Global Electronic Dominance with Spatial Grasp

Peter Simon Sapaty
Institute of Mathematical Machines and Systems, National Academy of Sciences,
Kiev, Ukraine
Email: peter.sapaty@gmail.com

Received July 1, 2012; revised September 20, 2012; accepted September 28, 2012

ABSTRACT

A high-level control technology will be revealed that can dynamically establish overwhelming dominance over distributed networked systems with embedded electronic devices and any communications between them. It is based on implanting of universal control modules (that may be concealed) into key system points which collectively interpret complex but compact mission scenarios in a special high-level Distributed Scenario language (DSL). Self-evolving and self-spreading in networks, matching them in a super-virus mode, DSL scenarios can analyze their structures and states and set up any behavior needed, including creation of benign or elimination of unwanted infrastructures. The scalable technology allows us to convert any distributed networked systems into a sort of integral spatial brain capable of analyzing and withstanding unpredictable situations in a variety of important domains.

Keywords: Electronic Dominance; Distributed Dynamic Worlds; Asymmetric Situations and Threats; Spatial Grasp Technology; Distributed Scenario Language; Parallel Networked Interpretation; Global Awareness; Multi-Robot Systems

1. Introduction

In our modern dynamic world we are constantly meeting numerous irregular situations and threats where proper reaction on them could save lives and wealth and protect critical infrastructures. For example, it is no secret that large and powerful traditional world armies, having most sophisticated weapons, are often losing to terrorists, insurgents or piracy with primitive gadgets but very smart and flexible structures making them hard to detect and fight. And delayed reaction on environmental crises like earthquakes, tsunamis or forest fires with their severe consequences can also be the result of inadequacy of existing organizational structures for dealing with emergency situations.

The current paper just deals with these system organizational features where a novel system philosophy and supporting high-level networking technology are described which, using any available electronic communication and data processing means, can quickly react on asymmetric situations, establish dominance over distributed systems, and quickly organize available human and technical resources into operable systems.

The approach, based on a holistic spatial grasp vision resembling in some sense the work of a human brain, 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 shifting traditional organizational routines to automated up to fully automatic implementation. Contrary (and in supplement) to traditional interoperability principles of organization of large distributed systems, in military domain including, the technology offered establishes a higher supervisory, or overoperability [1,2] layer guaranteeing global awareness, pursuit of both local and global goals, and self-recovery from indiscriminate damages.

In this paper we will outline the basics of this Spatial Grasp Technology, details of its implementation in dynamic networked systems, as well as present and explain exemplary scenarios in DSL from different domains where global electronic dominance can be the key to solutions of most urgent and complex tasks.
prehend and control complex operations in distributed environments.

Traditional systems are based on design and creation of their multi-component structures first, with global system function and overall behavior being a result of operation of the predefined structure. Such 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 may result in a considerable loss of system’s integrity and performance.

The presented spatial grasp approach largely starts from the opposite-from global goal and top semantics of the needed overall behavior, expressed in a special DSL formalism, making the system structure and its internal organization runtime dependent on (changing) mission goals and states of the environment in which the mission evolves. This may provide highest possible flexibility of runtime system organization, especially in responses to asymmetric events, offering also enhanced possibilities for automated up to fully automatic (unmanned) solutions.

### 2.2. Distributed Scenario Interpretation

The approach in practice works as follows. A network of universal control modules U, embedded into key system points, collectively interprets mission scenarios expressed in DSL, as shown in Figure 2. The scenarios, based on the spatial grasp idea (capable of representing any parallel and distributed algorithms, spatial cycles and loops including), can start from any node, subsequently covering the whole system or its parts needed at runtime.

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

More details on the SGT, its core language DSL, and its distributed interpreter can be found elsewhere [8-14], with some key features necessary for explanation of the chosen here applications briefed in the following sections.

### 3. Distributed Scenario Language

DSL differs radically from traditional programming languages. It allows us to directly move through, observe, and make any actions and decisions in fully distributed environments.

