High-Level Management of Distributed Air and Missile Defence Systems

Peter Simon Sapaty

Institute of Mathematical Machines and Systems
National Academy of Sciences
Glushkova Ave 42, 03187 Kiev Ukraine
Phone:+380-67-4199223; Fax: +380-44-5266457
sapaty@immssp.kiev.ua, peter.sapaty@gmail.com
The world is "dangerously unprepared" for future disasters

The international development secretary
Andrew Mitchell
BBC News, 27 December 2011
Rationale

A novel high-level ideology and technology will be revealed that can effectively convert any distributed system (manned, unmanned or mixed) into a universal globally programmable spatial machine capable of operating without central resources, self-optimizing and self-recovering from indiscriminate damages.

Integral mission scenarios in a special Distributed Scenario 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.

Air & missile defense based on these principles can gain a new qualitative dimension allowing us to effectively withstand asymmetric situations and threats and defeat potential adversaries.
Contents

1. History Points on Distributed Control Technology Developed
2. Air & Missile Defense as Large Distributed Systems
3. Traditional System Organizations and Their Problems
4. An Alternative: Spatial Grasp Technology (SGT)
5. Distributed Scenario Language (DSL)
6. DSL Networked Interpreter
7. DSL Programming
8. Collective Robotics
9. Air & Missile Defense under Spatial Grasp Model
10. European Missile Defense Scenarios
11. Possible Nonlocal Nuclear Scenarios
12. Conclusions
13. References
1 History Points on Distributed Control Technology Developed

- Power networks: 66-71
- Surface-to-air missile control: 65-75
- Distributed computer networks: 69-79
- Parallel computations, mobile agents: 71-80
- Nationwide global networked management system: 75-80
- Intelligent hardwired computers: 70-80
- Distributed macro-pipeline supercomputer: 78-85
- Wave model of distributed parallel computations: 74-90
- Intelligent network management project for Siemens: 90-93
- Distributed simulation of battlefields UK, DIS project US: 93-98
- WAVE system in Germany, UK, US, Canada: 90-00
- Cooperative robotics, Japan: 01-05
- European patent: 90-93
- John Wiley books: 99, 05, new one in progress
Air and missile defense capabilities are growing globally. These systems are usually distributed on large territories, consist of many interacting elements (from sensors to shooters), and are expected to work in complex conditions to effectively protect national and international infrastructures and withstand unpredictable events.

Their efficient organization needs introduction of radically new philosophies, models and technologies that can operate in extremely dynamic, uncertain, and dangerous situations.

“Blowing a balloon with emergent holes” comparison. Global 100% safety solutions required, especially in cases with nukes.

Growing role of advanced Unmanned Systems and Directed Energy Weapons.
3 Traditional System Organizations and Their Problems
System Structure -- First

- System structure & organization – primary
- Global function & overall behavior – secondary, as a result of the predefined structure
The cerebellum is a region of the brain that plays an important role in motor control. It is also involved in cognitive functions such as attention and language. It receives input from sensory systems and from other parts of the brain and spinal cord, and integrates these inputs to fine tune motor activity.
Real Time Control System (RCS) was inspired 30 years ago by a model of the cerebellum.

It contains many layers of computational nodes each containing elements of sensory processing (SP), world modeling (WM), value judgment (VJ), and behavior generation (BG).
Multi-agent organizations, including the ones in knowledge processing, are based on structure first philosophy too, expected to produce the required external functionality by inter-agent interactions.
Problems with Traditional System Organizations

- The related systems are often clumsy and static, prone to failures in 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, also new environments, may result in a considerable loss of system’s integrity and performance.
4 An Alternative: Spatial Grasp Technology (SGT)
SGT Basic Idea – Function First

- Global function & overall behavior, expressed in a special high-level formalism – primary
- System structure & organization (C2 including) – secondary, as a derivative of the formalized system function

Advantages: High flexibility of runtime system organization, especially in responses to asymmetric events, enhanced possibilities for automated up to fully automatic (unmanned) solutions
Parallel Integral Grasp of Distributed Worlds

- SGT is based on wavelike integral seamless navigation & invasion & coverage & conquest & perception of physical or virtual spaces versus traditional interaction of parts (agents).

