TNC Transactions on Networks and Communications
SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM
VOLUME 2, ISSUE 2
ISSN: 2054 - 7420

Unified Transition to Cooperative Unmanned Systems under Spatial Grasp Paradigm

Peter Simon Sapaty
Institute of Mathematical Machines and Systems, National Academy of Sciences of Ukraine;
sapaty@immsp.kiev.ua, peter.sapaty@gmail.com

ABSTRACT

A novel philosophy, ideology, methodology, and supporting high-level networking technology will be revealed capable of guiding gradual transition to intelligent unmanned systems with a variety of important practical applications. The approach is based on a completely different type of high-level language capable of grasping top semantics of complex spatial operations in dynamic and unpredictable environments while shifting numerous technical details to effective automatic implementation. The language is based on holistic and gestalt principles rather than traditional multi-agent organizations, providing high integrity and super-summative features of the solutions described. Cooperative networked interpretation of the language in distributed systems and different parallel and distributed scenarios in it will be demonstrated that can be performed by any combination of manned and unmanned components under unified command and control provided by the technology described.

Keywords: Distributed Systems; Gestalt Philosophy; Spatial Grasp Language; Networked Interpretation; Spatial Scenarios; System Integrity; Unmanned Systems; Human-Robotic Integration.

1. INTRODUCTION

Unmanned systems are widely used today, and their role will definitely be increasing in the future. Their further development and engagement will, however, largely depend on how efficiently and naturally robotic means unite with manned systems within overall human activity. There should be clear philosophical, methodological, linguistic and technological grounds for manned-unmanned integration, division of jobs between humans and robots, and common command and control for combined missions.

Within this global context we are pursuing a *tasking approach* aimed at formalizing and describing problems to be solved and tasks to be executed in physical and virtual environments which may need robotic involvement. Effective tasks definitions may help to define which

DOI: 10.14738/tnc.22.83
Publication Date: 3rd April 2014
URL: http://dx.doi.org/10.14738/tnc.22.83

human and which robotic components should be used, and how they must be organized as a system to perform the tasks needed in most efficient way.

And this approach should allow for a general enough, semantic level of task presentations to allow maximum flexibility of their implementation with possibly unknown in advance and scarce resources, as well as for easiness of their redefinition when conditions, goals, and states of environment change. This will allow us to be in line with growing world dynamics and withstand numerous asymmetric situations and threats the humankind is facing, which may need asymmetric solutions too, and with broad engagement of advanced robotic means.

As we will be touching system organizations throughout this paper, let us provide some comments on and comparison of different system organizations.

The *traditional* approach to system design, development and management supposes the system structure and system organization to be predominantly primary, created in advance, whereas global function and overall behavior appearing as secondary, like in Figure 1.

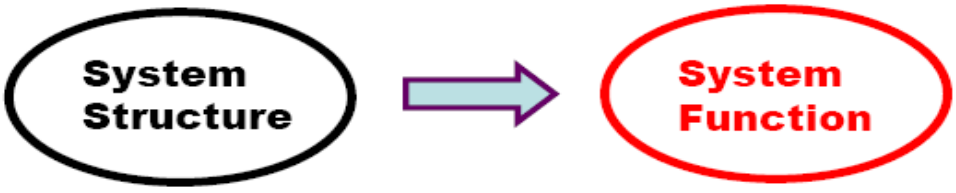

Figure 1: Traditional Approach to System Design

The systems based on this vision, ideologically relevant to multi-agent systems [1] still prevailing nowadays, are often clumsy and static, they may fail to quickly adapt to 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 systems to new goals and functionality may result in a considerable loss of their integrity and performance.

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

We are pursuing an *alternative* system approach, 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 (see Figure 2).

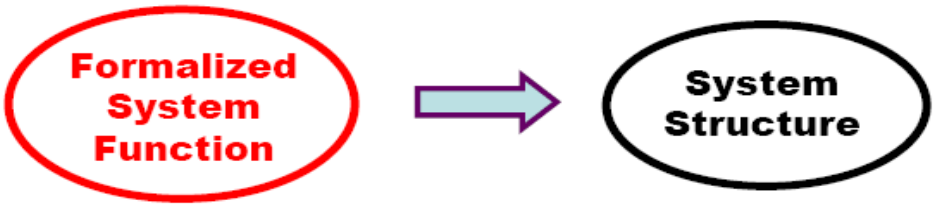

Figure 2: The Alternative System Organization Considered

The advantages of such (actually, the other way round) organization may include high potential flexibility of runtime system creation and management, especially in quick responses to asymmetric events. This quality allows us formulate top semantics of the needed reaction on world events in a special high level language, shifting most of traditional organizational routines to automated up to fully automatic implementation, with effective engagement of unmanned systems evolutionally and most naturally.