#### 3.1. The DSL 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 and accessed by physical coordinates with certain precision.
- **Virtual-Physical World** (VPW), finite and discrete, similar to VW, but associating some or all virtual nodes with certain PW coordinates.

#### 3.2. DSL Basic Features

Any sequential or parallel, centralized or distributed, stationary or mobile algorithm operating with information and/or physical matter can be written in DSL at any lev-
els, including the highest semantic ones. Its top level recursive structure is shown in Figure 3.

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 (which may be multiple) or remain in the same one.
- Each prop has a resulting value and resulting state.
- Different actions may evolve independently or interdependently from the same prop.
- Actions may also spatially succeed each other, with new ones applied sequentially or in parallel from all or some props reached by the previous actions.
- Elementary operations may directly use states and 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 be simultaneously linked with the same points of the worlds, sharing local information at them.
- Staying with world points, it is possible to directly access and impact local world parameters, whether virtual or physical.

3.3. DSL Rules

The basic construct, 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 embraced 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.

3.4. 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 activities currently associated with these nodes.

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

Traditional to existing programming languages abbreviations of operations and delimiters can be used too, substituting certain rules as in the examples throughout this text, in order to shorten and simplify DSL programs. The latter, however, are obeying the general syntactic structure shown in Figure 3.

3.5. The Main DSL Constructs

The list of basic DSL constructs is shown in Figure 4, where syntactic categories are shown in italics, vertical bar separates alternatives, the construct in square brackets is optional, and the ones in braces (except boldfaced ones) indicate zero or more repetitions, whereas the remaining symbols and words are the language terminals (including the boldfaced braces).

4. Distributed DSL Interpreter

4.1. DSL Interpreter Organization

The DSL interpreter [2,7-9] with its internal organization

![Figure 3. DSL top level recursive syntax.](image-url)
The “heart and nerve system” of the distributed interpreter is its spatial track system with its parts kept in the Track Forest memory of local interpreters—these being logically interlinked with such parts in other interpreter copies, forming altogether global space control coverage.

This forest-like distributed track structure enables hierarchical command and control as well as remote data and code access, with high integrity of emerging parallel and distributed solutions, without any centralized resources.

The dynamically networked DSL interpreters are effectively forming a sort of a universal and parallel spatial machine ("machine" rather than computer as it operates with physical matter too, and can move partially or as a whole in physical space too) capable of solving any problems in a fully distributed mode, without any special central resources.

5. Creation, Activation, and Management of a Distributed World

We provide here a simple interactive example of a number of DSL and SPT possibilities in dealing with distributed interconnected systems, from their creation and activation to management and supervision.

5.1. Distributed World Creation

An exemplary networked virtual world with named nodes and links may be the one shown in Figure 6.

This (so far) passive world can be created and arbitrarily distributed between computers (from all nodes in the same computer to each node on a separate one) by the following DSL program based on parallel depth-first tree creative template (see also Figure 7), to be initially applied to the empty space:

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

The stages of how this creative template evolves, gradually creating nodes and links connecting them, are shown in Figures 8(a)-(f) (starting in full, then losing worked parts unless becoming empty).
5.2. World’s Invasion with Mobile Objects

Invading the world created with nameless active mobile objects (agents) randomly moving between nodes, as shown in Figure 10, may be done by the following DSL program (starting, say, from nodes 1, 4 and 5 where the agents should start their existence):
hop (1, 4, 5); repeat (sleep (60); hop (random, all links))

Giving personal identity (names) to these mobile objects and allowing them see each other at nodes (by self-registering in shared nodal variables Stay), as in Figure 11, with locally reporting the fact of this seeing, may be accomplished by:
(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))

5.3. Adding Nodal Activity

Adding permanent personal activity to all nodes allowing them, for example, to regularly inform all neighbors on the objects currently staying at them (see Figure 12) as a possible alarm for certain applications, may be done by:
hop (all nodes);
loop (nonempty (Stay); (hop (all links); OUT) = ‘seen at:’ & NAME & Stay; sleep (30))

