GRASPING SPATIAL SOLUTIONS IN DISTRIBUTED DYNAMIC WORLDS

Peter Simon Sapaty

Institute of Mathematical Machines and Systems
National Academy of Sciences
Glushkova Ave 42, 03187 Kiev, Ukraine

ABSTRACT

A high-level ideology and technology will be revealed that can effectively convert any distributed system (manned, unmanned or mixed) into a globally programmable spatial machine capable of operating without central resources. Compact mission scenarios in a special high-level language can start from any point, runtime covering & grasping the whole system or its parts needed, setting operational infrastructures, and orienting local and global behavior. The approach offered can be particularly useful for quick reaction on asymmetric situations and threats the world is facing, paving the way to massive use of cooperative robotics and gradual transition to unmanned systems for solving critical problems in unpredictable environments

Keywords

Distributed dynamic worlds, asymmetric situations and threats, Spatial Grasp Technology, Distributed Scenario Language, parallel networked interpretation, multi-robot systems, holism, gestalt theory.

1. INTRODUCTION

In our modern dynamic world we are facing numerous irregular situations and threats, where proper reaction on them could save lives and wealth and also protect critical infrastructures and key national and international resources. For example, no secret that world's most powerful armies with traditional, classical system organizations are often losing to terrorists or piracy with limited numbers and primitive gadgets but with very flexible organizational structures making them hard to detect and fight. And delayed reaction on earthquakes or tsunamis may also be considered as a result of inadequacy of existing system organizations.

A novel philosophy and supporting high-level networking technology will be described that can quickly react on irregular situations and threats and assemble any available human and technical resources into highly operable systems providing global awareness and will, pursuing global goals, and self-recovering from indiscriminate damages. The approach offered allows us at runtime, on the fly, formulate top semantics of the needed reaction on asymmetric events in a special Distributed Scenario Language (DSL), while omitting insignificant details and shifting most of traditional organizational routines to automated up to fully automatic implementation under a unified command and control, with effective engagement of advanced unmanned systems.

The details of this paradigm based on gestalt and holistic principles [1] rather than traditional multi-agent organizations [2,3] will be revealed in this paper, with explanation of different DSL scenarios that can be effectively executed by self-organized robotic groups with minimum external intervention. These scenarios will include collective navigation of distributed spaces, individual and simultaneous coastline patrols, collective outlining and impacting of forest fire zones, and a possible swarm-against-swarm solution where highly organized robotic swarms can fight another (manned including) groups, related, say, to piracy activities.

The technology offered can provide a unified solution to human-robot interaction and organization of effective multi-robot behaviors -- just as a derivative of automatic parallel and distributed interpretation of high-level system scenarios in DSL in networked environments.

2. TRADITIONAL SYSTEM ORGANIZATIONS AND THEIR PROBLEMS

2.1 From system structure to system function

The traditional approach to system design, development and management is when the system structure and system organization are primary, created in advance, and global function with overall behavior are secondary, as in Figure 1.

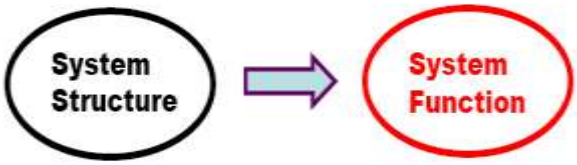


Figure 1. Traditional approach to system design.

Typical examples of the traditional approach are multi-agent organizations [2,3], where global system behavior is the result of the work and interaction of predetermined parts (agents). In this respect we can name the 4D/RCS Model Architecture [4], with its block diagram shown in Figure 2.

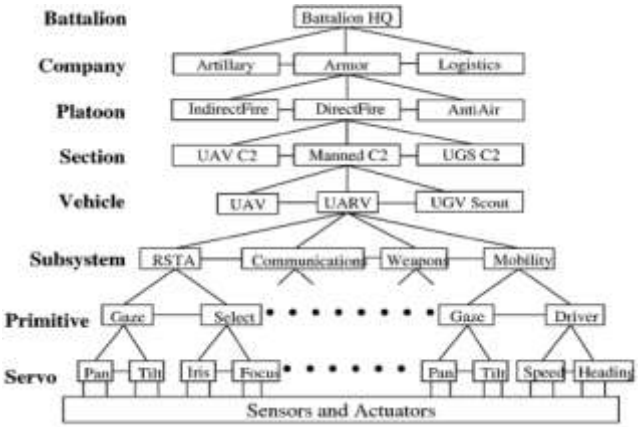


Figure 2. 4D/RCS model architecture.

4D/RCS prescribes a hierarchical control principle, where commands flow down the predefined hierarchy, and status feedback and sensory information flows up. Large amounts of

communication may also occur between nodes at the same level, particularly within the same subtree of the command tree. Future Combat Systems (FCS) project [5] was ideologically and technologically based on this organizational (as well as artificial intelligence) hierarchical model.

2.2 The problems with classical organizations

The related systems, where we first formalize and build the system structure and organization and then try to get from these the global behavior needed, are usually *static*, and may often fail to adapt to highly dynamic and asymmetric situations. If the initial goals change, the whole system may have to be partially or even completely redesigned and reassembled. Adjusting the already existing system to new goals and functionality needed may result in a considerable *loss of system's integrity and performance*.