The related paradigm and accompanying networking technology we are developing is based on formalized wavelike seamless navigation, coverage, or grasping of distributed physical and virtual spaces, as symbolically shown in Figure 3. This believably inherits and psychologically matches of how human mind operates [3], especially in comprehension of distributed environments, in a holistic [4], gestalt-based [5-7], and integral way, and finds complex spatial solutions in them. These features are placed in our case on advanced highly parallel and fully distributed networking platforms often exhibiting clear advantages before intelligent biological systems in specific, especially distributed applications.

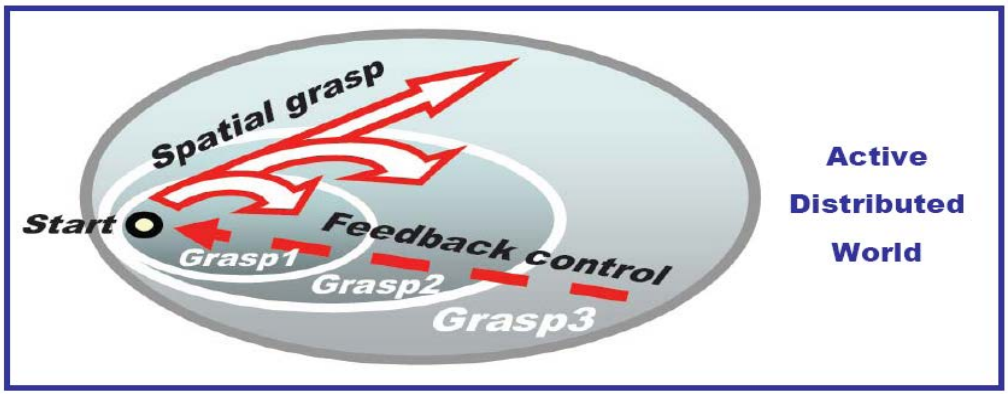

Figure 3: Incremental Spatial Grasp of Distributed Worlds

The approach in general works as follows. A network of universal control modules U, embedded into key system points (like humans, robots, smart sensors, mobile phones including), collectively interprets mission scenarios expressed in a special high-level Spatial Grasp Language (SGL), as shown in Figure 4. These scenarios, capable of representing any parallel and distributed algorithms, can start from any node while covering the whole system or its parts needed at runtime.

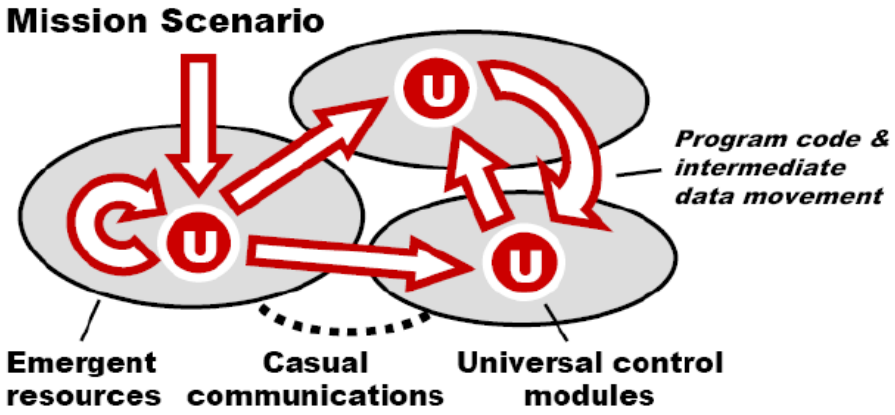

Figure 4: Collective Scenario Execution in Distributed Dynamic Environments

SGL scenarios, often expressing top semantics of spatial operations, are very compact and can be created on the fly. Different scenarios can cooperate or compete in a networked space as overlapping fields of solutions. Self-spreading scenarios can create runtime knowledge infrastructures distributed between system components, as shown in Figure 5. These can effectively support distributed databases, advanced command and control, global situation awareness, as well as any other computational or control models.