- SGT has strong psychological and philosophical background, reflecting how humans mentally plan, comprehend and control operations in complex environments, being put in our case on highly parallel and fully distributed platform.

- SGT pursues gestalt and holistic ideas and principles rather than multi-agent philosophy.
A network of intelligent control modules U, embedded into key system points, collectively interprets integral system scenarios expressed in Distributed Scenario Language (DSL).

The scenarios can start from any node and can cover the system or its parts needed at runtime.

Being understandable for both manned and unmanned units, DSL scenarios are very compact and can be created and modified on the fly.
Injected from the same or different points and at any time, the spatial scenarios can cooperate or compete in a dynamic networked space as overlapping active fields of solutions.
Self-spreading scenarios can dynamically create **knowledge infrastructures** arbitrarily distributed between system components (robots, sensors, humans).

These infrastructures can effectively support **distributed databases, command and control, situation awareness, and autonomous decisions**, as well as any other real or hypothetic computational and (or) control models.
5 Distributed Scenario Language (DSL)

DSL is fundamentally different from traditional programming languages. Rather than describing data processing in a computer memory, it allows us to directly move through, observe, and make any actions and decisions in fully distributed environments (whether physical or virtual).
The Worlds DSL Directly Operates with

- **Virtual World (VW)**, which is finite and discrete, consisting of nodes and semantic links between them.
- **Physical World (PW)**, infinite and continuous, where each point can be identified by physical coordinates (with a certain precision).
- **Virtual-Physical World (VPW)**, being finite and discrete similar to VW, but associating some or all virtual nodes with PW coordinates.
DSL Basic Features

- A DSL scenario develops as parallel transition between sets of progress points (props).
- Starting from a prop, an action may result in one or more props.
- Each prop has a resulting value and a resulting state.
- Different actions may evolve independently or interdependently, and in parallel 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 a position in PW, or both -- when dealing with VPW.
- Any number of props can simultaneously link with the same points of the worlds.
- Staying with the world points (nodes), it is possible to directly access and impact local world parameters, whether virtual or physical, as well as internode links.
Spatial Variables in DSL

- **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 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.
DSL Rules

The rules can represent any action, decision or context, being, for example, 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 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
Main DSL Constructs

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
A spatial grasp $G$ can be of any complexity, from elementary operation in a world point (node) to the ones covering large physical or virtual spaces, with **unified representation** of starting position, overall forward and feedback control, resultant control state, and final results obtained.
Other advancing rules: advance synchronized, repeat, repeat synchronized

Successive operations have the ability of physically moving through the worlds the previous operations navigate and create, providing extended opportunities for massive distributed processing and control in a mobile code (agents) mode, without centralized control.
Other branching rules: if, while, parallel, or, or parallel, and, and parallel, cycle, loop, sling

- Breadth-first operations allow for coordinated automatic code splitting, replication and distribution, simplifying application programs while providing extended sequential and parallel control capabilities.
Combination of DSL depth and breadth mechanisms, taking into account that Gi can itself be any distributed program, allows us to create arbitrary complex spatial scenarios which are more transparent, compact, and integral than any agents-based solutions. These scenarios can however produce agents (mobile inc.) but on automatic implementation level.
Due to the unified recursive language definition, any operations (assignment, arithmetic, logic, string, etc. incl.) can work directly with any operands — from local data values to the results of arbitrary complex scenarios, which can be of any complexity and which can cover any distributed spaces themselves. Direct assignment of such results to multiple and remote variables belongs to these capabilities.
6 DSL Networked Interpreter
The interpreter consists of a number of specialized modules handling & sharing specific data structures.

The whole network of the interpreters can be mobile and open, changing the number of nodes and communication structure at runtime.

The heart of the distributed interpreter is its spatial track system.

