, operations or constraints over moves, which, for example, may be:

- elementary arithmetic, string or logic operations on data returned by moves;
- hops in physical, virtual or combined spaces parameterized by moves;
- hierarchical fusion and return of (remote) data provided by moves;
- parallel and distributed control over the development of moves as programs, covering control flow of usual languages too;
- special contexts of navigation in space, for example, causing creation of different infrastructures by the evolving and spreading moves.
- sense of values expressed by moves, for their proper use by other rules.

Moves can represent values directly, as a *constant* or *variable*, or can recursively be arbitrary waves themselves. Variables can be classified as *nodal*, or stationary, associated with space positions and shared by different waves; *frontal*, moving in space with the program control; and *environmental*, accessing the navigated environment in the points reached. If control splits by parallel moves, each move receives independent copies of all frontal variables. Constants may reflect information or physical matter, the latter to be processed if proper physical equipment is available.

Wave is applied in a certain position of the distributed world, providing data processing and space navigation and transformation, eventually terminating in the same or in other positions (which may be multiple and remote). It provides final result that unites local results in the positions (or nodes) reached, and also produces a resultant control state. These two can be subsequently used for further data processing and decision making on higher program levels.

If moves are set to advance in space one after the other (defined by a proper rule), each new move is applied in parallel in all the nodes reached by the previous move, with the rest of the program also moving to the new locations (virtually or if needed, physically). Different or same moves (by other rules) can also apply independently from the same node, reaching new nodes asynchronously and in parallel.

The syntax shown above can represent any program in DSL, but if convenient, other notations can be used, like the infix one. For example, the program

```
advance (move1, move2, move3)
```

ordering three moves develop sequentially, each move

from the positions in space reached by the previous move, can be represented as:

```
move1. move2. move3
```

with the period indicating advancement in space. Similarly,

```
parallel (move1, move2, move3)
```

can be written as:

```
move1; move2; move3
```

with the semicolon setting independent and parallel development. As another example,

```
assign (R, multiply (sum (a, b, c), d))
```

can be substituted by:

```
R = (a + b + c) * d
```

For improving readability, spaces can be inserted in arbitrary places of the programs; they (as well as carriage returns) will be automatically removed (except when reside in strings in quotes) during the program interpretation. Also, it is often useful to show programs using indentations, with placing related opening and closing parentheses exactly one over the other.

More details about different constructs and their meanings can be obtained from the previous versions of the language [4, 5].

5 Parallel Language Interpretation

The peculiarities of the syntax and semantics of DSL allow us to provide its effective, fully distributed and parallel, interpretation without central resources. During this process, the spatial unwrapping and replication of the recursive program formula takes place, rather than its traditional reduction.

A DSL program covers and matches the physical or virtual world in parallel, establishing full control over the distributed space. Each operation is performed in the reached nodes on local data there (environmental, and in nodal variables), on what has been brought to these nodes in frontal variables with the program control, and on the obtained and returned results (possibly, remote) by subordinate waves.

The intermediate and final results of the work of DSL programs may be scattered throughout the whole navigated space; they may be grouped and returned into a certain point (or points) if this is needed. Different evolving parallel and distributed waves can cooperate or compete in the common, open, distributed space.

The DSL interpreter consists of a number of specialized modules working in parallel (like *parser*, *data processor*, *control processor*, and *communication processor*). These are handling and sharing specific data structures supporting persistent virtual worlds and temporary hierarchical control mechanisms, like *wave queue*, *incoming and outgoing queues*, local part of the *distributed knowledge network*, *track forest*, *nodal, environmental and frontal variables*, etc. [4, 5].

The interpreter may have its own physical body (say, in the form of mobile or humanoid robot), or can be mounted on humans (e.g. in mobile phones). The whole network of the interpreters can be mobile and open, changing the number of nodes and communication structure between them, as robots or humans can move at runtime. The DSL operations may trigger a combination of data processing and physical movement in space, with exchange of information and physical matter between the interpreters both electronically and in a direct contact.

6 Dynamic Creation of Distributed Infrastructures

We will consider here simplified programs in DSL, which are creating different infrastructure topologies over the distributed, and possibly scattered, system nodes. The main feature of all these programs is that they can do the job in a fully distributed manner, by covering and flooding the dynamic and open system in parallel and cooperative mode, without any central resources.