5.4. Adding Global Supervision and Inspection

Adding regular global inspection with collecting names of all objects currently staying at nodes, in a dynamic breadth-first spanning tree mode starting from a certain point (here node 4), as shown in Figure 13, may be done by the following DSL program:
hop (4); loop (output (repeat (free (NAME & Stay),
hop (first, all links))))
5.5. Runtime Restructuring of the Active Distributed World

And any restructuring of this active distributed system can be done at runtime too. For example, removing node 1 with all adjacent links and, in parallel, adding link w
between nodes 2 and 3 (as in Figure 14) can be done by the following program:
remove (hop(1); all links #), (hop (3); linkup (w, 2))
The modified distributed system will remain active and operational under the changed configuration, same as before.

6. Analyzing and Impacting Network Structures in Distributed Systems

Of considerable importance in dealing with distributed systems may be finding weak (or weakest) and strong (or strongest) parts in them, whether these systems being civil or military (say, battlefields in the latter case), and friendly or hostile. Solution of other problems may relate to finding certain substructures in distributed organizations by their proper descriptions, or patterns. In the examples below we formulate and solve some of these tasks on colored graphs where each graph node may be located in a separate computer and links can connect nodes in the same or in different computers.

6.1. Finding Weakest Points

To find the weakest nodes in a graph like articulation points (see Figure 15), which when removed split it into disjoint parts, the following program suffices (resulting in node d which is chosen to be physically removed for a certain application).

```
nodal (Mark);
hop (all nodes); COLOR = NAME; Mark = 1;
and ( (hop (random, all links);
repeat (grasp (Mark == nil; Mark = 1);
hop (all links)),
(hop (all links); Mark == nil),
remove (CONTENT) )
```

This program works in the following steps.
- Starting in each node with personal color, marking it.
- Parallel marking all accessible subnetwork with personal color from a randomly chosen neighbor, excluding itself from the marking process.
- Checking if the current node solely connects parts of network.

6.2. Finding Strongest Parts

Cliques (or maximum fully connected sub-graphs of a graph, as in Figure 16), on the contrary, may be considered as strongest parts of a system. They all can be found in parallel by the following simple program resulting for Figure 15 in cliques: (a, b, c, d), (c, d, e), and (d, e, f). These cliques are then chosen to be printed locally rather than removed, as in the previous case.

```
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));
if (length (Clique) >= 3, output (Clique))
```

The program operates in the following steps:
- Starting in each node.
- Growing potential clique in a unique node order until possible.
- Printing the clique grown, with threshold size given.

6.3. Finding Arbitrary Structures by Parallel Pattern Matching

Any structures in distributed networked systems can be found by describing them in DSL, like the one in Figure 17, which can be applied from any network node, evolving subsequently in a parallel replication and pattern-matching mode. The following DSL program, reflecting the search pattern (template) of Figure 16 (with variable

Copyright © 2012 SciRes.
More on parallel and distributed operations on general graph may be found in [12,13], where the DSL’s predecessor called WAVE was used [13].

7. Providing Global Awareness & Targeting

Establishing global electronic supervision over any distributed systems, SGT effectively provides global awareness of complex situations in them, for example, for discovering, collecting and distributing hostile targets seeing locally from their different points, as shown in Figure 18, and by the following DSL program.

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

This constantly looping, self-evolving and self-spreading distributed program, providing global collection of possible targets throughout the region of concern and their subsequent distribution back to local units (the latter selecting which targets to shoot individually), can start from any component of the system having DSL interpreter installed (communication links between the interpreters, which can be dynamic and casual, are called infra in Figure 18).

8. Fighting Viruses in Distributed Networks

SGT can also allow us to find independently and in parallel potential nodes from which viruses flooding a network might have originated, by backward tracing their spread until their footprints remain in the nodes passed, as shown in Figure 19.