Copies of the interpreter can be concealed if to operate in hostile environments.
Forward Track Operations

Evolving grasps & Frontal Variables

Track nodes

Track links

Nodal variables

Terminal track nodes with frontal variables
Echo Operations & Tracks Optimization

- Echoing & merging states
- Collecting remote data
- Tracks optimization

1. Resultant data value
2. Resultant control state
3. N1
4. N2
5. N3
6. 6
7. 8
8. 9
9. F2
10. F1
Forwarding Further Grasps
DSL interpretation Network As a Universal Parallel Spatial Machine

Parallel Distributed DSL Scenario

Operational scenarios & their parts
Infrastructure links
Frontal variables
Control states
Data transfer
Commands
Addresses
7 DSL Programming
Elementary Examples in DSL

Assignment
assign (Result, add (27, 33, 55.6))

Parallel move to physical locations
move (location (x1, y3), location (x5, y8))

Creating virtual node
create (Peter)

Adding virtual link-node pair
advance (hop (Peter), create (+ fatherof, Alex))
Creating & Activating of a
Distributed World

The World To Be Created
Depth-First-Tree-Based Creative Scenario

Creative self-evolving matching formula:
create (# 1; c # 4; e # 5; (b # 2; a ## 1, d ## 4),
  (i # 6; j # 3; f ## 1, g ##4))
Distributed World Creation: Step 1

create ( # 1; c # 4; e # 5; (b # 2; a ## 1, d ## 4),
    (i # 6; j # 3; f ## 1, g ##4))
Distributed World Creation: Step 2

create (c # 4; e # 5; (b # 2; a ## 1, d ## 4),
      (i # 6; j # 3; f ## 1, g ##4))
Distributed World Creation: Step 3

create (e # 5; (b # 2; a ## 1, d ## 4), (i # 6; j # 3; f ## 1, g ## 4))
Distributed World Creation: Step 4

create (b # 2; a ## 1, d ## 4)

create (i # 6; j # 3; f ## 1, g ## 4)
Distributed World Creation: Step 5

create (a # 1)
create (d # 4)
create (j # 3; f # 1, g # 4)

1 -- a -- 2
  |         |
  v         v
   c -- d -- e
   |     |
   i     j

4 -- b -- 5

3 -- 6
Distributed World Creation: Step 6

Graph:
- Nodes: 1, 2, 3, 4, 5, 6
- Edges: a, b, c, d, e, f, g, i, j
- Arrows: create (f##1), create (g##4)
Invading the World with Nameless Active Mobile Objects

sequence ( 

1 create (#1; c#4; e#5; (b#2; a##1, d##4), (i#6; j#3; f##1, g##4)),

2 (hop (1, 4, 5); repeat (sleep (60); hop (random, all links))) )
Allowing Named Mobile Objects See Each Other at Nodes

frontal (ID); nodal (Stay);
sequence (create (#1; c#4; e#5; (b#2; a##1, d##4), (i#6; j#3; f##1, g##4)),
((ID = Peter; hop (1)), (ID = Simon; hop (4)), (ID = John; hop (5));
repeat (if (nonempty (Stay), output (ID, 'sees', Stay)); append (Stay, ID);
sleep (60); remove (Stay, ID); hop (random, all links)))
)
Adding Nodal Activity Informing Neighbors on Objects Seen

fronal (ID); nodal (Stay);
sequence (

1 create (#1; c#4; e#5; (b#2; a##1, d##4), (i#6; j#3; f##1, g##4)),
((ID = Peter; hop (1)), (ID = Simon; hop (4)), (ID = John; hop (5));

2 repeat (if (nonempty (Stay), output (ID, NAME, ‘sees’, Stay)); append (Stay, ID);
sleep (60); remove (Stay, ID); hop (random, all links)))

3 (hop (all nodes); loop (nonempty (Stay); (hop (all links); OUT) = NAME & Stay)).)
Adding Global Regular Inspection of Objects from Any Point

frontal (ID); nodal (Stay);
sequence (

1 create (#1; c#4; e#5; (b#2; a##1, d##4), (i#6; j#3; f##1, g##4));
2 (if (ID = Peter; hop (1)), (ID = Simon; hop (4)), (ID = John; hop (5));
3 repeat (if (nonempty (Stay), output (ID, NAME, 'sees', Stay)); append (Stay, ID);
4 sleep (60); remove (Stay, ID); hop (random, all links))

(hop (all nodes); loop (nonempty (Stay); (hop (all links); OUT) = NAME & Stay))
(hop (4); loop (output ('all:', repeat (free (NAME & Stay), hop first (all links))))))))