3. AN ALTERNATIVE: SPATIAL GRASP TECHNOLOGY (SGT)

3.1 SGT basic idea

Within the approach offered, also coined and known as *"over-operability"* [6,7] in contrast to the conventional *interoperability*, the global function and overall behavior are considered as much as possible to be primary. Whereas system structure and organization (command and control including) are secondary, with the latter as a derivative of the former, as in Figure 3.

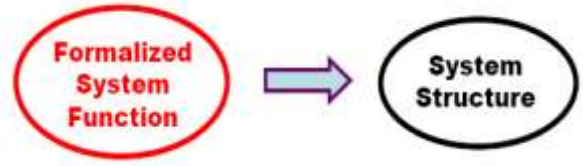


Figure 3. SGT basic idea of system creation and organization.

The advantages of this (actually the other way round) approach include *high potential flexibility* of runtime system creation, organization and modification, especially in *quick responses to asymmetric events* and enhanced opportunities for automated up to *fully automatic* (unmanned) solutions.

3.2 Parallel spatial grasp of distributed worlds

SGT is based on a formalized controlled seamless navigation, coverage, penetration, and grasping of distributed physical and virtual spaces, as symbolically shown in Figure 4.

Figure 4. Incremental integral grasping of distributed worlds.

This top mode of system vision has strong psychological and philosophical background, reflecting, for example, how humans (esp. top commanders) mentally plan, comprehend, and control complex operations in distributed environments.

3.3 Distributed scenario interpretation

The approach in practice works as follows. A network of universal control modules U, embedded into key system points, collectively interprets system scenarios expressed in DSL, as shown in Figure 5. These scenarios, based on the spatial grasp idea (and capable of representing any parallel and distributed algorithms, spatial cycling & branching including), can start from any node, subsequently covering and matching the system at runtime.

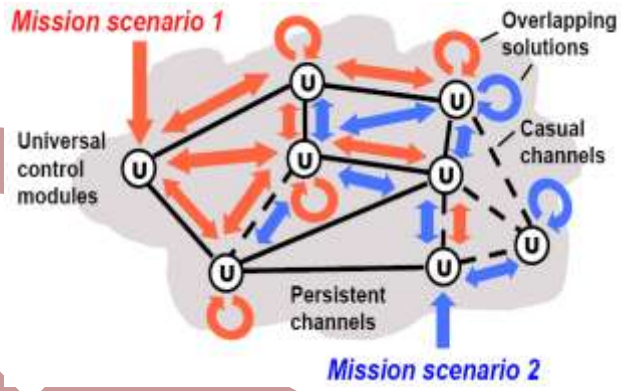


Figure 5. Scenario execution in dynamic environments.

DSL scenarios are very compact and can be created on the fly. Different scenarios can cooperate or compete in a networked space (depending on real control or distributed simulation mode) as overlapping fields of solutions. Self-spreading scenarios can create runtime knowledge infrastructures distributed between system components (robots, sensors, humans). These infrastructures can effectively support distributed databases, command and control, situation awareness, autonomous decisions, as well as any other computational or control models.

More details on the SGT, its core language DSL (including the predecessor version WAVE), and its distributed interpreter can be found elsewhere [8-19], with some key features necessary for explanation of the programmed application examples throughout this paper briefed in the following sections.

4. DISTRIBUTED SCENARIO LANGUAGE, DSL

DSL differs radically from traditional programming languages, allowing us to directly move through, observe and make any actions and decisions in fully distributed environments, both virtual and physical. DSL directly operates with:

- *Virtual World* (VW), finite and discrete, consisting of nodes and semantic links between them.
- *Physical World* (PW), infinite and continuous, where each point can be identified and accessed by physical coordinates.
- *Virtual-Physical World* (VPW), finite and discrete, similar to VW but associating virtual nodes with certain PW coordinates.

4.1 DSL basic features

Any sequential, parallel, centralized, distributed, stationary or mobile algorithm operating with information and/or physical

matter can be written in DSL. Its top level recursive structure is shown in Figure 6.

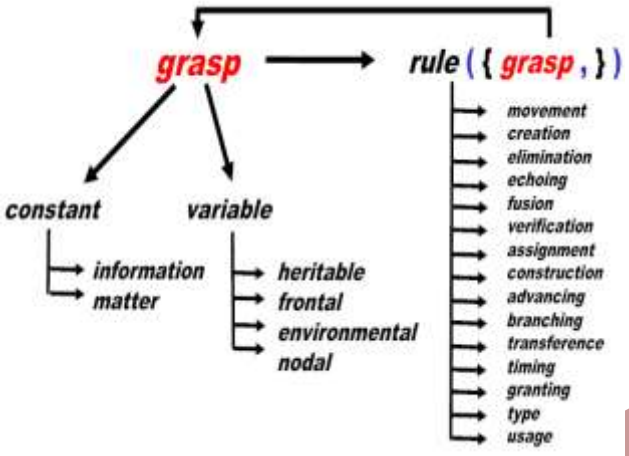


Figure 6. DSL top level syntax.

