19th International
Command and Control
Research and Technology Symposium
June 16-19 - Alexandria, Virginia, USA 2014

Automated System Organizations Under Spatial Grasp Technology

Peter Simon Sapaty

Institute of Mathematical Machines and Systems, National Academy of Sciences
Glushkova Ave 42, 03187 Kiev Ukraine, peter.sapaty@gmail.com

Abstract: A high-level organizational approach is offered based on grasping top mission semantics and overall system functionality in a special high-level language, which allows most of traditional management routines, command and control including, to be effectively shifted to automated up to fully automatic levels. This “spatial grasp” language is collectively interpreted by a dynamic network of communicating interpreters embedded into key system points, in open or stealth mode. This allows us to concentrate on global mission goals and overall system efficiency and timely react on unpredictable and asymmetric situations and events. The technology offered having a ubiquitous super-virus feature with extended capability of self-recovery can create runtime intelligent spatial infrastructures governing distributed systems or penetrate into existing infrastructures investigating their origin, weak and strong points, and providing the impact needed. The approach also allows us to naturally engage robotic components in the force mix under automatic command and control resulting from parallel and distributed language interpretation, with unified transition to fully unmanned systems.

Keywords: Spatial Grasp Language, over-operability, system integrity, system dynamics, battle management, automated command and control, infrastructure protection, unmanned systems, asymmetric response.

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 organizations 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 traditional approach to system design, development and management, command and control including, supposes the system structure and system organization to be predominantly primary, created in advance, whereas global function and overall system behavior appearing as secondary, like shown in Fig. 1.

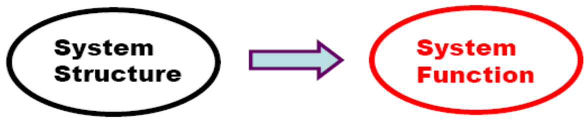


Fig. 1. Traditional approach to system design

The systems based on this vision, ideologically relevant to multi-agent organizations [1] still prevailing almost everywhere, are often clumsy and static, they may fail to quickly adapt to dynamic and asymmetric situations.

With global goals changing, the whole projects based on creating structures and overall system organizations first may quickly become outdated or not needed at all despite huge investments made into them, like, for example, the robotized Future Combat Systems project, or FCS [2], see Fig. 2. The latter, designed mainly for classical battlefields, became obsolete even in its infancy after the main world operations changed towards terrorism fight, for which quite different system ideology and organization appeared to be needed.

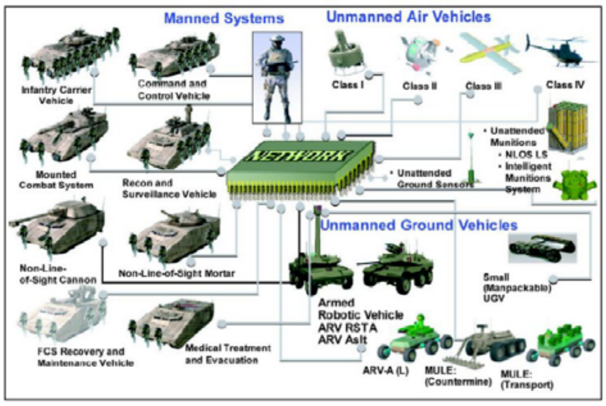


Fig. 2. The canceled FCS project

FCS was also based on a rigid 4D-RCS architectural model inheriting classical hierarchical military organizations, even for AI purposes (Fig. 3), which became inadequate for asymmetric guerrilla-style warfare emerging and dominating in the 21st century.

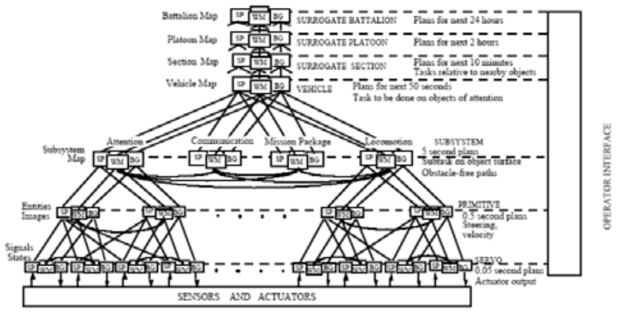


Fig. 3. FCS Basics: 4D-RCS model mrchitecture

Taking this into account, we are pursuing an alternative system approach, known as spatial grasp [3] and over-operability [4,5] in contrast to traditional interoperability, where the global function and overall behavior should be considered, as much as possible, primary, and the system structure and organization as secondary, the latter as a dynamic derivative of the former (Fig. 4).

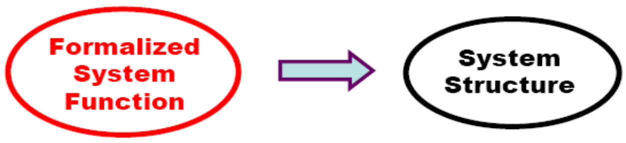


Fig. 4. The alternative system organization considered

The related Spatial Grasp Technology (SGT) starts from global goal and top semantics of the needed overall behavior which are expressed in a special Spatial Grasp Language (SGL), making the system structure and its internal organization a runtime derivative from changing mission goals and states of the environment. This may provide high flexibility of runtime system organization, especially in responses to asymmetric events, offering also enhanced possibilities for automated up to fully automatic (unmanned) solutions.

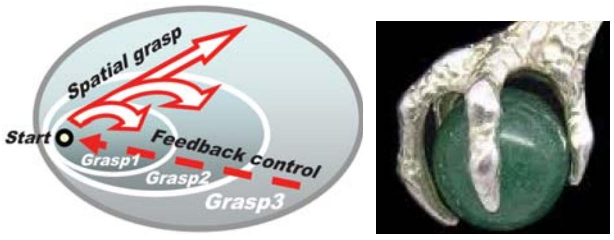
The approach offered has been tested on numerous applications and in different countries, with some history development stages depicted in Fig. 5.

- **Simulating power networks: 66-71**
- **Creating first distributed computer networks: 69-79**
- **Parallel computations, mobile agents: 71-80**
- **Nationwide global networked management system: 75-80**
- **Intelligent hardwired computers, serially produced: 70-80**
- **Distributed macro-pipeline supercomputer: 78-85**
- **Wave model of distributed parallel computations: 74-90**
- **WAVE system at Basic AI Lab in Czechoslovakia: 84-87**
- **Alexander von Humboldt WAVE project, Germany: 88-90**
- **Intelligent network management project for Siemens: 90-93**
- **WAVE system in Germany, UK, US, Canada: 90-00**
- **Distributed simulation of battlefields: UK, DIS project US: 93-98**
- **Cooperative robotics, Japan: 01-05**
- **European patent: 90-93**
- **John Wiley books: 99, 05, new one in progress**

Fig. 5. SGT development history

2 Spatial Grasp Model

The spatial grasp model underlying the technology offered is based on formalized stepwise, wavelike, seamless navigation, coverage, or grasping of distributed physical and virtual spaces, as shown in Fig. 6.



a) b)

Fig. 6. Incremental integral grasp of distributed worlds: a) virtual interpretation, b) symbolic physical analogy

This mode of high-level system vision based on holistic and gestalt principles [6-8] rather than cooperating parts or agents [1] has psychological and philosophical background [9] reflecting, for example, how humans (especially top commanders) mentally plan, comprehend, and control complex operations in distributed environments.