Frontal object ID and nodal Stay sequence is defined in the diagram. The process iterates over nodes, checking if the terminal object Stay is nonempty. If it is, it outputs the ID and name of the object associated with the Stay, appends the ID to the Stay, sleeps for 60 units of time, and removes the ID from the Stay before hopping randomly to any link. The process loops over all nodes, checking for nonempty Stay and outputting the all: statement with free Name & Stay, followed by hopping first to all links.
Solving Distributed Network-Defined Problems
Finding Shortest Path in a Fully Distributed and Maximum Parallel Mode

Start

a, b, d, e

frontal (Far, Path); nodal (Distance, Before); hop (a); sequence (Distance = 0;
repeat (hop (all links); Far += LINK; or (Distance == nil, Distance > Far); Distance = Far; Before = BACK))

output (Path = NAME;
repeat (hop (all links); BACK == Before;
Path &= NAME; if (NAME = e, (Path; stop))))

Shortest path tree creation

Shortest path collection
Starting in each node with personal color, marking it

Parallel marking all accessible network with personal color from a randomly chosen neighbor, excluding itself from the process

Checking if the current node solely connects parts of network

Outputting the node’s name
Analyzing Distributed Structures: Finding Cliques

1. **frontal (Clique); hop (all nodes); Clique = NAME**
   
   repeat (hop (all links); not belong (NAME, Clique);
   
   if (and parallel (hop (any links, Clique)),
      if (BACK > NAME, Clique &= NAME, done), fail))

2. **if (length (Clique) >= 3, output (Clique))**

---

1. Starting in each node
2. Growing potential clique in a unique node order until possible
3. Outputting the clique grown, with threshold size given
Finding Arbitrary Structures in Arbitrary Networks

Distributed Networked Space
frontal \((\text{Match})\); hop (all nodes);  
5 \((\text{Match} \&= \text{NAME}; \text{all } \#; \text{not belong } (\text{NAME}, \text{Match}))\);  
if ( and (any \# \text{Match} \[2, 3\]),  
\quad (\text{Match} \&= \text{NAME}; \text{all } \# \text{Match} \[1\]; \text{if (any } \# \text{Match} \[5\], \text{OUT} = \text{Match}))))
Finding Arbitrary Structures: The Result

Applying search template in DSL from any node

X1, X2, X3, X4, X5, X6

8 Collective Robotics

- Installing DSL interpreter into 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, shifting traditional management routines to automatic levels.

- Human-robot interaction and gradual transition to fully unmanned systems is essentially assisted too.
Integration of DSL Interpreter with Traditional Robotic Functionality

- **Other robots or external systems**
  - V – video cameras
  - S – sensors
  - M – motors
  - N – neural networks
  - G – gadgets & manipulators

**Robot 1**
- U
- M
- G
- S
- N

**Robot 2**
- U
- M
- G
- S
- N
Collective Robotics: Task level

By embedding DSL interpreters into robotic vehicles we can task them on a higher, *semantic* level, skipping numerous traditional details of management them as a group -- fully *delegating these to an automatic solution*. 
Example Task

Go to physical locations of the disaster zone with coordinates:

50.433, 30.633
50.417, 30.490
50.467, 30.517

Evaluate damage in each location and return the maximum damage value.

The DSL Program

```
maximum (move ((50.433, 30.633),
              (50.417, 30.490),
              (50.467, 30.517));
evaluate (damage))
```
Initial Scenario Injection

maximum (move ((50.433, 30.633), (50.417, 30.490), (50.467, 30.517)); evaluate (damage))

Mobile robots
Scenario Partitioning & Distribution, Group Infrastructure Creation

maximum

move (50.417, 30.490); evaluate (damage)

Group Infrastructure

move (50.433, 30.633); evaluate (damage)

R1

mov (50.467, 30.517); evaluate (damage)

R2

R3

50.417, 30.490

50.433, 30.633

50.467, 30.517
Simultaneous Robot Movement

maximum

evaluate (damage)

move

Group
Infrastructure

R1

50.417, 30.490

R2

50.467, 30.517

R3

50.433, 30.633

evaluate (damage)
Simultaneous Damage Evaluation

Evaluate

Raw damage data

Group Infrastructure

R1

50.417, 30.490

R2

50.467, 30.517

R3

50.433, 30.633

maximum

evaluate
Finding Maximum Damage & Final Result Return

Maximum damage return

Damage 1

50.417, 30.490

Damage 2

50.467, 30.517

Damage 3

50.433, 30.633

Group Infrastructure

50.417, 30.490
Collective Robotics: Behavior level

By embedding DSL interpreters into robotic vehicles we can easily set up any needed collective behavior of them — from loose swarming to a strictly controlled holistic unit obeying external orders. Any mixture of different behaviors within the same scenario can be provided.
Initial Distribution of Units