These programs, as well as those in subsequent sections, have a quite different semantics than usual ones, as each their construct may work in other locations in space (and in other computers), not in the same as the previous ones. Despite this, all the programs constantly preserve full integrity and controllability as the whole, similar to traditional single-machine programs operating in the same memory. This possibility is effectively achieved by a powerful implicit distributed track system and also internal "command and control" infrastructures underlying the distributed DSL interpretation, see for previous versions at [4, 5].

6.1 Star

Starting in any node, the following program forms oriented links with name `star` to all other nodes that can be reached directly or indirectly (i.e. via other nodes) from the current node (as in Fig. 2).

```
Create links (+ 'star', other nodes)
```

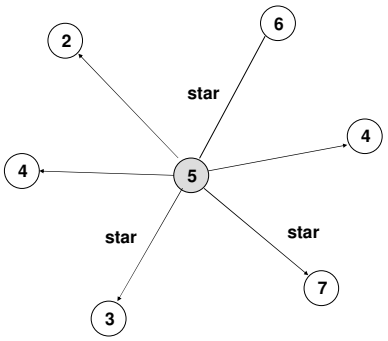

Figure 2. A star infrastructure.

6.2 Full Graph

Starting in any node too, the program below first hops to all directly or indirectly reachable nodes, including itself, and then from all the nodes reached forms a non-oriented link named `full` to all other nodes,

reached directly or indirectly too (see Fig. 3).

```
Hop (all nodes).
Create links (
 'full', other nodes (inferior)
)
```

To avoid duplicate links, the formation of a link between any two nodes takes place only in one way, allowing the superior node (or inferior, as another solution) to create the link (by comparing certain node's values, e.g. addresses).

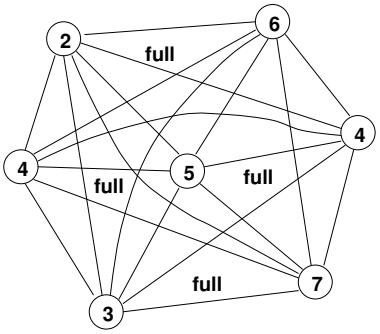

Figure 3. The full graph infrastructure.

6.3 Hierarchy

Starting from any node, the following program, in a repeated advancement in the distributed space, creates oriented links named `hierarchy` from the current nodes to new nodes reached. A new iteration (together with the whole program code) starts in parallel from all nodes reached by the previous iteration.

```
Repeat (
 Create links (
  + 'hierarchy', first come,
  range (20)
 )
)
```

To have the resultant network structured as a tree (with its root in the node the program started), the program allows entering nodes only once, on the first arrival into them. It also tries to establish the next level of hierarchy within certain vicinity (`range`) from the current nodes, to make this hierarchy optimized territorially.

This program, however, may not cover the whole system if the distance between nodes may exceed the range given. In this case, we may decide either to increase the range or select the descendent nodes of the hierarchy among any other nodes that can be reached, at any distance, also allowing us to have only a certain number of subordinate nodes for each node, as follows (see Fig. 4):

```
Repeat (
 Create links (
   + 'hierarchy', first come,
   other nodes, quantity (2)
 )
)
```

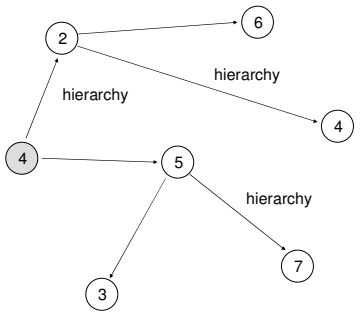

Figure 4. A hierarchical infrastructure.

We can also offer a combined solution where, first, subordinate nodes are tried to be chosen within the range given, and second, if this is not possible, among any nodes reached (directly or indirectly). We may also set up the maximum number of subordinate nodes allowed as the precondition for the both options, with the resultant program being as follows:

```
Repeat (
 Create links (
  + 'hierarchy', first come,
  Or (range (20), other nodes),
  quantity (2)
 )
)
```

Another solution, easily programmable too, may be where the range is floating, gradually increasing unless the needed number of subordinate nodes is found.

6.4 Line

Any other distributed topologies can be created in a similar way. For example, a line connecting all nodes can be just produced by the previous program by allowing only a single new node at each step (let it uses now non-oriented link `line`), as follows (see Fig. 5):

```
Repeat (
 Create links (
  'line', first come,
  Or (range (20), other nodes),
  quantity (1)
 )
)
```

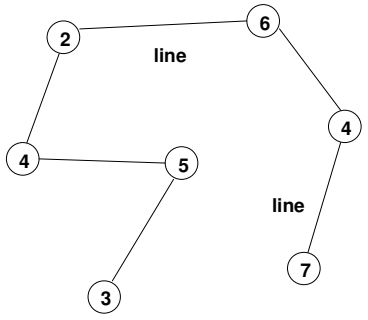

Figure 5. A line infrastructure.