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 SGL, as shown in Fig. 7. The scenarios based on the spatial grasp model (capable of representing any parallel and distributed algorithms, spatial branching, cycles and loops including) can start from any node, subsequently covering the whole system or its parts needed at runtime.

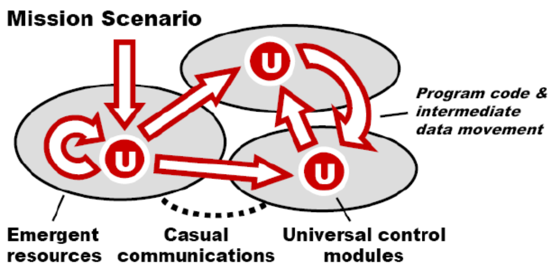

Mission Scenario
Program code & intermediate data movement
Emergent resources
Casual communications
Universal control modules

Fig. 7. Collective scenario execution in dynamic environments

SGL 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, as shown in Fig. 8.

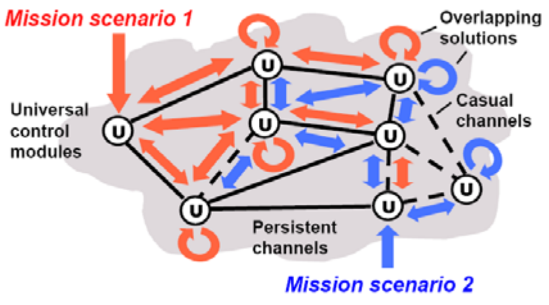


Fig. 8. Distributed scenario spatial interactions

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.

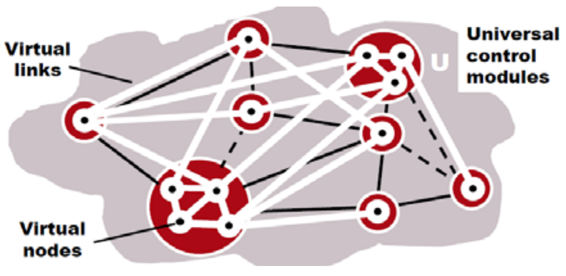


Fig. 9. Creating Spatial Infrastructures by SGT

3 Spatial Grasp Language

3.1 SGL orientation and peculiarities

SGL differs fundamentally from traditional programming languages. Rather than working with information in a computer memory, as usual, it allows us to directly move through, observe, and make any actions and decisions in fully distributed environments, whether physical or virtual. In general, the whole distributed world, which may be dynamic and active, is considered in SGL as a substitute to traditional computer memory. SGL directly operates with:

- *Virtual World* (VW), which is finite and discrete, consisting of nodes and semantic links between them, both nodes and links capable of containing any information, of any nature and volume.

- *Physical World* (PW), infinite and continuous, where each point can be identified and accessed by physical coordinates expressed in a proper coordinate system, and with the precision given.

- *Execution world* (EW), consisting of active doers with communication channels between them, where doers may represent humans, robots, laptops, smartphones, any other devices or machinery capable of operating on the previous three worlds.

Different combinations of these worlds can also be possible, for example, *Virtual-Physical World* (VPW) allowing not only for a mixture of the both worlds but also their deeper integration where VW nodes can be associated with certain PW coordinates, thus making their presence in physical reality too. Another possibility is *Virtual-Execution World* (VEW) where doer nodes may be associated with virtual nodes like having special names (or nicknames) assigned to them, having now semantic relations between them too, like between pure VW nodes. *Execution-Physical World* (EPW) can pin some or all doer nodes permanently to certain PW coordinates, and *Virtual-Execution-Physical World* (VEPW) can combine features of the previous two variants.

3.2 Top SGL syntax

SGL has recursive structure top level of which is shown in Fig. 10. Such organization allows it to express any spatial algorithm, create and manage any distributed structures with any topologies, static as well as dynamic, also solve any problem in, on, and over them—and all this can be expressed in a very compact, transparent, and unified way.

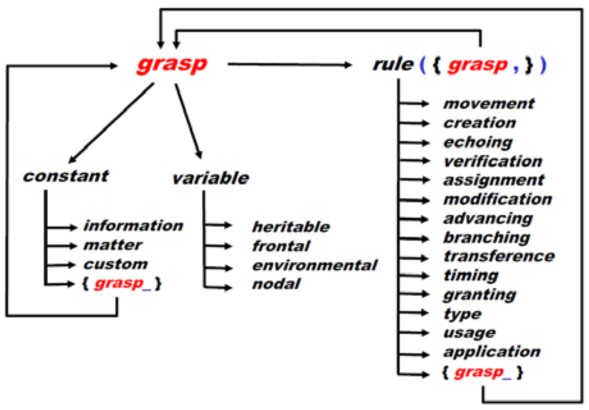


Fig. 10. SGL recursive syntax

Let us explain the language basics in a stepwise top-down manner.

The SGL topmost definition, where scenario in it is named *grasp* (reflecting the spatial navigation-grasp-conquest model explained above) can be expressed as follows:

grasp → *constant* | *variable* | *rule* ({ *grasp*, })

with syntactic categories shown in italics, vertical bar separating alternatives, braces to indicate repetitive parts with the delimiter (here comma) between them shown at the right, whereas parentheses and commas being the language symbols.

As follows from this notation, an SGL scenario, applied in a certain world point (i.e. of PW, VW, EW or their combinations), in the simplest form can just be a constant defining the result explicitly. It can also be a variable containing certain data, say, assigned to it previously by some or other SGL scenarios, which may have happened to visit this point of the world before.

The next option may be one or more constants, variables, or recursively grasps again (treated as operands and separated by a comma if more than one) which are embraced by a certain operation, or *rule*, with the use of parentheses. The rules, starting in the current world position, can be of most diverse natures -- from local matter or information processing to global management and control. The rules can produce results (which may be single or multiple) in the *same or other* world locations.

Due to recursion in the language definition, the results obtained and world locations reached by rules may, in their turn, become, respectively, operands and/or starting places for other rules, with new results and new locations (single or multiple too) obtained after their completion, and so on.

The scenario can thus dynamically spread & process & match the world or its parts needed, with the scenario code capable of virtually or physically moving (the local data too, as will be clear afterwards) in the distributed space, matching the latter and possibly losing utilized parts if not needed any more. This movement can take place in single or multiple, parallel, same or different parts/copies, dynamically linking with each other within automatic spatial control spreading and covering the navigated world too.

SGL constants can represent *information*, physical *matter* (or physical objects), or *custom* defined data items extending the language for specific applications, as follows:

constant → *information* | *matter* | *custom* | { *grasp_* }

The word "constant" is used rather symbolically in the SGL definition, as the last option shown above is recursively defined as *grasp* again (possibly, even an aggregate of grasps separated by underscore), thus capable or representing any complex objects, passive or with embedded activities, for their partial or complete processing.

SGL variables called "spatial" which may be stationary or mobile and contain information or matter, are serving different features of distributed scenarios. As follows, they may be of the four types: *heritable* (stationary), *frontal* (mobile), *environmental* (stationary or mobile), and *nodal* (stationary), with their semantics and usage explained later.

variable → *heritable* | *frontal* | *environmental* | *nodal*

And rules can belong to the following main classes:

rule → *movement* | *creation* | *echoing* | *verification* |
assignment | *modification* | *advancing* |
branching | *transference* | *timing* | *granting* |
type | *usage* | *application* | { *grasp_* }

The final rule's option, *grasp* again, brings another level of recursion into SGL where operations may not only be named explicitly but can also represent results of spatial development of corresponding SGL scenarios of any world coverage and complexity. This option also offers composition or aggregates (separated by underscore) of different operations dedicated to work jointly on operands they commonly embrace.