DSL Interpreters
**Swarm Movement Scenario**

**swarm_move**

*(Starting from any unit)*

```
hop (all nodes);
frontal (Limits = (dx (0, 8), dy (-2, 5)), Range = 50);
repeat (nodal (Shift) = random (Limits);
        if (empty (hop (all, WHERE + Shift, Range)), shift (Shift)))
```
Swarm Movement Snapshot

Spreading swarm movement scenario

Threshold range to other units

Global direction of randomized movement
Finding Topologically Central Unit and Hopping into It

**find_hop_center**

*(Starting from any unit)*

```plaintext
frontal (Average = average (hop (all nodes); WHERE); hop (min (hop (all nodes); distance (Average, WHERE) & ADDRESS) : 2)
```
Central Unit Found

Central unit finding & hopping scenario

Central unit
Creating Runtime Infrastructure

infra_build

(Starting from the central unit)

stay (frontal (Range) = 100;
repeat (linkup (+ infra, first come, Range)))
 Targets Collection & Distribution & Impact

collect_distribute_impact

(Starting from the central unit)

```plaintext
loop (
  nonempty (
    frontal (Seen) = repeat (free (detect (targets)), hop links (+ infra)));
  repeat (free (select_move_shoot (Seen)), hop links (+ infra)))
```
Hierarchical Targets Collection & Distribution & Impact

Targets collection & distribution & impact scenario

Attacking targets

Collection-distribution
Removing Any Infrastructure

infra_remove

(Starting from the central unit)

stay (hop (all nodes); remove (all links))
Resultant Combined Solution

The obtained resultant scenario, which can start from any unit, combines loose swarm movement in a distributed space with regular updating topologically central unit and runtime hierarchical infrastructure between the units. The latter controls observation of the distributed territory, collects data on targets and distributes them among the units for impact operations.

(Starting from any unit)

```
swarm_move,
repeat ( find_hop_center;
    stay (infra_remove); stay (infra_build);
    orparallel ( loop (collect_distribute_impact),
                sleep (delay) ) )
```
9 Air & Missile Defense under Spatial Grasp Model
Directed Energy Systems
Unified Integration of DEW with Conventional Systems in SGT
In each sensor (Relay Mirror):

- **Frontal** \((\text{Path}, \text{Element}, \text{Range1} = \ldots, \text{Range2} = \ldots)\);
- **Loop** (if \((\text{seen} (\text{target}, \text{Range1}), \) \\
  \((\text{loop} (\text{orient} (\text{target})), \) \\
  \(\text{repeat} (\text{append} (\text{Path}, \text{WHERE}); \text{hop first} (\text{Range2}); \) \\
  \text{if} (\text{NAME} = \text{DE}, \) \\
  \(\text{repeat} (\text{nonempty (Next = Path : increment (Element)}); \) \\
  \text{adjust (WHERE, Next); hop (Next)); \) \\
  \text{repeat (hop (Path : decrement (Element)); \) \\
  \text{activate (DE))})})}))}

Each mirror-sensor observes space, and if sees target, finds a path to the DE source (possibly via other mirrors), adjusts to each other the obtained chain of mirrors and DE, then activates DE via the chain received, while constantly correcting its own angle towards moving target.
Distributed Global Awareness in DSL: Collecting, Distributing, Shooting Multiple Targets

1. Global collection of all possible targets
2. Subsequent distribution to different units, with selection of which targets to shoot individually

```
loop(
    frontal (Seen) = repeat (free (detect (targets)), hop first (infra));
    repeat (free (select_shoot (Seen)), hop first (infra))
)
```
Distributed Radar Network