DSL main features may be summarized as follows:

- A DSL scenario develops as parallel transition between sets of progress points (*props*).
- Starting from a prop, an action may result in other props.
- Each prop has a resulting *value* and a resulting *state*.
- Different actions may evolve *independently or interdependently* from the same prop.
- Actions may also spatially *succeed each other*, with new ones applied in parallel from all props reached by the previous actions.
- Elementary operations may directly use values of props obtained *from other actions* whatever complex and remote.
- Any prop can associate with a *node* in VW or *position* in PW, or *both* -- when dealing with VPW.
- Any number of props can be simultaneously linked with the same points of the worlds.
- Staying with the world points, it is possible to *directly access and impact* local world parameters, whether virtual or physical.

4.2 DSL rules

The basic construct, or *rule*, of the language may represent any action or decision and can, for example, be as follows:

- Elementary arithmetic, string or logic operation.
- Hop in a physical, virtual, or combined space.
- Hierarchical fusion and return of (remote) data.
- Distributed control, both sequential and parallel.
- A variety of special contexts for navigation in space, influencing enclosed operations and decisions.
- Type or sense of a value, or its chosen usage, guiding automatic interpretation.
- Creation or removal of nodes and links in distributed knowledge networks.

4.3 Spatial variables in DSL

Working in fully distributed physical or virtual environments, DSL has different types of variables, called *spatial*, effectively serving multiple cooperative processes:

- *Heritable variables* – these are starting in a prop and serving all subsequent props which can share them in both read & write operations.
- *Frontal variables* – are an individual and exclusive prop's property (not shared with other props), being transferred between the consecutive props, and replicated if from a single prop a number of other props emerge.
- *Environmental variables* – are accessing different elements of physical and virtual words when navigating them, also a variety of parameters of the internal world of DSL interpreter.
- *Nodal variables* – allow us to attach an individual temporary property to VW and VPW nodes, accessed and shared by all props associated with these nodes.

These types of variables, especially when used together, allow us to create spatial algorithms working *in between components* of distributed systems rather than *in* them, thus allowing for flexible, robust, and capable of self-recovery solutions, even though different components may indiscriminately fail. Such algorithms can freely move in distributed processing environments (partially or as an organized whole), always preserving global integrity and overall control, if needed.

4.4 Main DSL constructs

The main language constructs are as follows.

grasp → *phenomenon* | *rule* ({ *grasp* , })
phenomenon → *constant* | *variable* | *special*
constant → *information* | *matter*
variable → *heritable* | *frontal* | *environmental* | *nodal*
rule → *movement* | *creation* | *elimination* | *echoing* | *fusion* | *verification* | *assignment* | *construction* | *advancement* | *branching* | *transference* | *timing* | *granting* | *type* | *usage*
Information → '*string*' | **{***string* **}** | *number*
matter → "*string*"
movement → hop | hop links | move | shift
creation → create | linkup
elimination → delete | unlink | remove
echoing → order | rake | min | max | sort | sum | average | product | count | state
fusion → add | subtract | multiply | divide | degree | separate | unite | attach | append | common | content | index | rand
verification → equal | not equal | less | less or equal | more | more or equal | empty | nonempty | belongs | not belongs | intersects | not intersects
assignment → assign | assign peers
construction → inject | replicate | split | partition | select
advancement → advance | advance sync | repeat | repeat sync
branching → parallel | sequence | if | while | or| or parallel | and | and parallel |

		cycle | loop | whirl
transference	→	run | call | output | input
timing	→	sleep | remain
granting	→	free | release | quit | none | lift | stay | grasp
type	→	nodal | heritable | frontal | environmental | info | matter | number | string
usage	→	address | name | place | center | range | time | speed | doer | node | link | unit
heritable	→	H { *alphameric* }
frontal	→	F { *alphameric* }
nodal	→	N { *alphameric* }
environmental	→	TYPE | CONTENT | ADDRESS | QUALITIES | WHERE | BACK | PREVIOUS | LINK | DIRECTION | WHEN | TIME | SPEED | STATE | VALUE | COLOR | OUT
special	→	abort | thru | done | fail | any | first | last | random | all | in | out | infinite | nil | virtual | physical | combined | global | local | direct | no back

4.5 Elementary DSL programming examples

Some DSL programming examples are shown in Figure 7, with assignment of a sum of three values to a variable, parallel hop into two physical locations, creating a new node in a virtual space, and the network extension with a new link-node pair.

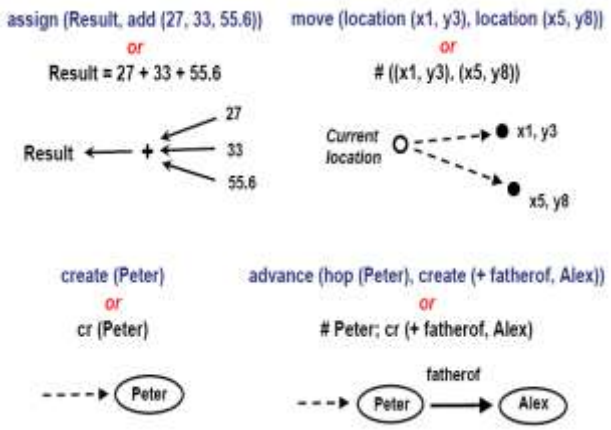


Figure 7. Elementary examples in DSL.