3.3 SGL main features

Here are some general aspects of SGL scenarios explaining their evolution in distributed worlds.

The basic concept is *progress point*, or *prop* (the latter word is used here as an abbreviation and differs from traditional meaning of the word "prop"). The prop identifies a *combined scenario development and control step* in the united space & time continuum. By using the notion of props we can clearly explain how SGL scenario operates and evolves, with key points being as follows.

- Applied to some point of the world (which may be of different natures as explained before), an SGL scenario is considered to be in the *starting prop* associated with this entry point.
- When activated, the scenario develops in a stepwise manner, generally as a parallel transition between consecutive sets of props (the initial set containing the starting prop only). Each new set (or sets, as scenario may branch) identifies the final result of the current step of scenario evolution having started from the previous set (or its subsets).
- Starting from a prop, a scenario action may result in new props (which may be multiple, as a set) or remain in the same prop. In the latter case, this prop may again be added to the resulting set of props obtained from other starting props, for further common activities from all these, if multiple space & time propagations occur.
- Each prop has a resulting *value*, which may be single one representing information or matter or a list of values (potentially: nested), and a resulting control *state* (one of `thru`, `done`, `fail`, or `fatal`, with their meanings explained later).
- Different operations (represented by arbitrary scenarios) may evolve *independently* or *interdependently* in space and time from the same prop, and in *ordered*, *unordered*, or *parallel* manner.
- Operations may also *spatially succeed* each other, with new ones applied from the props reached by the previous actions. This potentially parallel wavelike evolution in space-time continuum may take place in *synchronous* or *asynchronous* mode.
- Operations and decisions represented by rules can use states and values associated with props reached by other operations, *whatever complex and remote* the latter might be.
- Any prop is always *associated* with a *point of the world* (i.e. physical, virtual, execution or combined node) the related scenario branch is currently developing in.
- *Any number* of props can be simultaneously associated with the *same* world points, sharing local information at them, if needed.
- Staying with world points, it is possible to *directly* access and change local parameters in them, whether physical or virtual, thus impacting the worlds (or trying to do so) via these points.
- Overall organization of the breadth and depth world navigation and coverage is provided by a variety of powerful SGL rules, which may be arbitrarily *nested* within complex processing, evolution, control, management, and supervision structures.

As was shown in previous publications, any sequential or parallel, centralized or distributed, stationary or mobile algorithm operating with information and/or physical matter can be written in SGL on any levels. The latter ranging from top semantic (also close to what is called "command intent") to those detailing system partitioning, composition, infrastructures, subordination between active components, and overall management and command and control.

3.4 The Sense and Nature of SGL Rules

Explaining the language basics further, let us shed some light on the sense and nature of rules, to be explained later in detail. A rule, representing in SGL any action or decision, may, for example, be as follows:

- Elementary arithmetic, string, or logic operation.
- Move or hop in a physical, virtual, execution, or combined space.
- Hierarchical fusion and return of (potentially remote) data.
- Distributed control, both sequential and parallel, and in breadth or depth.
- A variety of special contexts detailing navigation in space while influencing embraced operations and decisions.

- Type or sense of a value, or its chosen usage, guiding and simplifying automatic language interpretation.
- Creation or removal of nodes and/or links in distributed knowledge networks.
- Result of local or global operations of arbitrary complexity and space coverage, which can find, select, or produce the rule needed.
- As already mentioned, a rule can also be a compound one integrating a number of other rules.

All rules, regardless of their nature, sense, or complexity, are obeying the same ideology and organization, as follows:

- Starting from a certain space location, initially linked to it.
- Performing certain operations in a distributed space.
- Producing final results in the resultant set of props with their states and values.
- Linking to same or new world positions reached by the rule's activity.

This uniformity allows us to effectively compose highly integral and transparent spatial algorithms of any complexity and any world coverage, which can operate altogether under fully automatic, parallel and distributed control.

3.5 SGL Spatial Variables

Let us consider some details on the nature and sense of spatial variables, stationary or mobile, which can be used in fully distributed physical, virtual, or executive environments, effectively serving multiple cooperative and integral processes:

- *Heritable variables* – stationary, starting in a prop and staying with this prop permanently (even though the prop has become a past history only, with active processes already gone with other props) and serving all subsequent props, which can share them in read & write operations.
- *Frontal variables* – mobile, temporarily associated with currently active props (not being shared with other props), and then moving with the scenario evolution to subsequent props, accompanying scenario activity. These variables replicate if from a single prop a number of other props emerge.
- *Nodal variables* – stationary, being a private, direct property of the world locations/nodes reached by the scenarios. Staying at world nodes, they can be accessed and shared by all activities having reached these nodes under same scenario identity and at any time (their life span will be explained later).
- *Environmental variables* – these allow us to access different features of physical and virtual words during their navigation, also internal parameters of the distributed SGL interpretation system, to assess, guide, and optimize scenario execution. Most of them are stationary, associated with stationary world positions, but some, related to the execution system itself, can be mobile.

These types of variables, especially when used together, allow us to create advanced spatial algorithms working *in between* components of distributed systems rather than *in* them, providing flexible, robust, and self-recovering solutions. Such algorithms can freely replicate, partition, spread and *migrate* in distributed environments (partially or as an *organized whole*), preserving global integrity and overall control.

3.6 Control States and Their Hierarchical Merge

The following control states appear in props during scenario evolution in distributed space-time continuum. They are used for distributed control of multiple sequential and parallel processes, with making intelligent decisions at different levels.

- `thru` – indicates full success of the current branch of the scenario with capability of further development (i.e. indicating successful operation not only in but also *through* this stage of control). Next scenario stages, if any, will be allowed to proceed from the current prop.
- `done` – indicates success of the current stage with its planned termination after which no further development of this particular branch from the current prop will be possible (unless this status is subsequently changed by a special higher-level rule).
- `fail` – indicates non-revocable failure of the current branch, with no possibility of further development. This state relates to the current branch/prop only, not influencing directly the development of other branches of the scenario. It, however, same as the previous states, can influence decisions on higher levels by control rules which can allow or block development of other branches.

- `fatal` – reports fatal, terminal failure with nonlocal effect, triggering abortion of all evolving processes and associated temporary data, which may be parallel and distributed, also active, regardless of their current world locations and their success or failure. The scope of this global cancellation process may be the whole scenario or only its part embraced by a special rule (explained later) supervising the area in which this state may happen to occur.

These control states appearing in different branches of a parallel and distributed scenario at bottom levels can be used to obtain generalized control states for higher scenario levels, up to the whole scenario, for making proper decisions. The hierarchical bottom-up merge & generalization of states is based on their comparative importance, where the stronger state will always dominate when ascending towards the root.

For example, the merge of states `thru` and `done` will result in `thru`, thus generally classifying successful development at a higher scenario level with possibility of further expansion from all or at least some of its branches. Merging `thru` and `fail` will result in `thru` too, indicating general success with possibility of further development despite some branch (or branches) terminated with failure, but others remained open to further evolution. Merging `done` and `fail` will result in `done` indicating successful termination in general while ignoring local failures, without possibility of further development in this direction.

And `fatal` will always dominate when merging with any other states unless its influence is restricted at top by a special rule which, in case of discovering state `fatal` under its supervision, will itself result with `fail` for higher assessment and control. So ordering these states by their powers from maximum to minimum will be as follows: `fatal`, `thru`, `done`, `fail`.