Limited Visibility Range

Communication Channels

Sensors

neighbor

neighbor
Spatial Object Tracking

Tracking mobile intelligence

Circuits in nodes

Sensors

Moving object
The following program, starting in all sensors, *catches the object* it sees and then *follows* it wherever it goes, if not seen from this point any more (i.e. its visibility becomes lower than a given threshold).

```plaintext
frontal (Object, Threshold = 0.1);
hop (all nodes); Object = search (aerial);
visibility (Object) > Threshold;
repeat (loop (visibility (Object) > Threshold);
max destination (hop (all neighbors);
visibility (Object)) > Threshold))
```
Multiple Objects Tracking & Shooting

Emergency Anti-Swarm Scenarios

Regular Shooters

Moving targets

Shoot link

neighbor

1

2

3
Multiple Objects Tracking & Shooting Details

- Each sensor regularly searching for new targets.
- Each new target is assigned individual tracking intelligence which propagates in distributed virtual space following the target’s movement in physical space.
- If there are available shooters in the vicinity and shooting is allowed and technically feasible, a kill vehicle is launched against the target, reducing the number of them available in the region and in total.
- If target is hit, it is removed from the observation.
- Independently, a global observation & analysis as parallel and fully distributed process is launched.
- This process assesses total number of targets at different locations and over the whole area, total number of shooters available, and by established thresholds classifies the situations occurred, where targets may be organized as swarms.
- If a target swarm is identified, a special anti-swarm scenario is launched using emergency swarms of kill vehicles, with distributed management of swarm-against-swarm tactics.
Multiple Objects Tracking & Shooting in DSL

Nodal (Seen, Thr2 = 3, Thr3 = 0.3, Number, Location, All, Targets, Shooters);
frontal (Object, Threshold = 0.1);

{ hop (all sensors);
whirl (Object = search (aerial, not_belong (Seen)); visibility (Object) > Threshold;
release (repeat (append (Seen, Object);
loop (visibility (Object) > Threshold;
if ((hop (shoot link); CONTENT > 0; allowed (fire, Object);
    shoot (Object); decrement (CONTENT);
    success (shoot, Object)), (withdraw (Object, Seen); done)));
withdraw (Object, Seen);
max destination (hop (neighbor, all); visibility (Object) > Threshold)))
whirl (Targets = sum (hop (all sensors); number (Seen));
Shooters = sum (hop (all shooters); CONTENT);
List = sort (hop (all sensors); nonempty (Seen); number (Seen) & ADDRESS);
loop (nonempty (List); (Number, Location) = peers (withdraw (List, first));
if (and (Number >= Thr2, Number / Targets >= Thr3, Targets > Shooters),
    release (hop (Location); apply (Swarm against Swarm, Seen))))

Global, from any point
Swarm-against-Swarm Scenario

Central target

Peripheral targets

Priority attack

Chasers

Targets

Visual range
Swarm-against-Swarm Scenario Principles

- Initial launch of the swarmed chasers into the targets area.
- Forming targets priority list by their positions in physical space.
- Highest priority is assigned to topologically central targets as potential command and control units.
- Other targets are sorted by their distance from the topological center of the group.
- Peripheral targets are considered more dangerous as having more chances to escape from chasers and cause damage, therefore being of primary priority to be chased and shot.
- Assigning available, free chasers to targets, classifying them as engaged, and subsequently returning back to status free if they are of multiple use (i.e. were not destroyed themselves).
- The vacant chasers are again engaged in the priority targets selection and impact.
- All chaser swarm management is done exclusively within the swarm itself, without external influence.
Swarm against Swarm Scenario in DSL

nodal (Targets, Aver, List, Chaser); frontal (Next);
sequence (  
  Initial launch (chasers, targets (Seen)),
  repeat (    
    hop (random, free chasers);
    Targets = merge (hop (all free chasers); seen (targets, coordinates));
    nonempty (Targets);  Aver = average (Targets);
    List = sort (split (Targets); distance (VALUE, Aver) & VALUE);
    List = append (withdraw (last, List), List)
  )
  loop (    
    nonempty (List); Next = withdraw (first, List) : 2;
    Chaser = min (hop (all free chasers);
      distance (WHERE, Next) & ADDRESS) : 2;
    release (hop (Chaser); STATUS = busy;
      pursue_shoot_verify (Next; STATUS = free) )
  )
)
10 European Missile Defense Scenarios
1: **Infrared satellite system** picks up heat signatures of hostile ballistic missiles launched towards the target
2: Information is transmitted to **ground stations** for processing
3: Processed information sent to **command and control network**
The command and control network relays information to sensor and weapons systems in the region. Once the missiles' engines burn out, the infrared satellite can no longer detect them.
1: Long-range sensors continue tracking the missile to help command system calculate options for destroying them
2: Constantly sharing Information among sensors & weapons systems
Command system has the option of shooting down the hostile missiles while in the upper or lower layers of the atmosphere by upper or lower layer shooters. As tracking continues, greater accuracy is achieved.
Extended European Scenario in DSL