6.5 Ring

With minimal extension, the previous program can create a ring, by connecting the last node of the line with its first node (also changing link names to `ring` and making them oriented), as follows, where the starting node address is always accessible by the special variable `START` (see Fig. 6):

```
Repeat (
 Create links (
  + 'ring', first come,
  Or (range (20), other nodes),
  Quantity (1)
 )
).
Create link (+ 'ring', START)
```

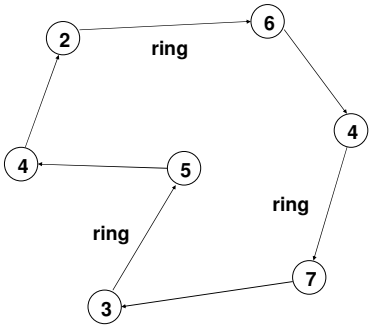

Figure 6. A ring infrastructure.

All these programs can work in a fully distributed way, without any central control, as already mentioned. All the infrastructures above can be built (and can exist) simultaneously between the same nodes, and any changes to them can be easily done at runtime, without interrupting local or global processes which may take place over them.

7 Hierarchical Command and Control

Using the infrastructures built above, we can organize any centralized or distributed system management and control in DSL. Let us show how traditional command and control can work via the hierarchical infrastructure, starting from its root node and using oriented links `hierarchy`. The following program applies recursive command and control procedure (enclosed in braces) in each reached node of the hierarchy, after being delivered there in frontal variable `C2`, as its content.

```
Frontal (C2, Command, Control, Level).
Assign (C2,
 {Increment (Level).
  Detail and apply (Command, Level);
  Detail and apply (Control, Level);
  (Hop (+ 'hierarchy'). Apply (C2))
 }
).
Assign (Command, command scenario).
Assign (Control, control scenario).
Apply (C2)
```

This procedure `C2`, in its turn, applies in parallel (which is indicated by semicolons, see Section 4) the command and control scenarios given from the beginning in frontal variables `Command` and `Control` (these scenarios may be of human, robotic, or mixed orientation). `C2` also simultaneously passes them and itself for a recursive activation and execution in all directly subordinate nodes of the hierarchy, which will be acting in the same way, and so on. At each level of hierarchy (incremented downwards), the original command and control scenarios are detailed for their execution with taking into account peculiarities of this level, as shown in Fig.7.

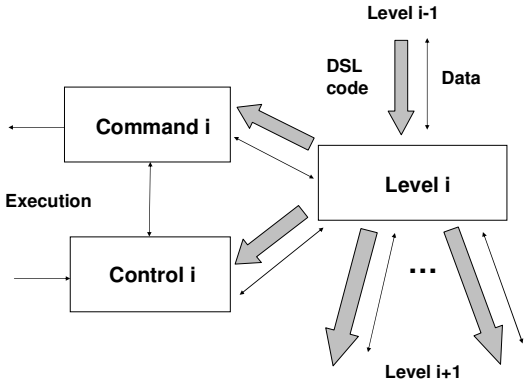

Figure 7. The C2 interpretation in nodes.

The scenarios delivered to, and activated in, nodes can be written in DSL in such a way that could be capable of accessing and sharing data and operations in any nodes of the whole hierarchy, not only in the current one, therefore the organization in Fig. 7 can potentially be much more advanced and flexible than the strictly hierarchical command and control. The program above reflects only a possible organizational skeleton, or framework, that can be set up in DSL.

This program works on the already existing hierarchical infrastructure, where it starts from the root and then covers with activity the whole infrastructure at runtime, beginning to work during, rather than after, being deployed. We can modify this program to be capable of creating the very infrastructure it operates on, during its coverage, not before. We can also form a temporary hierarchy without explicit links (based on internal, invisible interpretation infrastructures) and for the current mission only, with its automatic self-removal after the mission is completed. This latter case can be represented by the following modified program.

```
Frontal (C2, Command, Control, Level).
Assign (C2,
 {Increment (Level).
  Detail and apply (Command, level);
  Detail and apply (Control, Level);
  (Hop (first come, range (20)).
   Apply (C2)
  )
 }
).
Assign (Command, command scenario).
Assign (Control, control scenario).
Apply (C2)
```

This program absorbs mechanisms of the hierarchy-creation programs discussed earlier, with their spatial

repetition effectively expressed now by spatial recursion.