Traditional abbreviations of operations and usual delimiters can be used too, in order to shorten DSL programs (always remaining, however, within the general recursive syntactic structure shown in Figure 6).

5. THE DSL INTERPRETER

5.1 Distributed interpreter organization

The DSL interpreter consists of specialized modules (which can work in parallel) handling and sharing specific language interpretation data structures [13, 17-19], as shown in Figure 8.

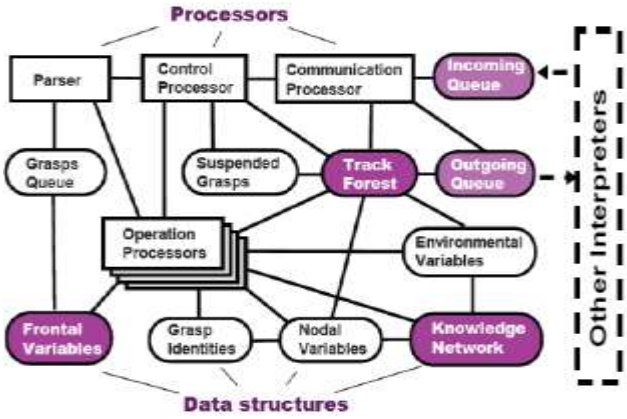
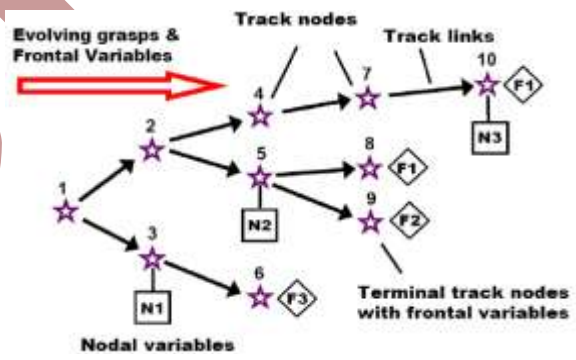


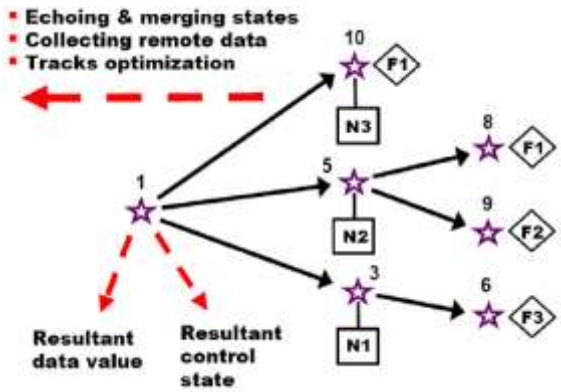
Figure 8. Organization of DSL interpreter.

The network of the interpreters can be mobile and open, changing the number of nodes and communication structure at runtime. Communicating copies of the interpreter can be concealed, if needed (say, for operation in hostile environments).

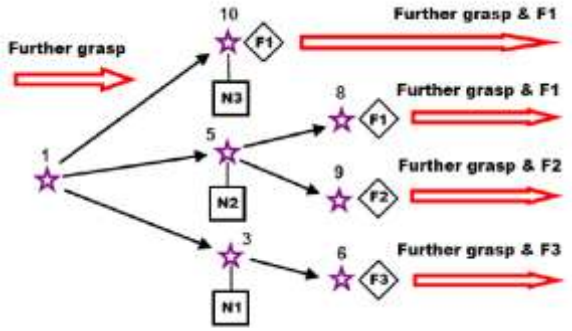
The heart of the distributed interpreter is its spatial *track system* (Figure 9). The dynamically crated track forests are used for supporting (or removing) spatial variables and echoing & merging different types of control states and remote data. Being self-optimized in the echo processes, the track forests are dynamically covering the systems in which DSL scenarios evolve, keeping the overall (parallel and distributed) process integrity as well as providing local and global control. They also route further grasps (as spatial waves) to the positions in physical, virtual or combined spaces reached by the previous grasps, uniting them with the frontal variables left there by preceding grasps.



(a)



(b)



(c)

Figure 9. Distributed track system: (a) forward operations; (b) backward operations with tracks optimization; (c) forwarding further grasps.

6. INTEGRATING WITH ROBOTIC FUNCTIONALITY

6.1 Installing DSL interpreters in robotic units

Installing DSL interpreters (as universal modules U, see Figure 10) in mobile robots (ground, aerial, surface, underwater, space, etc.) allows us to organize effective group solutions (incl. any swarming) of complex problems in distributed physical spaces in a clear and concise way, effectively shifting traditional management routines to automatic levels. Human-robot interaction and gradual transition to fully unmanned systems are essentially assisted too.

Any groups of manned-unmanned devices with DSL interpreters implanted into them, with any communication networks in between can serve as *spatial machines* capable of doing any jobs together under a unified control automatically emerging from high-level DSL scenarios.