3.7 The Use of Conventional Notations

To simplify SGL programs, traditional to existing programming languages abbreviations of operations, also conventional delimiters can be used too, substituting certain rules as in numerous examples throughout this book, always remaining, however, within the general syntactic structure shown in Fig. 10. A number of such code simplifications will be used in the subsequent sections when describing different scenarios in SGL for solving concrete problems.

3.8 Some elementary examples in SGL

- Just representing result directly, as a numerical, string, or custom constant:
77, ‘Peter’, Peter

- Multiplication of two constants with the result as an open value :
multiply (34, 5.5) or 34 * 5.5

- Assigning a sum of values to variable Result:
Assign (Result, add (27, 33, 55.6)) or
Result = 27 + 33 + 55.6

- Moving to two physical locations (x1, y3) and (x5, y8) in parallel:
Move (location (x1, y3), location (x5, y8))
or in a shortened way:
move (x1_y3, x5_y8).

- Creating isolated virtual node Peter:
create (‘Peter’) or create (Peter) if Peter is a custom name.

- Extending node Peter as father of Alex, the latter to be a new node:
advance (hop (‘Peter’), create (+‘fatherof’, ‘Alex’)) or
hop (Peter); create (+fatherof, Alex) – shortened, and for custom names.

- Tasking doer D1 to shift in physical space on coordinate deviation (dx, dy):
advance (hop (D1), increment (WHERE, (dx,dy))) or
hop(D1); WHERE += dx_dy
(`WHERE` is a special environmental variable keeping physical coordinates of the node, here `D1`, in which scenario control is currently staying.)

- Tasking D1 to move directly to new physical coordinates (x, y) will be as follows:
advance (hop (D1), assign (WHERE, (x, y))) or
hop (D1); WHERE = x_y.

3.9 Full SGL Summary

SGL full description summarizing the listed above language constructs is as follows, where, as already mentioned, syntactic categories are shown in italics, vertical bar separates alternatives, and parts in braces indicate zero or more repetitions with a delimiter at the right. The remaining characters and words are the language symbols (including braces shown in bold).

grasp → *constant* | *variable* | *rule* ({ *grasp*, })
constant → *information* | *matter* | *custom* |{ *grasp_* }
variable → *heritable* | *frontal* | *nodal* | *environmental*
rule → *movement* | *creation* | *echoing* | *verification* | *assignment* | *modification*| *advancing* |*branching* | *transference* | *timing* | *granting* | *type* | *usage* | *application* | { *grasp_*}
information→ *string* | *number* | *special*
string → '{*character*}' | **{**{*character*}**}**
number → *standard* | zero | one | two | three | four | five | six | seven | eight | nine | plus | minus | dot
matter → "{*character*}"
movement → hop | move | shift
creation → create | linkup | delete | unlink
echoing → state | order | rake | element | content | index | count | sum | first | last | min | max | random | average | access | sortup | sortdown | reverse | add | subtract | multiply | divide | degree | separate | unite | attach | append | common
verification → equal | notequal |less | lessorequal | more | moreorequal | none | empty | nonempty | belongs | notbelongs | intersects | notintersects
assignment → assign | remove | withdraw | assignpeers
modification → inject | replicate | split
advancement → advance | slide | repeat | fringe
branching → branch | sequence | parallel | if | or | and | choose | firstrespond | cycle | loop | sling | whirl| *empty*
transferen → run | call | input | output |transmit | send | receive
timing → sleep | remain
granting → supervise | release | free | blind | lift | none | stay | seize
type → heritable | frontal | nodal | environmental | matter | number | string
usage → address | coordinate | content | index | time | speed | name | place | center | range | doer | node | link | unit | scenario | world | *empty*
heritable → H{*alphameric*}
frontal → F{*alphameric*}
nodal → N{*alphameric*
environmental→ TYPE | NAME | ADDRESS | QUALITIES | WHERE | BACK | PREVIOUS | PREDECESSOR | DOER | RESOURCES | LINK | DIRECTION | WHEN | TIME | SPEED | STATE | VALUE | COLOR | IN | OUT | STATUS | *specific*
special → thru | done | fail | fatal | infinite | nil | nodes | links | any | all | allother | passed | existing | ` neighbors | direct | noback | # firstcome | forward | backward | global | local | synchronous | asynchronous | virtual | physical | executive | engaged | vacant

4 Distributed SGL Interpreter

4.1 The Interpreter General Organization

The SGL interpreter's general organization and its main components are shown in Fig. 11, stemming from [10].

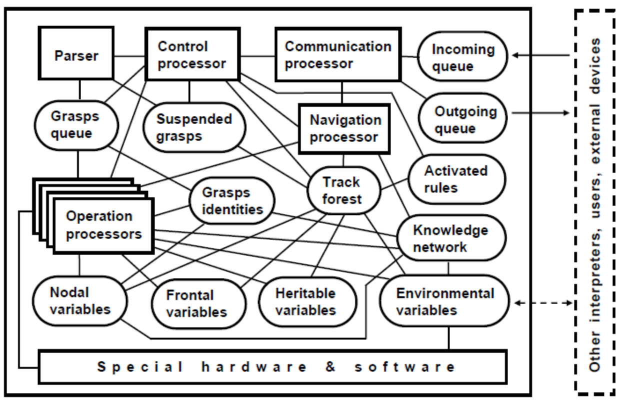


Fig. 11. SGL interpreter organization

The interpreter consists of a number of specialized functional processors handling, processing, and sharing specific data structures. The processors performing different language interpretation functions (shown by rectangles) include: communication processor, control processor, navigation processor, parser, different operation processors, also special hardware and software directly accessible from SGL. Main data structures on which these processors operate (shown by ovals) comprise: grasps queue, suspended grasps, track forest, activated rules, knowledge network, grasps identities, heritable variables, fontal variables, nodal variables, environmental variables, incoming queue, and outgoing queue.

SGL interpreter can communicate with other interpreter copies where the whole network of them can be mobile and open, capable of changing the number of nodes, their communication structure, and relative and absolute positions in physical space. This interpretation network can effectively implement and support global and local data structures, situation awareness, and overall system control by direct interpretation of evolving spatial SGL scenarios in parallel and fully distributed manner.

4.2 Spatial Track System

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 automatic 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 crated track trees (generally: forests), spanning the systems in which SGL scenarios evolve, are used for supporting spatial variables and provide echoing & merging of different types of control states and remote data, being self-optimized in parallel echo processes. They also route further grasps to the positions in physical, virtual or combined spaces reached by the previous grasps, uniting them with the frontal variables left there by the preceding grasps.

Figs 13-17 exhibit some operation stages of this distributed automatic command and control system, with abbreviations of the main components involved in them shown in Fig. 12.