8 Collection and Distribution of Targets

8.1 Using Hierarchical Infrastructure

We can write a spatial program in DSL which self-spreads from the root node to all other nodes of the hierarchical infrastructure, picks up data on all objects (targets) seen in all nodes passed and reached, and merges and fuses all this data while echoing upwards the hierarchy. The data fusion may include removal of duplicate records, as the same targets can be seen from different nodes simultaneously.

Having collected in the root node all targets detected by a distributed system, we can replicate and spread their refined list back to all nodes of the hierarchy by the parallel self-descending process again, allowing each node to select individually targets of inertest from their list and impact those it can. All these operations can be performed on the hierarchical infrastructure by the following simple program, which globally loops in the root node, while repeating the detection-collection-fusion, followed by distribution-selection-impact indefinitely (as shown in Fig. 8).

```
Loop (
 Assign (frontal (Seen),
  Fuse (
   Repeat (
    Free (detect (targets));
    Hop (+ 'hierarchy')
   )
  )
 ).
 Repeat (
  Free (select and impact (Seen));
  Hop (+ 'hierarchy')
 )
)
```

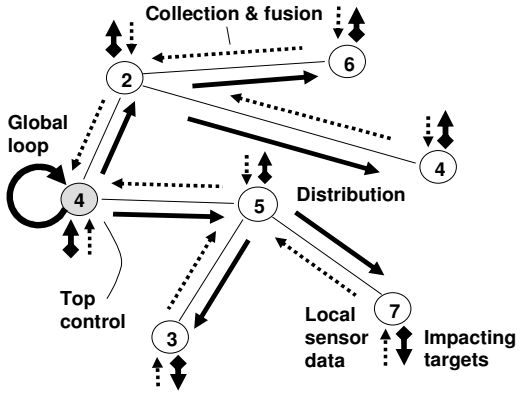

Fig.ure 8. Hierarchical collection and distribution of targets.

8.2 Using Any Infrastructure

Many other efficient solutions of collection and distribution of targets detected by a distributed system can be proposed in DSL. One of them, the most universal and simple, is a fully distributed one, with using any available infrastructure (including the ones we have built before like star, tree, line, ring or their combinations). The idea is in the following.

Each node, fusing what it directly sees with what it gets from other nodes, regularly exchanges all information it accumulates with direct infrastructure neighbors only, updating the records in them if the targets brought are fresher or new. With all nodes doing this in an infinite local loop, the information about all targets seen by all nodes of the distributed system will be eventually reaching all its nodes (of course, if the system is connected in principle), thus guaranteeing global awareness in each node. The following simple program implements this distributed algorithm, additionally setting certain time delay between the iterations in each node (see also Fig. 9).

```
Nodal (Seen).
Hop (all nodes).
Loop (
 Wait (60).
 Fuse and assign (
  Seen, detect (targets)
 ).
 Select and impact (Seen);
 Fuse and assign (
   (Hop (all links). Seen), Seen
 )
)
```

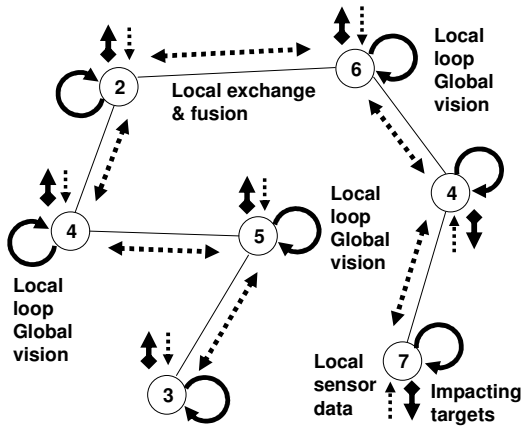

Figure 9. Fully distributed global vision.

This fully distributed algorithm can work well also without any infrastructures built in advance, by merely regularly accessing from all nodes any other nodes in their vicinity (say, within available or given radio or laser range, and not necessarily same neighbors each time, as nodes can move), by substituting the line with the `hop` by the following one:

```
(Hop (range (100)). Seen), Seen
```

These were only the simplest programming examples of dynamic, on the fly, creation of distributed infrastructures and using them for emergent command and control and global vision and targeting. More complex ones can be effectively written too, without any limitations on centralization or, vice versa, distribution of activities (and any their combinations) in distributed systems, with program code optimally staying in nodes or moving between them, cooperatively covering the whole system or its pars with activities needed. And all

this (re)organization can be implemented at runtime, on the active and operating system, which is especially important in crisis situations, where improvisation and flexibility of dealing with emergent resources are the qualities of primary values.