All virus sources finding program in DSL may be as
Figure 19. Finding probable virus sources in parallel.

follows, where the probable virus sources are lifted, sorted, and output in the order of their likelihood.

```plaintext
nodal (Trace);
sequence (hop (all nodes); nonempty (check (viruses));
repeat (increment (Trace); nonempty (Predecessor =
where from (viruses)); hop (Predecessor))),
output (sort (hop (all nodes); empty (Predecessor);
nonempty (Trace); Trace & ADDRESS)))
```

This DSL program, spreading itself as a computer virus too, may be symbolically considered as a sort of “biological” weapon for fighting another, malicious viruses in a computer network. Having, however, an advantage over usual viruses as reflecting a powerful globally controlled spatial algorithm, dynamically interlinked in space as an integral global oriented unit.

9. Coastal Waters Unmanned Patrol Example

This example relates to a physical world, with physical movement of equipment with orientation on mobile robotics (like unmanned underwater vehicles, or UUV), as shown in Figure 20 (for simplicity, we consider here only a two-dimensional example).

At the beginning we should create a coastal waypoint map in the form of a semantic network, as follows.

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

The two-vehicle parallel solution may be achieved by the following program searching the water space for alien objects to the depth available by vehicle’s sensors, with vehicles moving forward and backward independently, according to the coastal map, assuming each capable of avoiding collisions when on opposite courses:

```plaintext
(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 may be when each vehicle turns back if discovers another patrol vehicle on its way (as a confirmation that the way ahead has already been serached), checking for this its vicinity by depth2, as follows.

```plaintext
(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 cases, the whole coastline will always be searched in full if at least one vehicle remains operational. The case can be easily extended to any number of patrol vehicles searching the same coastline simultaneously.

10. Expressing Battlefield Scenarios

Formalization of Command Intent (CI) and Command and Control (C2) in general, are among the most urgent and challenging problems on the way to creation of effective multinational forces, integration of simulations with live control, and natural transition to robotized armies. Specialized languages for unambiguous expression of CI and C2 (like BML and its derivatives C-BML, JBML, geoBML, etc.) [15,16] are not programming languages themselves, needing therefore integration with other linguistic facilities and organizational levels to provide required system parameters.

On the contrary, working directly with both physical and virtual worlds, DSL allows for effective and universal expression of any battlefield scenarios and orders in parallel and fully distributed manner, also allowing for straightforward implementation in robotized up to fully robotic systems. DSL scenarios are also much shorter and simpler, as in the following example taken from [16], both (simplified) Figure 21 and the BML code.

The task is to be performed by two armoured squadrons BN-661 Coy1, and BN-661 Coy3, which are ordered to cooperate in coordination. The operation is divided into four time phases: from TP0 to TP1, from TP1 to TP2, from TP2 to TP3, and from TP3 to TP4, to finally secure objective Lion, and on the way to it, objective Dog. Their coordinated advancement should be achieved by passing Denver, Boston, Austin, Atlanta, and Ruby lines, while fixing and destroying enemy units Red-1-182, Red-2-194, Red-2-196, and Red-2-191.