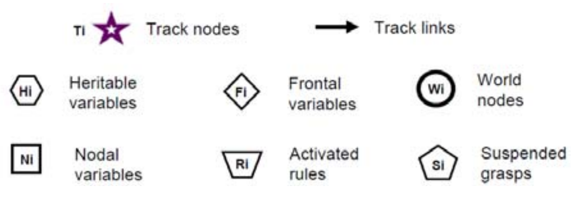


Fig. 12. Main track-based management components

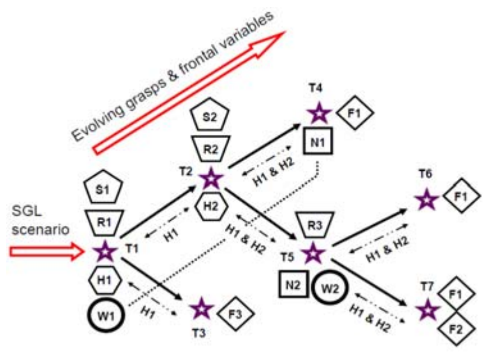


Fig. 13. Tracks creation in forward grasping

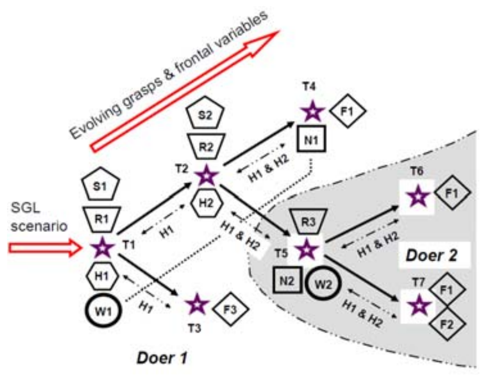


Fig. 14. Automatic distribution of track system between different doers

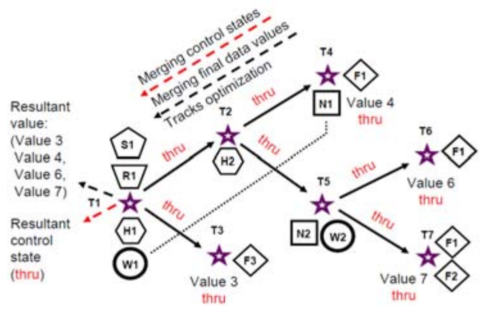


Fig. 15. Track echoing and optimization

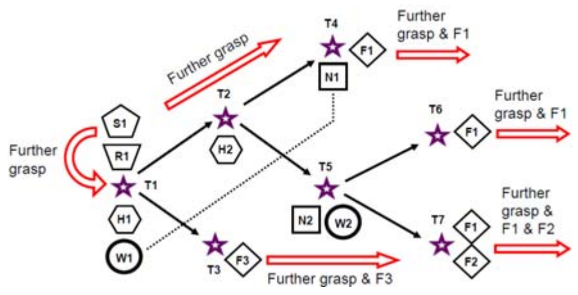


Fig. 16. Development of further grasps

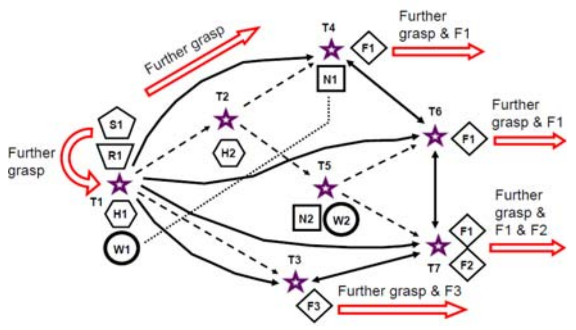


Fig. 17. More advanced track structure

4.3 Networked SGL Interpreter As a Universal Spatial Machine

The whole network of the interpreters can be mobile and open, changing at runtime the number of nodes and communication structure between them. Copies of the interpreter can be concealed if to operate in hostile environments, allowing us to analyze and impact the latter in a stealth manner, if needed.

The dynamically networked SGL interpreters are effectively forming a sort of a *universal parallel spatial machine* (as shown in Fig. 8) capable of solving any problems in a fully distributed mode, without any special central resources. "Machine" rather than computer as it can operate with matter too, and can move partially or as a whole in physical environment, possibly, changing its distributed shape and space coverage. This machine can operate simultaneously on many mission scenarios which can be injected at any time from its arbitrary nodes.

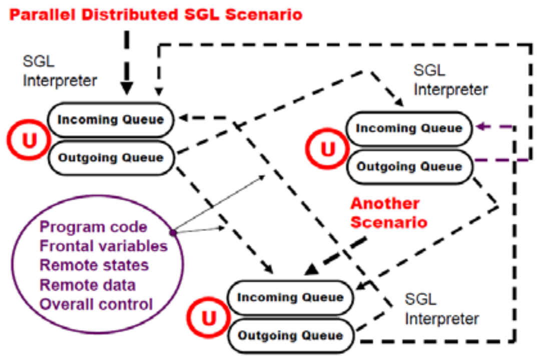


Fig. 18. SGL interpretation network as a universal spatial machine

Installing communicating SGL interpreters into mobile robots (ground, aerial, surface, underwater, space, etc.) on top of their existing functionality allows us to organize effective group solutions 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 drastically assisted too.

Some possible integrative scenario skeletons uniting dissimilar types of robotic units (ground, surface, underwater, space) operating under the unified C2 automatically provided via embedded SGL interpreters communicating with each other are shown in Fig. 19.

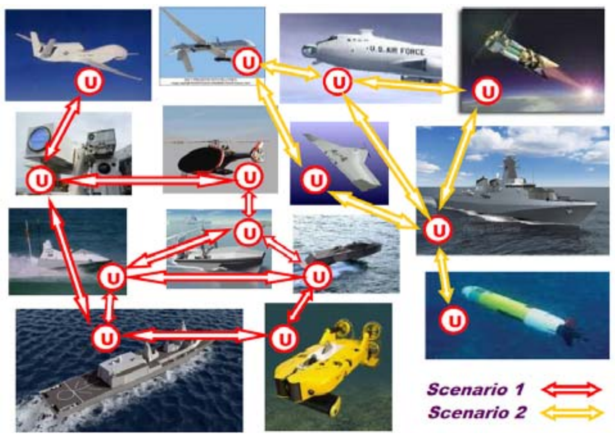


Fig. 19. Integrated distributed robotics with SGT

5 Examples of Distributed Programming in SGL

5.1 Finding Weakest Points

To find the weakest nodes in a graph like articulation points (see Fig. 20), which when removed split it into disjoint parts, the following program suffices (resulting in node d which is chosen to be physically removed, say, for a specific application).

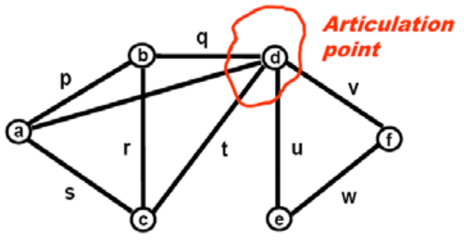


Fig. 20. Finding weakest points

```
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 (NAME))
```

This parallel and distributed SGL scenario 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 the network.
- Removing the node.

5.2 Finding Strongest Parts

Cliques (or maximum fully connected sub-graphs of a graph, as in Fig. 21), 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 Fig. 21 in cliques: (a, b, c, d), (c, d, e), and (d, e, f). These cliques are then chosen to be output locally rather than removed, as in the previous case.