Figure 10. Examples of cooperative robotic scenario skeletons.

6.2 Semantic level tasking

By embedding DSL interpreters into robotic vehicles we can task them on a higher, *semantic* level, skipping numerous traditional details of management of them as a group, fully delegating these to an automatic solution. An exemplary semantic level tasking may look like follows.

Go to physical locations of the disaster zone with coordinates: (x1, y1), (x2, y2), (x3, y3), evaluate radiation level at each location, return its maximum value with attached exact coordinates of the respected location to the headquarters, and launch from the latter a massive cleanup operation at the location found.

The corresponding DSL program will strictly follow this scenario:

```
Location =
    maximum (move (x1_y1, x2_y2, x3_y3);
                attach (evaluate (radiation), WHERE));
move (Location : 2); massive_cleanup (radiation)
```

This (inherently parallel and fully distributed) scenario can be executed with any available number of mobile robots (practically reasonable: from one to four), and the number of robots may change at runtime. Distributed DSL interpreter automatically creates the needed operational and command and control infrastructures of the robotic group and guarantees full task execution under any variations [10, 12, 13, 16].

6.3 Programming explicit behavior level

After embedding DSL interpreters into robotic vehicles, we can also provide any needed detailed collective behavior of them (say, at a lower than top task level as before)—from *loose swarm* to a *strictly controlled integral unit* obeying external orders. Any mixture of different behaviors within the same scenario can be easily programmed too. Expressing different scenarios in DSL and their integration into a more complex, combined one may look like follows.

- Swarm movement, starting from any unit (naming it as **swarm_move**):

```
hop (allnodes);
Limits = (dx (0, 8), dy (-2, 5)); Range = 500;
repeat (Shift = random (Limits);
        if (empty (hop (Shift, Range), move (Shift)))
```

- Finding topologically central unit and subsequently hopping into it, starting from any unit (naming it as **find_hop_center**):

```
Frontal (Aver) = average (hop (allnodes); WHERE);
hop (min (hop (allnodes);
          distance (Aver, WHERE) & ADDRESS) : 2)
```

- Creating runtime infrastructure of the robotic group, starting from the central unit found (naming it as **infra_build**):

```
stay (repeat (linkup (+infra, first, Depth)))
```

- Targets collection & distribution & impact, starting from the central unit found (naming it as **collect_distribute_impact**):

```
loop (nonempty (frontal (Seen) =
        repeat(free(detect(targets)), hop(+infra)));
      repeat (free (select_move_shoot (Seen)),
              hop(+infra)))
```

- Removing previous infrastructure (for creating a new, updated one), starting from any unit (naming it as **infra_remove**):

stay (hop (allnodes); remove (alllinks))

- Resultant combined solution (integrating previous DSL programs named), starting from any unit:

```
parallel (
  swarm_move,
  repeat (find_hop_center;
          infra_remove; infra_build;
          orparallel (collect_distribute_impact,
                      sleep (delay))))
```

The obtained resultant scenario combines loose randomized swarm movement in a distributed space with periodic finding & updating its topologically central unit and setting-updating runtime hierarchical infrastructure between the units (as physical distances between units can change). The latter infrastructure controls observation of the distributed territory, collecting potential targets on its whole while distributing them back to the vehicles, with subsequent selection and impacting suitable targets individually by the units. A related snapshot, say, for aerial vehicles, is shown in Figure 11. More on this integral scenario can be found in [13, 16].

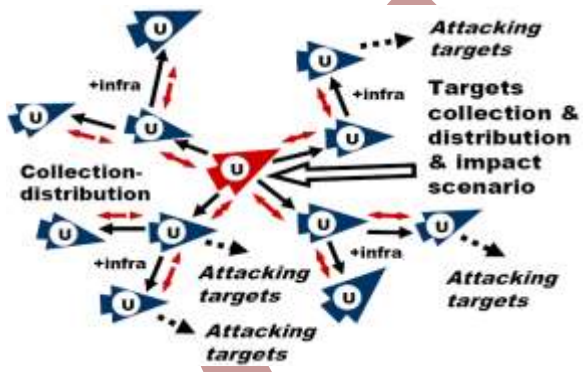


Figure 11. Collecting, disseminating and impacting targets by an unmanned aerial team using dynamically created and updated command and control infrastructure, while moving altogether as a loose swarm.

7. PATROLLING COASTAL WATERS

This scenario may be suitable for both surface and varying depth underwater search of intrusions in the coastline zone, but for simplicity we will be assuming here only two-dimensional space to be navigated. At the beginning let us create a coastal waypoint map in the form of a semantic network, as in Figure 12 (where r is chosen as an arbitrary name of links between nodes-waypoints).

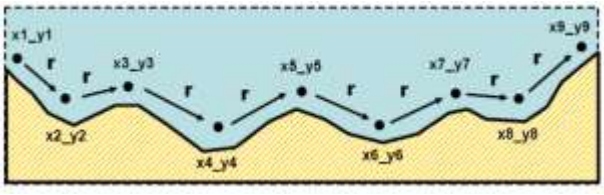


Figure 12. Coastal waypoint map.