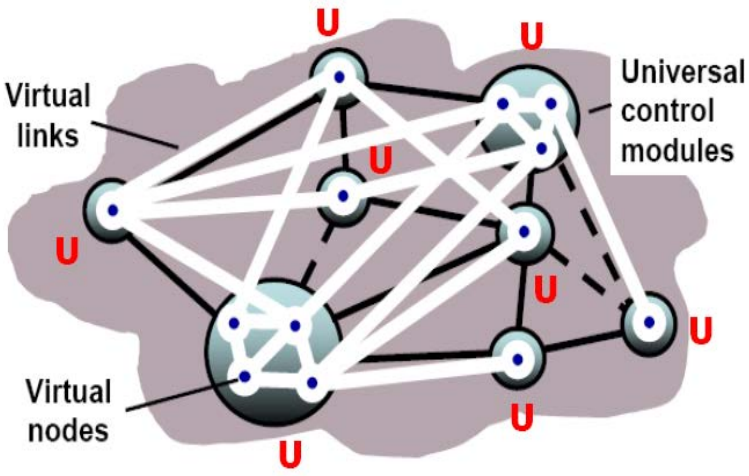

Figure 5: Creating Distributed Knowledge Infrastructures

This paper represents the first report on the latest version of SGL with extended repertoire of its functions (or rules). It also describes the related updated structure of the SGL interpreter as well as presents a number of cooperative robotic applications of SGT confirming simplicity and compactness of SGL-based scenarios (usually at least an order of magnitude more compact than in other known languages like Java for general data processing or BML [8] for command and control applications).

The development history, various philosophical and technological aspects of this Spatial Grasp Technology (SGT) as well as detailed descriptions of its researched areas can be found in many existing publications, including [10-20].

2. SPATIAL GRASP LANGUAGE

2.1 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. An SGL program (rather: *scenario*, to highlight its generality and orientation on direct problem solving in real worlds by both manned and unmanned components) can be viewed from different angles:

- As the first linguistic means to describe and formalize the notion of *gestalt* [5-7] allowing us to effectively grasp top semantics, integrity, and super-summative features of large complex systems.
- As an active recursive *self-matching pattern* applied against distributed physical, virtual, executive, or combined worlds, ruling and changing these worlds appropriately.
- As a universal *genetic* mechanism, expressed in a special integral formalism, allowing any distributed systems, whether passive or active, to be created, grown, evolved, and modified while starting from any world point. This also relates to the relatively new concept called *memetics* [9].
- As a sort of a *"soul"* (very symbolically, however) to be injected into and spread in a controlled manner throughout the distributed world, giving it new life, breath, and consciousness providing the needed both local and global awareness and control.

2.2 The SGL Worlds

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.

2.3 Top SGL Syntax

SGL has recursive structure top level of which is shown in Figure 6. 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.