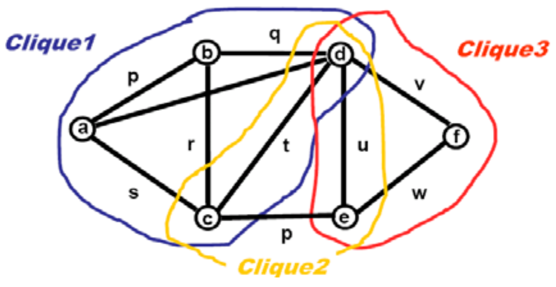


Fig. 21. Finding strongest parts

```
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.

5.3 Finding Arbitrary Structures in Arbitrary Networks by Parallel Pattern Matching

Any structures in any distributed networked systems can be found by describing them in SGL, like the one in Figure 22, which can be applied from any network node, evolving subsequently in a parallel replication and pattern-matching mode. The following SGL program, reflecting the search pattern (template) of Figure 22 (with variable nodes X1 to X6), is based on a path through all template's nodes.

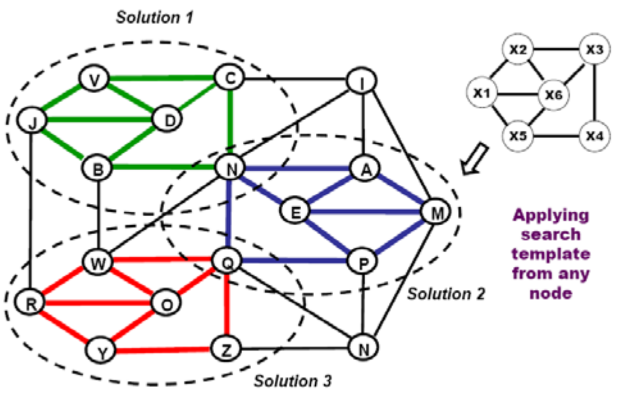


Fig. 22. Finding arbitrary structures in arbitrary networks

```
frontal (Match); hop (all nodes);
(repeat, 5) (append (Match, NAME); all links #;
                 not belong (NAME, Match));
if (and (any link # Match [2, 3]),
     (append (Match, NAME); all links # Match [1];
       if (any link # Match [5], OUT = Match)))
```

Three substructures have been found by the template in Fig. 22, with template variables matching the following network nodes:

(X1, X2, X3, X4, X5, X6) →
  (J, V, C, N, B, D), (M, A, N, Q, P, E), (R, W, Q, Z, Y, O)

More on parallel and distributed operations on general graphs and networks may be found in [11], where the SGL's predecessor WAVE language was used.

5.4 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 targets seeing locally from their different points, as shown in Fig. 23, and by the following DSL program.

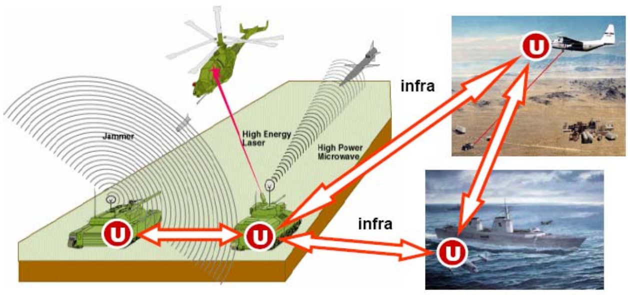


Fig. 23. Providing overall awareness and global targeting in a distributed space

```
loop (
  frontal (Seen) =
    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 SGL interpreter installed (communication links between the interpreters, which can be dynamic and casual, are represented as infra).

6 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., [12, 13]) 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, SGL allows for effective and universal expression of any battlefield scenarios and orders in parallel and fully distributed manner, also allowing for their straightforward implementation in robotized up to fully robotic systems. SGL scenarios are much shorter and simpler, as in the following example taken from [13] (Fig. 24 and the following 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.

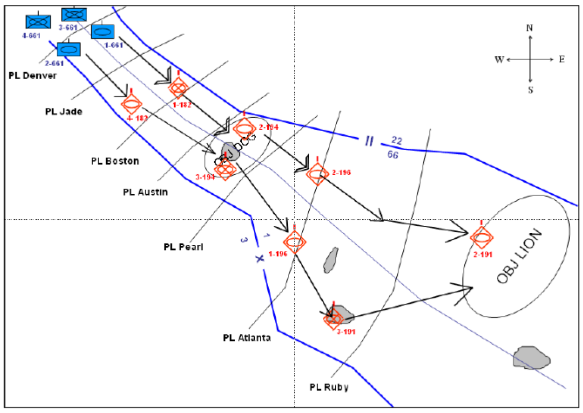


Fig. 24. Coordinated advancement in physical space

Tasks assigned to Coy1 in BML are 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-o33 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-o18 label-o17;
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 SGL (reflecting what to do in a distributed space and which key decisions to make rather than who/what will be doing this) 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 SGL interpretation). This can effectively relieve human commanders from multitude of traditional explicit C2 routines, allowing them concentrate on global mission objectives and efficiency instead.

```
FIXER = BN_661_Coy1;
SUPPORTER_DESTROYER = BN_661_Coy3;
advance (
  deploy (Denver, TFIN = TP0),
  move_destroy (PL: Boston,
     TARGET: Red_1_182, TFIN = TP1),
   move_destroy (PL: Austin, OBJ: DOG,
      TARGET: Red_2_194, TFIN = TP2),
   move_destroy (PL: Atlanta,
      TARGET: Red_2_196, TFIN = TP3),
   move_destroy (PL: Ruby, OBJ: LION,
      TARGET: Red_2_191, TFIN = TP4));
seize (LION, TFIN = TP4)
```

Any further scenario generalization in SGL can be provided within the same SGL syntax, as follows.

Not Mentioning Own Forces:

```
advance (
  deploy (Denver, TFIN = TP0),
   move_destroy (PL: Boston,
      TARGET: Red_1_182, TFIN = TP1),
   move_destroy (PL: Austin, OBJ: DOG,
      TARGET: Red_2_194, TFIN = TP2),
   move_destroy (PL: Atlanta,
      TARGET: Red_2_196, TFIN = TP3),
   move_destroy (PL: Ruby, OBJ: LION,
      TARGET: Red_2_191, TFIN = TP4));
seize (LION, TFIN = TP4)
```

Not mentioning adversary's forces:

```
deploy (Denver, TFIN = TP0);
move (PL: Boston, TFIN = TP1);
move (PL: Austin, OBJ: DOG, TFIN = TP2);
move (PL: Atlanta, TFIN = TP3);
move (PL: Ruby, OBJ: LION, TFIN = TP4));
seize (LION, TFIN = TP4)
```

Setting main stages only:

```
deploy (Denver, TFIN = TP0);
advance (PL: Boston, Austin, Atlanta, Ruby);
seize (LION, TFIN = TP4)
```

Final goal only:

```
seize (LION, TFIN = TP4)
```

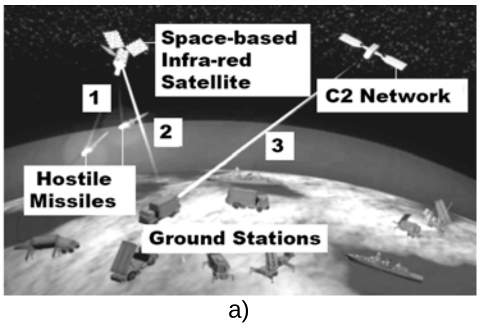
Expressing operations in the integral spatial formalism provided by SGL directly operating with distributed spaces enables us to drastically clarify and simplify mission descriptions and increase flexibility of their possible implementations with any available resources, both manned and unmanned, which can appear and change at runtime.

Many other applications of the spatial grasp paradigm can be found in [14-27], some examples exhibited below.

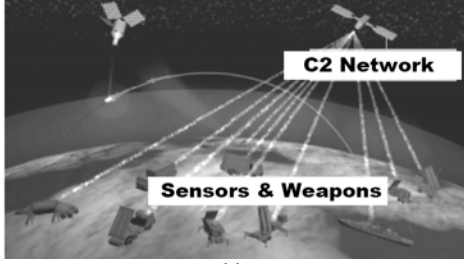
7 Other Application Scenarios

7.1 Europe-Related Missile Defense

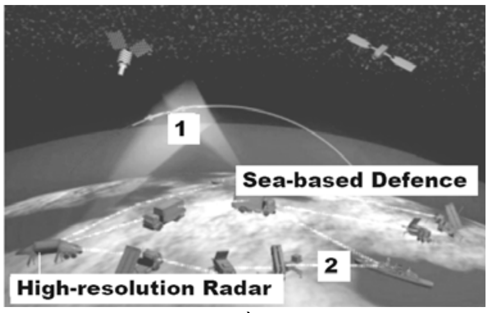
Let us consider here some scenarios related to the European missile defense plans, copied in Fig. 25.