The corresponding DSL solution is as follows.

create (*#x1_y1*; +r*#x2_y2*; +r*#x3_y3*; ... +r*#x9_y9*)

A single USV (or UUV) solution repeatedly navigating all coastal area (in both directions) by the map created is shown in Figure 13 and by DSL program that follows (searching the water space for alien objects by the *depth* available by vehicle's sensors).

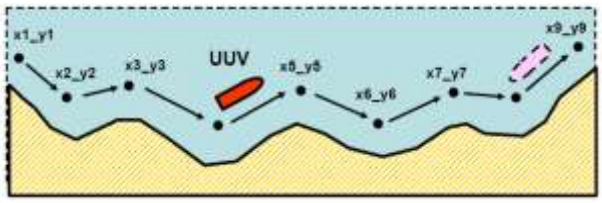


Figure 13. Patrolling coastal waters with a single vehicle.

```
move (hop (x1_y1)); R = +r;
repeat (repeat (move (hop (R)); check_report (depth));
        invert (R))
```

A two-vehicle parallel solution is shown in Figure 14 and by the following program, with vehicles moving according to the coastal map independently, assuming each having embedded automatic mechanisms and procedures for avoiding possible collisions with the other vehicle.

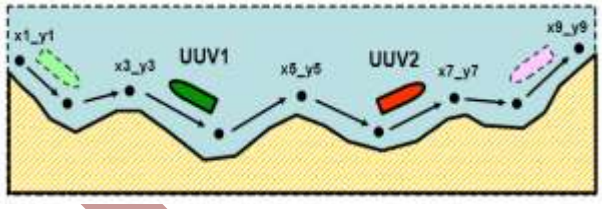


Figure 14. Patrolling coastal waters with two vehicles.

```
(move (hop (x1_y1)); R = +r),
(move (hop (x9_y9)); R = -r);
repeat (repeat (move_avoid (hop (R));
                check_report (depth));
        invert (R))
```

Another solution for the two-vehicle case may be when each vehicle turns back if discovers another patrol vehicle on its way, checking for this its vicinity by *depth2*).

```
(move (hop (x1_y1)); R = +r),
(move (hop (x9_y9)); R = -r);
repeat (repeat (none (depth2); move (hop(R));
                check_report (depth));
        invert (R))
```

For the both latter cases, the whole coastline will always be searched in full if at least a single vehicle remains operational.

8. FIGHTING FOREST FIRES

We will consider a solution where distributed physical space is randomly searched by simultaneous propagation of multiple reconnaissance units, which when discover irregularities (e.g. forest fires) move towards and encircle the corresponding zones, collect their perimeter coordinates, transfer them to the headquarters (HQ), and ultimately initiate massive impact on the zones. The zones with fires and initial positions of reconnaissance units are shown in Figure 15, and intermediary positions of the robotic unites moving randomly-oriented (repeatedly shifting their positions within certain coordinate sector) are in Figure 16.

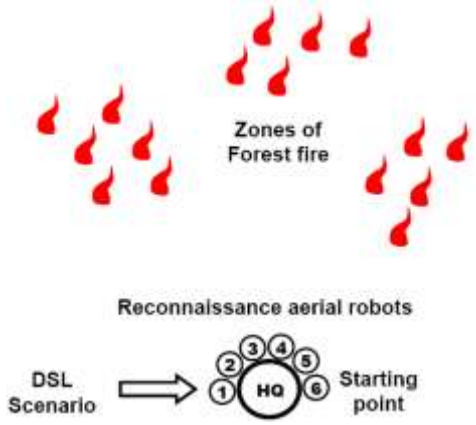


Figure 15. Initial fire fighting scenario injection.

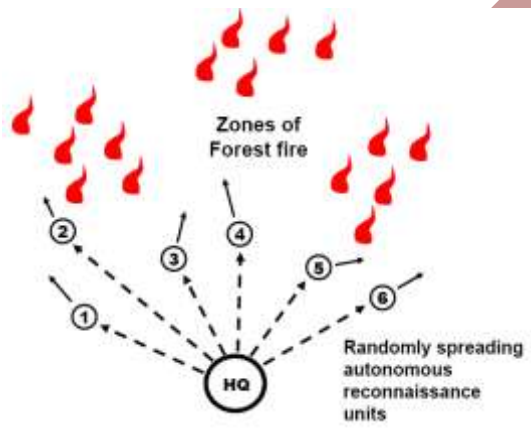


Figure 16. Randomized robots movement.