9 Conclusions

A new approach to organization of distributed dynamic systems has been presented which, as was shown, can effectively support and implement basic trends of organization of distributed systems with limited local resources but capable of solving complex global problems. Among these trends are the well known Cooperative Engagement Capability and Ubiquitous Command and Control. Using our approach, the CEC and UC2 systems will also be able to function under indiscriminate damages of their components and infrastructures, while preserving functionality and goal orientation.

The approach presented is based on a special language describing mission scenarios on a semantic level while delegating usual programming routines to automatic interpretation. Communicating interpreters of the language can be installed in internet hosts, mobile robots and cellular phones, integrating any manned and unmanned resources under a unified control. Being both high level and fully formal, the approach may open a real way to a massive use of mobile robotics in advanced crisis relief missions, where job division and subordination between humans and robots may be emergent. Its advantages for robotized crisis management applications may include the following:

- It drastically simplifies the overall control of multi-robot systems, making it comparable to the control of a single robot, regardless of the number of robotic platforms used, which can vary at runtime.
- Does not need any central resources as its global control can start from any available manned or unmanned unit and cover the system at runtime.
- Allows robotized systems to be designed and implemented from the topmost, linguistic, level with considerable reduction of the time and funds needed in comparison with usual bottom up integration.

The previous, or WAVE, version of the technology has been written in C under Unix, and is being used in different countries, especially for intelligent network management, with recent results in Ireland [10, 11] and Canada [12]. The WAVE system is available on the Internet, and can be downloaded, say, from [13]. The current version of the language is in patenting and reimplementation process; it can be installed on any platform within a short time. The universal wave chip, as hardware language interpreter for distributed crisis management systems, is being planned too.

References

[1] 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, USA, pp.29-36 (www.emergencyjournal.com).

[2] K. Tierney, “Disaster Beliefs and Institutional Interests: Recycling Disaster Myths in the Aftermath of 9-11”, Terrorism and Disaster: New Threats, New Ideas, Research in Social Problems and Public Policy, Volume 11, 33—51, 2003 Published by Elsevier Ltd., JSSN: 0196-11 52.

[3] K. Tierney, C. Bevc, and E. Kulikovski, “Metaphors Matter: Disaster Myths, Media Frames, and Their Consequences in Hurricane Katrina”, The Annals of the American Academy, AAPSS, 604, March 2006, pp. 57-81.

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

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

[6] The Cooperative Engagement Capability, John Hopkins APL Technical Digest, Vol. 16, No. 4, 1995. http://www.jhuapl.edu/techdigest/td1604/APLteam.pdf.

[7] Lambert, D.A. (1999a), “Ubiquitous Command and Control”, Proceedings of the 1999 Information, Decision and Control Conference, Adelaide, Australia, pp. 35–40, IEEE. http://www.eleceng.adelaide.edu.au/ieee/idc99/papers/lambert_dale_2.pdf.

[8] Lambert, D.A, and Scholz, J.B. (2005), “A Dialectic for Network Centric Warfare”, Proceedings of the International Command and Control Research and Technology Symposium (ICCRTS). http://www.dodccrp.org/events/2005/10th/CD/papers/016.pdf.

[9] P. Sapaty, “Crisis Management with Distributed Processing Technology”, International Transactions on Systems Science and Applications, vol. 1, no. 1, 2006, UK, pp. 81-92, ISSN 1751-1461 (http://www.xiaglow-institute.org.uk/itssa/itssa.htm).

[10] S. Dragos and M. Collier, “Macro-routing: a new hierarchical routing protocol”, Proc. 47th annual IEEE Global Telecommunications Conference, Texas, USA.

[11] Sanda-Maria Dragos, “Scalable QoS Routing in MPLS Networks Using Mobile Code”, PhD Dissertation, School of Electronic Engineering, Dublin City University, July 2006, 235p.

[12] Gonzalez-Valenzuela, S. and Leung, V.C.M, "QoS-Routing for MPLS Networks Employing Mobile Agents". IEEE Network Magazine, Special Issue, May-June 2002.

[13] Wave system public domain in Canada: http://www.ece.ubc.ca/~sergiog/wavefiles/.