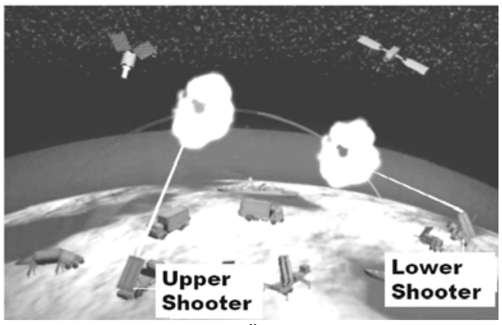
a)



b)



c)



d)

Fig. 25. Possible European missile defense scenarios.

The missile defense system is supposed to work in the following stages:

a) 1: Infrared satellite system picks up heat signatures of hostile missiles launched towards target. 2: Information transmitted to ground stations for processing. 3: Processed information sent to C2 network;

b) The C2 network relays information to sensor and weapons systems in the region;

c) 1: Long-range sensors continue to track the missile to help command system calculate options for destroying them. 2: Information is constantly shared among the sensors and weapons systems;

d) Command system has the option of shooting down the hostile missiles while in the upper or lower layers of the atmosphere.

Having extended these with advanced capabilities like DEW (say, high power lasers) located in space or on airborne (manned or UAV) platforms (synchronized with infrared satellite sensors), we can write the following very simple DSL scenario integrating infrared satellites, DEW facilities, long range sensors and upper and lower layer shooters into a dynamic distributed system capable of discovering hostile objects, tracing them at different stages of flight, and (re)launching target impact facilities with verification of their success or failure, until the targets are destroyed.

```
hop (infrared_satellite_sensors);
loop (
  nonempty (New = infrared (new_targets));
  release (
    split (New); frontal (Target) = VALUE;
    cycle (
      visible (Target); update (Target); hop (DE);
      if (try_shoot_verify (Target), done));
      hop (long_range_sensors);
      cycle (
        visible (Target); update (Target);
        if (distsance (Target) > threshold,
          hop (upper_layer_shooters),
          hop (lower_layer_shooters))
        if (try_shoot_verify (Target), done))));
```

The advantages of this scenario are that it can be initially applied to any available system component, automatically creating distributed C2 infrastructure particularly oriented on the currently discovered targets and dynamic situations. The automatically created distributed system organization can also self-recover at runtime after indiscriminate damages to any system components mentioned above (due to fully interpreted, mobile, virus-like implementation of SGL in distributed networked spaces).

7.2 Distributed Hostility Reconnaissance Scenario

An SGL solution is presented below where distributed physical space is randomly searched by simultaneous propagation of multiple reconnaissance units, which when discover unwanted activities encircle hostile zones, collect their perimeter coordinates, transfer them to mission headquarters (HQ), and initiate massive impact on the zones. Close to initial and final stages of this scenario are depicted in Figs 26 and 27.

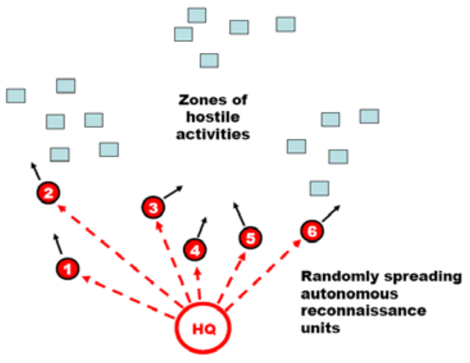

Zones of hostile activities
1
2
3
4
5
6
HQ
Randomly spreading autonomous reconnaissance units

Fig. 26. Initial development

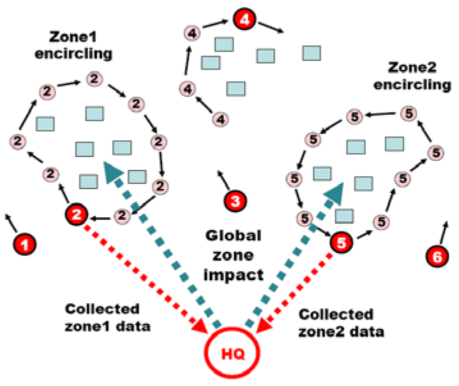


Fig. 27. Encircling and impacting hostile zones

```
move (*HQ*); create (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 (massive, Zone); done)))))
```

7.3 Distributed Objects Tracking by Mobile Intelligence

SGT allows us to use distributed sensor networks as highly integral self-organizes systems discovering, tracing, analyzing and impacting single and multiple mobile objects on vast areas despite physical limitations of individual sensor nodes.

Single Object Tracking in a Sensor Network

For a single object moving through the controlled area (as in Fig. 28), the following program starting in all sensors *catches the object* it sees and then *follows* wherever it goes, if not seen from the current point any more (i.e. its visibility becomes lower than a given threshold).

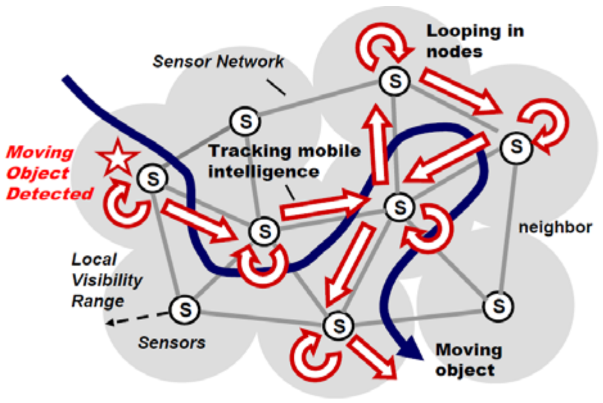


Fig. 28. Single object tracking

```
frontal (Object, Threshold = 0.1);
hop (all sensors); Object = search (aerial);
visibility (Object) > Threshold;
repeat (
      loop (visibility (Object) >= Threshold);
      max_destination (
                hop (neighbor, all); visibility (Object));
      if (visibility (Object) < Threshold),
          (output (Object & 'lost'); stop)))
```

Multiple Objects Tracking & Shooting

The following SGL scenario dealing with multiple object/targets including their shooting (as in Fig. 29) operates as:

- Each sensor is 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, decreasing the number of available kill vehicles in the region.
- If the target is hit, it is removed form the observation.