After detecting fire locations, the reconnaissance units that reached them begin moving around the fire zones, having initially randomly chosen the encirclement direction (i.e. clockwise or anticlockwise). In each step they accumulate coordinates of the periphery of fire zones, and upon termination of the encirclement send the completed zone coordinates to the headquarters (HQ). Getting the latter, the HQ is launching a massive direct impact on the zones outlined, as shown in Figure 17, which operation may be manned, unmanned, or mixed. The full DSL scenario for this task may be as follows.

```
move (HQ); create_nodes (1, 2, 3, 4, 5, 6);
repeat (
  shift (random (limits));
  if (check (fire),
     (Zone = WHERE;
      Direction = random (clockwise, anticlockwise);
      repeat (
        move_around (fire, Direction, depth);
        append (Zone, WHERE);
        if (distance (WHERE, Zone : 1) < threshold,
          (hop (HQ); impact (Zone); done))))))
```

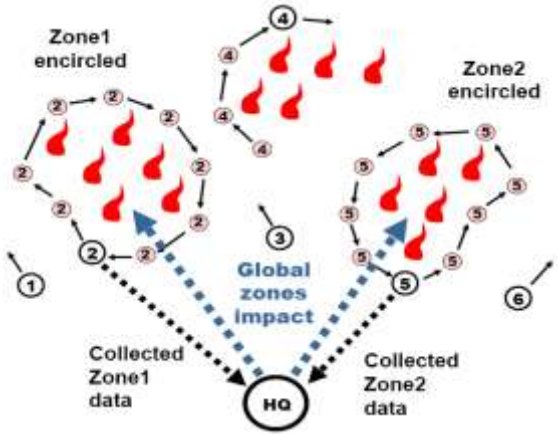


Figure 17. Encircling fire zones followed by global impact on them.

Other interpretations of this scenario may be dealing with radiation or environment pollution zones, terrorist activities zones, fish concentration, etc., with aerial, ground, surface or underwater robots engaged.

9. SWARM AGAINST SWARM SCENARIO

As a more complex scenario example in DSL, we will consider here the case where an unmanned swarm is opposing other (possibly, manned) group/swarm, as in Figure 18. This may relate, for example, to fighting piracy in maritime environment or to the air and missile defense where (aerial, surface and/or underwater) unmanned vehicles, working cooperatively under a unified control as an integral global-goal-oriented unit, are used for withstanding multiple hostile activity.

Figure 18. Fighting group targets with unmanned swarms.

Main features of the scenario considered are as follows:

- Initial launch of the swarmed chasers (with DSL interpreters embedded), which can communicate with each other, into the expected hostility area.
- Discovering targets and forming their priority list by their positions in physical space, where maximum priority is assigned to topologically central targets as potential control units of the intruders.
- Other targets are sorted by their distance from the topological center of their group, estimated previously.

- The most peripheral targets (those in maximum distance from the topological center, as potentially having more chances to escape) are considered of highest priority too.
- Assigning available chasers to targets, classifying the former as engaged, with chasing and neutralizing targets, and subsequently returning the chasers into status free after performing the mission.
- The vacant chasers are again engaged in the targets selection & impact procedure.

This entire swarm-against-swarm scenario may be expressed in DSL in a very compact manner, as follows.

```
frontal (Next);
sequence (
 start_launch (all_free_chasers, targets_area),
 repeat (
    hop (any_free_chaser);
    All_targets = merge (hop (all_free_chasers);
      coordinates (targets_seen));
      nonempty (All_targets);
    Center = average (All_targets);
    List = min_max_sort (split (All_targets);
       attach (distance (VALUE, Center), VALUE);
    List = append (withdraw (List, last), List);
    loop (nonempty (List);
      Next = element (withdraw (List, first), second);
      Chaser =
        Element (min (hop (all_free_chasers);
        attach (distance (WHERE, Next), ADDRESS),
                second);
      release (hop (Chaser); STATUS = engaged;
                pursue_investigate_neutralize (Next);
                STATUS = free))));
```

It is worth mentioning that all the chaser swarm management expressed explicitly or induced automatically by the above program is done exclusively within the swarm itself, without any external intervention, which dramatically simplifies external control of this multi-robot operation.

10. OTHER APPLICATIONS

Many other applications of the presented paradigm can be found in the previous publications, including the ones cited above. Some of the researched and reported areas are as follows.

- **Emergency Management**. Using the interpreters installed in massively wearable devices may allow us to assemble workable systems from any wreckage after the disasters, using any remaining communication channels, manual including, thus saving lives and property.
- **Directed Energy Systems.** The technology can provide high flexibility in organizing directed energy systems and weapons, especially in asymmetric situations, making automatic distributed decisions with the "speed of light" too.
- **Distributed Avionics.** Implanting the interpreter copies into main control points of the aircraft may provide a higher, intelligent layer of its self-analysis and recovery, by the spreading recursive scenarios starting from any point and collecting & fusing key data from other points.
- **Sensor Networks.** Wireless sensors may be dropped from the air massively, as "smart dust". With a limited communication range, they must operate in a network to perform non local jobs in a distributed environment. The technology offered can convert their emergent networks into a universal parallel computer operating in DSL.
- **Battlespace Dominance.** The technology can provide superiority over an adversary by flexible runtime adaptable system organizations rather than troop numbers or power of individual weapons, countering asymmetric situations and threats by highly asymmetric solutions itself.
- **Societal Engagement and Support of Elders.** The percentage of older population is growing quickly, especially in developed countries. The technology offered, effectively integrating distributed physical and virtual worlds, can solve a variety of important problems -- from smart home to smart city to smart nation -- in continuing support of elders and their fruitful engagement in social life.

11. CONCLUSIONS

A brief summary of advantages of the approach offered may be as follows.