frontal (Target); nodal (New);

hop (infrared_satellite_sensors);

loop ()
    nonempty (New = infrared (new_targets));
    split (New); frontal (Target) = VALUE;
    release ()
        cycle (visible (Target); update (Target);
            hop (DEW); if (shoot_verify (Target), done));
    hop (long_range_sensors);
    cycle (visible (Target); update (Target);
        if (distance (Target) > threshold,
            (hop (upper_layer_shooters);
                if (shoot_verify (Target), done)))));
    hop (short_range_sensors);
    cycle (visible (Target); update (Target);
        hop (lower_layer_shooters);
        if (shoot_verify (Target), done)))

Independent intelligence for each target

DEW Long range Short range

In each satellite infrared sensor
11 Possible Nonlocal Nuclear Scenarios

- **Accidental:** Since countries have "launch on warning" systems that send off rockets before it is confirmed a nuclear attack is underway, any tensions between them can lead to *massive nuclear war within thirty minutes of a warning* -- no matter how false the warning may be.

- **Aggressive:** One or more nations decides to use weapons against nuclear or non-nuclear nations in order to *promote an economic, political or military goal*, as part of an ongoing war or as a first strike nuclear attack. (The state, of course, may claim it is a pre-emptive, retaliatory or even accidental attack.)

- **Pre-emptive:** One or more nations believes (correctly or incorrectly) or claims to believe *that another nuclear nation is about to use nuclear weapons* against its nuclear, military, industrial or civilian targets and pre-emptively attacks that nation. May result from political or military "brinkmanship."

- **Retaliatory:** Use of nuclear weapons *in response to a nuclear attack* - - or even a conventional, chemical or biological attack by a non-nuclear nation.
Dynamics of Possible Nuclear Conflicts, Programmed in DSL

Starting from:

- Mistake
- Retaliation
- Invasion
- Nuclear Exchange
- Threat
- Terrorism
12 Conclusions

We have described a novel ideology and the supporting Spatial Grasp Technology (SGT) for high-level management of distributed dynamic systems that can be useful for advanced air and missile defense. SGT, among others, offers the following possibilities:

- Many targets can be simultaneously captured over the defended area and individually followed & studied by spreading mobile intelligence propagating in networked space (between limited range radars).
- SGT can analyze many moving targets in parallel and cooperatively, discovering whether this is individual or swarm attack and properly orienting the global system response.
- In case of multiple targets and limited physical resources, SGT can globally assess which targets are most important to shoot.
- SGT can operate in both live and simulation modes, with runtime simulation of evolving events serving as a look-ahead facility for live control.
- SGT operates without any central resources, where global mission scenarios, created on the fly, can self-recover and succeed under any indiscriminate damages to computing resources and communication networks.
References


19. “Russia 'To Work with Nato on Missile Defence Shield’”,
   Foundation, Jan. 25, 1990,
   http://www.heritage.org/research/reports/1990/01/brilliant-pebbles-the-
   revolutionary-idea-for-strategic-defense.
22. C. Moore, “Six Escalation Scenarios Spiraling World to Nuclear War”,
   http://www.carolmoore.net/nuclearwar/alternatescenarios.html.
   Support of Crisis Relief Missions”, Journal of Emergency Management, Vol.4, No.4,
   ISSN 1543-5865, July/August 2006, pp. 29-36.
24. P. Sapaty, “Grasping the Whole by Spatial Intelligence: A Higher Level for
25. P. Sapaty, “Intelligent Management of Distributed Sensor Networks”, In: Sensors,
   and Command, Control, Communications, and Intelligence (C3I) Technologies for
   pp. 45-52.
   Europe Security + Defence, bcc Berliner Congress Centre, Berlin Germany, 2009,
   pp. 63-70.