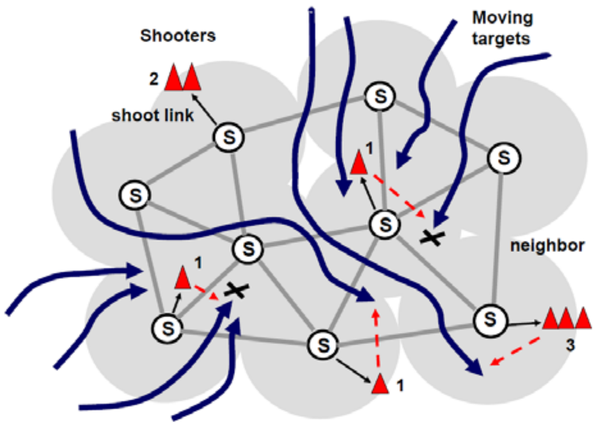


Fig. 29. Multiple Objects Tracking

```
Nodal (Seen);
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));
              if (visibility (Object) < Threshold),
                  (output (Object & 'lost'); stop)))))
```

7.4 Swarm-Against-Swarm Scenario

This describes a collective fight of a friendly unmanned swarm (consisting of "chasers") against group target which can be another robotic swarm or a manned team (or mixed), as in Fig. 30. The following SGL scenario operates by the following rules:

- 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.
- The most peripheral targets are considered particularly dangerous too as having better chances to escape from chasers and cause damage.
- Assigning available free chasers to targets, classifying them as *engaged*, and subsequently returning them back to status *free* (if 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 any external influence.

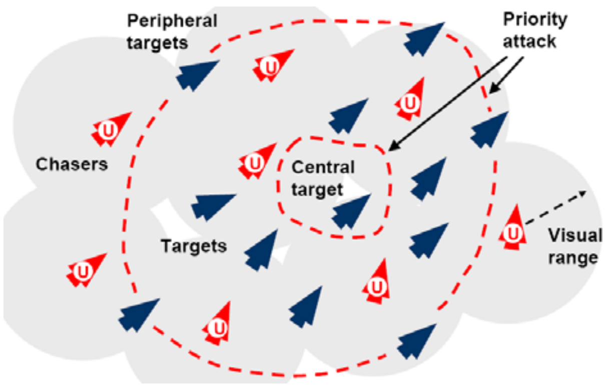


Fig. 30. Swarm against swarm operation

The scenario in SGL will look like follows:

```
nodal (Targets, Aver, List, Chaser); frontal (Next);
sequence (
  Initial launch (chasers, targets (Seen)),
  repeat (
    hop (random, free chasers);
    Targets = (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 = engaged;
    pursue_shoot_verify (Next); STATUS = free))))
```

Many more SGT applications in various areas can be found in existing publications on this approach [14-27].

8 Conclusions

We have briefed a new type of ideology and resulting networking technology aimed at establishing global control and supervision of distributed systems with any electronic means of communication and data processing embedded.

Within the technology developed, it is possible to describe in a special high-level language any local and global operations and control in both physical and virtual worlds and set up and supervise their behavior, including world's modifications and initial creation. The approach also allows us to 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 communication and management means while establishing powerful over-operability layer on top of them.

On the implementation layers, SGT effectively 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. 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, billions of mobile phone users, the technology's scalability and its virus-like nature, it can help launch and supervise global world missions in a great variety of areas including environmental protection, education, demographics, economy, space research, security, and defense.

References

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

[2] Feliciano CN (2009), The army's future combat system program (Defense, Security and Strategy Series). Nova Science

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

[4] P. Sapaty, "The Over-Operability Organization of Distributed Dynamic Systems for Asymmetric Operations", Proc. IMA Conference on Mathematics in Defence, Farnborough, UK, 19 November, 2009.

[5] P. S. Sapaty, "Over-Operability in Distributed Simulation and Control", The MSIAC's M&S Journal Online, Winter Issue, Volume 4, No. 2, Alexandria, VA, USA, 2002.

[6] M. Wertheimer, "Gestalt Theory", Erlangen. Berlin, 1925.

[7] P. Sapaty, "Gestalt-Based Ideology and Technology for Spatial Control of Distributed Dynamic Systems", International Gestalt Theory Congress, 16th Scientific Convention of the GTA, University of Osnabrück, Germany, March 26 - 29, 2009.

[8] P. Sapaty, "Gestalt-Based Integrity of Distributed Networked Systems", SPIE Europe Security + Defence, bcc Berliner Congress Centre, Berlin Germany, 2009.

[9] Wilber K (2009), Ken Wilber online: waves, streams, states, and self—a summary of my psychological model (or, outline of an integral psychology). Shambhala Publications

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

[11] Sapaty PS (1999), Mobile processing in distributed and open environments. John Wiley & Sons, New York.

[12] U. Schade, M. R Hieb, "Formalizing Battle Management Language: A Grammar for Specifying Orders", Paper 06S-SIW-068, 2006 Spring Simulation Interoperability Workshop (Paper 06S-SIW-068), Huntsville, Alabama, April 2006.

[13] U. Schade, M. R. Hieb, M. Frey, K. Rein, "Command and Control Lexical Grammar (C2LG) Specification", FKIE Technical Report ITF/2010/02, July 2010.

[14] P. S. Sapaty, "Withstanding Asymmetric Situations in Distributed Dynamic Worlds", invited paper, Proc. 17th International Symposium on Artificial Life and Robotics (AROB 17th '12), B-Con Plaza, Beppu, Oita, Japan, January 2012.

[15] P. S. Sapaty, “Meeting the World Challenges with Advanced System Organizations”, book chapter in: Informatics in Control Automation and Robotics, Lecture Notes in Electrical Engineering, Vol. 85, 1st Edition, Springer, 2011.

[16] P. S. Sapaty, “Distributed Technology for Global Dominance” Proc. SPIE 6981, Defense Transformation and Net-Centric Systems 2008, Raja Suresh, Ed., 69810T, 2008.

[17] P. S. Sapaty, “Ruling Distributed Dynamic Worlds”, John Wiley & Sons, New York, 2005.

[18] P.S. Sapaty, M.J. Corbin, S. Seidensticker, "Mobile Intelligence in Distributed Simulations", Proc. 14th Workshop on Standards for the Interoperability of Distributed Simulations, IST UCF, Orlando, FL, March 1995.

[19] P. Sapaty, V. Klimenko, M. Sugisaka, “Dynamic Air Traffic Management Using Distributed Brain Concept”, Proc. Ninth International Symposium on Artificial Life and Robotics (AROB 9th), Beppu, Japan, January 2004.

[20] P. Sapaty, M. Sugisaka, “Optimized Space Search by Distributed Robotic Teams”, Proc. World Symposium Unmanned Systems 2003, Jul. 15-17, 2003, Baltimore Convention Center, USA.

[21] P. Sapaty, M. Sugisaka, J. Delgado-Frias, J. Filipe, N. Mirenkov, “Intelligent management of distributed dynamic sensor networks”, Artificial Life and Robotics, Volume 12, Numbers 1-2 / March, ISSN: 1433-5298 (Print), 1614-7456 (Online), Springer Japan, pp. 51-59, 2008.

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

[23] P. Sapaty, M. Sugisaka, “Countering Asymmetric Situations with Distributed Artificial Life and Robotics Approach”, Proc. Fifteenth International Symposium on Artificial Life and Robotics (AROB 15th’10), B-Con Plaza, Beppu, Oita, Japan, Feb. 5-7, 2010.

[24] P. Sapaty, K.-D. Kuhnert, M. Sugisaka, R. Finkelstein, “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, Japan, Feb. 5-7, 2009.

[25] P. Sapaty, M. Sugisaka, J. Delgado-Frias, J. Filipe, N. Mirenkov, “Intelligent management of distributed dynamic sensor networks”, Artificial Life and Robotics, Volume 12, Numbers 1-2 / March, 2008, ISSN: 1433-5298 (Print) 1614-7456 (Online), Springer Japan, pp. 51-59.

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

[27] P. Sapaty, A. Morozov, M. Sugisaka, “DEW in a Network Enabled Environment”, Proc. international conference Directed Energy Weapons 2007, Feb. 28 - March 1, 2007, Le Meridien Piccadilly, London, UK.