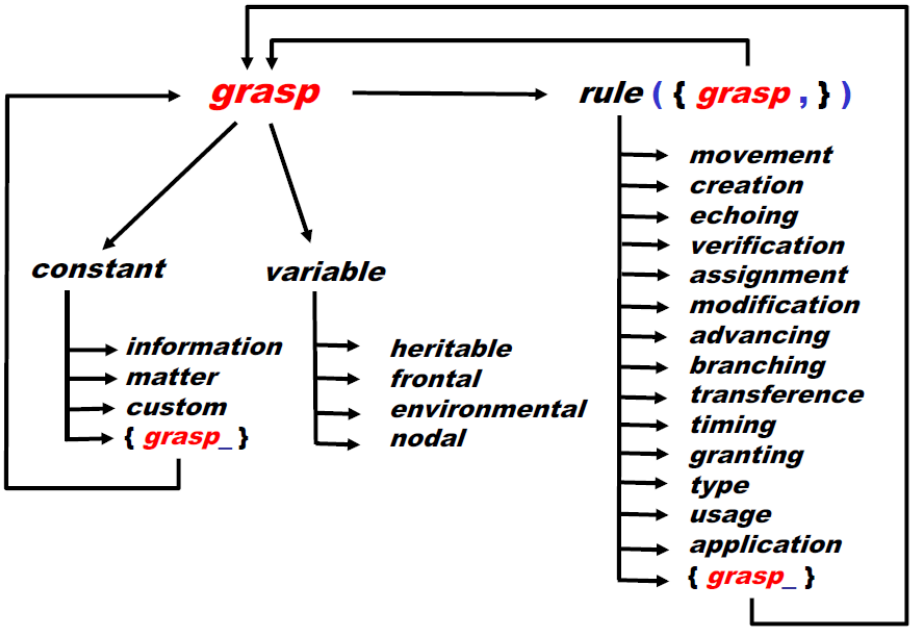

Figure 6: SGL Recursive Syntax

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

The SGL topmost definition, where scenario in it is named as *grasp* (reflecting the spatial navigation-grasp-conquest model explained in previous chapters) 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 of no need any more. This movement can take place in single or multiple, parallel, same or different scenario 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 (to be explained in detail afterwards):

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 explicit 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 to jointly work on the operands they commonly embrace.

2.4 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, which proved to be very useful on both conceptual and implementation levels, we can clearly explain how SGL scenario operates and evolves, with key points 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 control.

2.5 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.

2.6 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.

2.7 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.

2.8 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. 6. A number of such code simplifications will be used in the following sections when describing different scenarios in SGL for solving concrete problems.

2.9 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 of 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, explained later, 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`

2.10 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*
transference → `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` |

3. DISTRIBUTED SGL INTERPRETER

3.1 The Interpreter Components and Structure

The internal organization of SGL interpreter (in software, hardware or both) is shown in Figure 7. The interpreter consists of a number of specialized modules working in parallel and handling & sharing specific data structures supporting both persistent virtual worlds and temporary data and hierarchical control mechanisms.

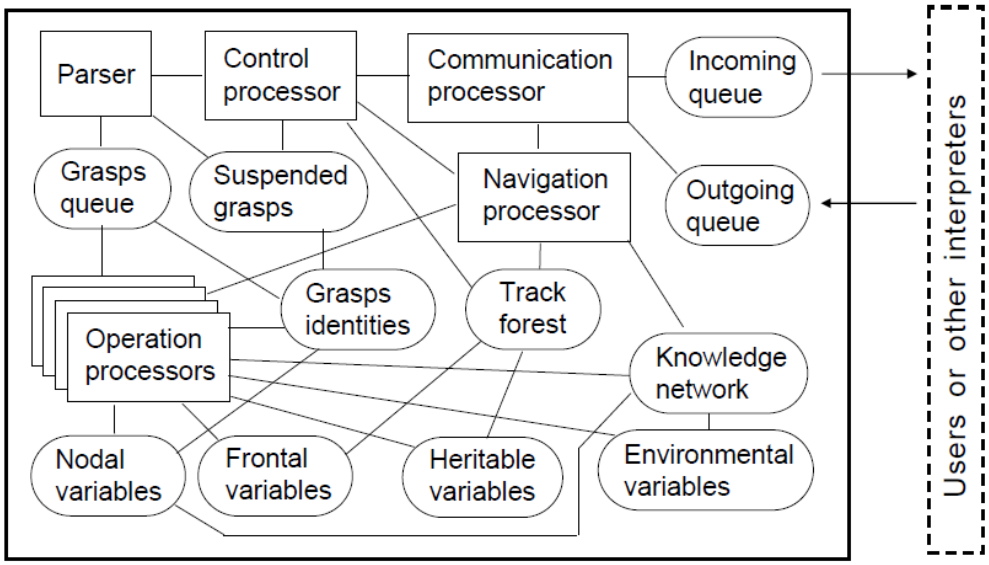

Figure 7: Organization of SGL Interpreter

The "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 similar parts in other interpreter copies, forming altogether *global control coverage*. This forest-like distributed track structure enables for hierarchical 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 echoing & merging different types of control states and remote data, self-optimizing in parallel echo processes and providing automatically of what is usually called (adaptive) command and control, or C2. They also route further grasps to the positions in physical, virtual, execution or combined spaces reached by the previous grasps, uniting them with frontal variables left there by the preceding grasps.