Tasks assigned to Coy1 are written in BML as follows:
deploy BN-661 Coy1 at Denver end before TP0
in-order-to enable label-o11 label-o10;
advance BN-661 Coy1 from Denver to Boston start
at TP0
in-order-to enable label-o12 label-o11;
fix BN-661 Coy1 Red-1-182 at Boston end nlt TP1
in-order-to enable label-o35 label-o12;
advance BN-661 Coy1 to Austin start at TP1
in-order-to enable label-o14 label-o13;
fix BN-661 Coy1 Red-2-194 at Dog end nlt TP2
in-order-to enable label-o35 label-o14;
advance BN-661 Coy1 to Atlanta start at TP2
in-order-to enable label-o16 label-o15;
fix BN-661 Coy1 Red-2-196 at Atlanta end nlt TP3
in-order-to enable label-o37 label-o16;
advance BN-661 Coy1 to Ruby start at TP3
in-order-to enable label-o17 label-o18;
fix BN-661 Coy1 Red-2-191 at Lion end nlt TP4
in-order-to enable label-o39 label-o18;
seize BN-661 Coy1 Lion at Lion end nlt TP4
in-order-to cause label-ci1 label-o19;
Tasks assigned to Coy3 in BML:
deploy BN-661 Coy3 at Denver end before TP0
in-order-to enable label-o32 label-o30;
support BN-661 Coy3 Coy1 at Troy start at TP0 end
at TP4 label-031;
attspt BN-661 Coy3 Red-1-182 from Denver to
Boston start at TP0 end nlt TP1
in-order-to enable label-o12 label-o32;
destroy BN-661 Coy3 Red-1-182 at Boston end nlt
TP1
in-order-to enable label-o13 label-o33;
attspt BN-661 Coy3 Red-2-194 from Boston to Dog
start at TP1 end nlt TP2
in-order-to enable label-o14 label-o34;
destroy BN-661 Coy3 Red-2-194 at Dog end nlt TP2
in-order-to enable label-o15 label-o35;
attspt BN-661 Coy3 Red-2-196 from Dog to Atlanta
start at TP2 end nlt TP3
in-order-to enable label-o16 label-o36;
destroy BN-661 Coy3 Red-2-196 at Atlanta end nlt
TP3
in-order-to enable label-o17 label-o37;
attspt BN-661 Coy3 Red-2-191 from Atlanta to Lion
start at TP3 end nlt TP4
in-order-to enable label-o18 label-o38;
destroy BN-661 Coy3 Red-2-191 at Lion end nlt TP3
in-order-to enable label-o19 label-o39;
The following same mission description, but now in
DSL, is much shorter; it can be created and modified on
the fly and executed by manned, mixed, or fully robotic
forces (with most of command and control hidden and
shifted to automatic internal DSL interpretation level).

This can effectively relieve human commanders from a
multitude of traditional explicit C2 routines, allowing
them concentrate on global mission objectives and effi-
ciency instead.

FIXER = BN-661 Coy1;
SUPPORTER_DESTROYER = BN-661 Coy3;
advance_synchronize (deploy (Denver, TFIN = TP0),
move_destroy (pl: Boston, target: Red-1-182, TFIN = TP1),
move_destroy (pl: Austin, target: Red-2-194, TFIN
= TP2),
move_destroy (pl: Atlanta, target: Red-2-196, TFIN = TP3),
move_destroy (pl: Ruby, target: Red-2-191, TFIN
= TP4));
seize (LION, TFIN = TP4)

Many other applications of the spatial grasp paradigm
can be found in [17-25].

11. Conclusions

We have briefed a new type of ideology and resulting
networking technology aimed at establishing global con-
tral and supervision of distributed systems with any elec-
tronic means of communication and data processing em-
bded.

The paradigm, called overoperability or spatial grasp,
believably resembles of how human brain comprehends
and manages active distributed worlds with integral ge-
stalt-like, non-atomistic world vision. But unlike the
brain operation, this approach has been put on a highly
parallel and fully distributed technological scalable plat-
form giving it advantages in solving problems in very
large and complexly interconnected domains where bio-
logical knowledge processing and intuition may fail.

Within the technology developed, it is possible to de-
scribe in a special high-level spatial language any local and global operations and control in both physical and virtual worlds and set up and supervise their behavior needed, including world’s modifications and initial creation. The approach also allows us penetrate into other systems and their organizations, both friendly and hostile, analyze their internal structures and behavior and change them in the way required, as well as integrate with other local and global electronic means, establishing overoperability layer on top of them.

On the implementation layer, SGT extensively employs replication and mobile code capability, allowing mission scenarios spread instructions, data and control in distributed worlds, spatially linking them with each other in a super-virus pattern matching mode, effectively confronting other networking technologies, computer viruses including. And electronic communications between system components may be local, limited, unsafe, and changing at run time, but the self-spreading interpreted spatial scenarios may always survive and fulfill objectives.

Applications of the technology offered may be numerous and in most diverse fields—from network management to networked battlefields and future robotized combat systems. Also, taking into account the overwhelming world computerization, use of internet, 3 billion mobile phone users, and its scalability and viruslike nature, it can help launch and supervise global world missions in a great variety of areas including environmental protection, education, economy, space research, security, and defense.

REFERENCES


Copyright © 2012 SciRes. IJCNS