- The Spatial Grasp ideology and technology can dramatically simplify application programming in distributed dynamic systems.
- Setting multi-robot solutions in DSL may often be comparable in complexity to routine data processing in traditional languages.
- External management of multi-robot systems may not depend on the number of components in them due to their internal self-organization and automatic command and control inside robotic groups.
- Formalization of mission scenarios in DSL can make human-robot interaction and transition to fully unmanned systems natural and straightforward.
- Spatial swarm intelligence in DSL can successfully compete with human collective intelligence, outperforming the latter in time critical situations.

In addition to the features listed above, we can state that in comparison with other systems, DSL interpreter represents an *embedded universal intelligence* common to all applications. Any scenario can be executed by a network of such intelligences, including the ones on topmost levels. In other approaches, most of the system intelligence has to be programmed *explicitly for each application*, thus enormously complicating mission planning, organization and management.

All communications among unmanned units, also between manned and unmanned ones, are on a high, semantic level in DSL. They are *very compact* (often hundreds of times shorter than in other languages) which may be crucial for maritime (especially underwater) operations with casual and limited data channels.

12. REFERENCES

[1] Wertheimer M. 1924. Gestalt theory, Erlangen, Berlin

[2] Minsky, M. 1988. The society of mind, Simon and Schuster, New York.

[3] www.wikipedia.org/wiki/Multi-agent_system

[4] 4D/RCS: A reference model architecture for unmanned vehicle system, version 2.0. 2002. Report NIST.

[5] Feliciano, C.N. 2009. The army's future combat system program (Defense, Security and Strategy Series), Nova

Science Pub Inc.

[6] Sapaty, P.S. 2002. Over-operability in distributed simulation and control, The MSIAC's M&S Journal Online, Winter Issue, Volume 4, No. 2, Alexandria, VA, USA.

[7] Sapaty, P. 2009. The over-operability organization of distributed dynamic systems for asymmetric operations, Proc. IMA Conference on Mathematics in Defence, Farnborough, UK.

[8] Sapaty, P. 2012. Withstanding asymmetric situations in dxistributed dynamic worlds, invited paper, Proc. 17th International Symposium on Artificial Life and Robotics (AROB 17th '12), B-Con Plaza, Beppu, Oita, Japan.

[9] Sapaty, P. 2012. Distributed air & missile defense with spatial grasp technology", Intelligent Control and Automation, Scientific Research, Vol.3, No.2.

[10] Sapaty, P. 2011. Spatial grasp technology for high-level management of distributed unmanned systems, Unmanned Systems Asia 2011, Singapore.

[11] Sapaty, P., Sugisaka, M. 2011. Advanced networking and robotics for societal engagement and support of elders, International Symposium on Artificial Life and Robotics, AROB 16th '11, Beppu, Oita, Japan.

[12] Sapaty, P.S. 2011. Seeing and managing distributed spaces using maritime unmanned systems, GLOBAL OPV and Maritime Unmanned Systems Summit, Dedeman Hotel, Istanbul, Turkey.

[13] Sapaty, P. 2011. Meeting the world challenges with advanced system organizations, Informatics in Control Automation and Robotics, Lecture Notes in Electrical Engineering, Vol. 85, 1st Edition, Springer.

[14] Sapaty, P., Kuhnert, D., Sugisaka, M., Finkelstein, R. 2009. Developing high-level management facilities for distributed unmanned systems, Proc. Fourteenth International Symposium on Artificial Life and Robotics (AROB 14th'09), B-Con Plaza, Beppu, Oita, Japan.

[15] Sapaty, P., Morozov, A., Finkelstein, R., Sugisaka, M., Lambert, D. 2008. A new concept of flexible organization for distributed robotized systems, Artificial Life and Robotics, Volume 12, Numbers 1-2, ISSN: 1433-5298 (Print) 1614-7456 (Online), Springer Japan.

[16] Sapaty, P. 2008. Distributed technology for global dominance, Proc. of SPIE, Volume 6981, Defense Transformation and Net-Centric Systems 2008, 69810T.

[17] Sapaty, P. 2005. Ruling distributed dynamic worlds, John Wiley & Sons, New York.

[18] Sapaty, P. 1999, Mobile processing in distributed and open environments, John Wiley & Sons, New York.

[19] Sapaty, P. 1993. A distributed processing system, European Patent No. 0389655, Publ. 10.11.93, European Patent Office.

Dr Peter Sapaty, Chief Research Scientist, Director of Distributed Simulation and Control at the Ukrainian Academy of Sciences, is with networking for 45 years. Worked in Germany, UK, Canada and Japan as Alexander von Humboldt researcher, project leader, visiting and special invited professor, also chaired a SIG on Mobile Cooperative Technologies within Distributed Interactive Simulation project in the US. Peter invented high-level distributed control technology used in different countries and resulted in a European Patent and two John Wiley books, with the third one in progress. Published more than 150 scientific papers worldwide on distributed system organizations. Current areas of interest: advanced command & control and models and languages for coordination and simulation of distributed dynamic systems with application in cooperative robotics, emergency and battlefield management, infrastructure protection, as well as counterterrorism. His bio is in Marquis Who's Who in the World and Cambridge Outstanding Intellectuals of the 21st Century.