3.2 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 *universal parallel spatial machine* (as shown in Figure 8) capable of solving any problems in a fully distributed mode, without any special central resources. "Machine" rather than computer or "brain" 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.

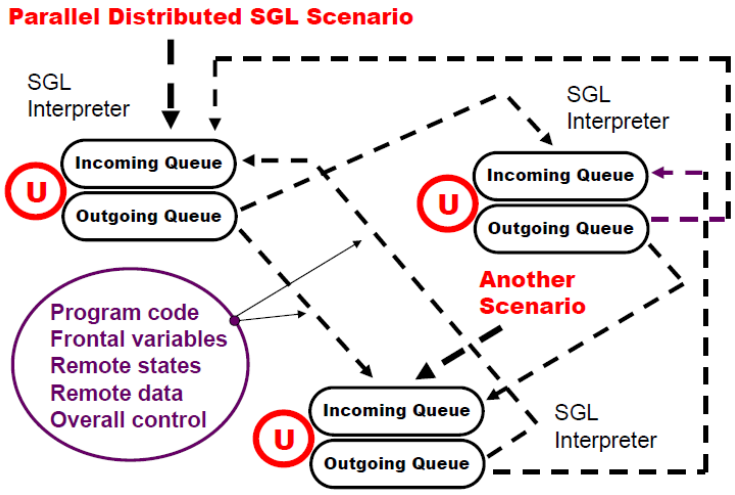

Figure 8: SGL interpretation Network as a Universal Spatial Machine

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 hypothetic integrative scenario skeletons, uniting very dissimilar types of robotic units (ground, surface, underwater, space), all operating under the unified command and control provided by SGT via embedded SGL interpreters communicating with each other, are shown in Figure 9.

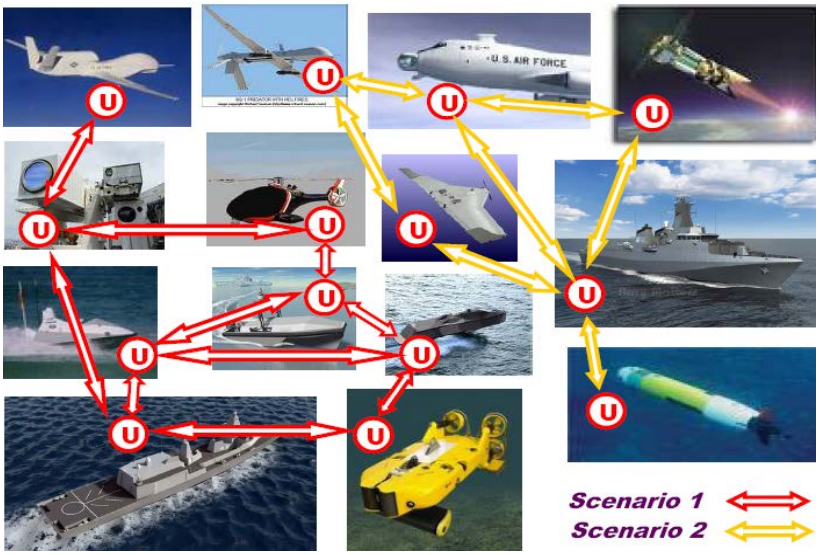

Figure 9: Possible Cooperative Scenario Skeletons

4. COOPERATIVE ROBOTICS

After embedding SGL interpreters into robotic vehicles, we can provide any needed collective behavior of them—from loose swarming to a strictly controlled integral unit obeying external orders. Any mixture of different behaviors within the same scenario can be easily programmed too.

We will consider here some collective robotic scenarios in SGL operating on different organizational levels and their integration under the unified control provided by the automatic language interpretation.

4.1 Integration of Loose Swarming with Hierarchical Command and Control

Imagine that a distributed area needs to be investigated by multiple unmanned aerial vehicles that should randomly search the space, collect information on unwanted objects, classifying them as targets, and organize collective reaction on emerging threats. Different group functionalities for this can be expressed in SGL which can be effectively integrated into the resultant holistic group scenario, as follows.

- Swarm movement scenario, starting from any unit (let us call this *swarm_move*):

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

A snapshot of such swarm movement is shown Figure 10.

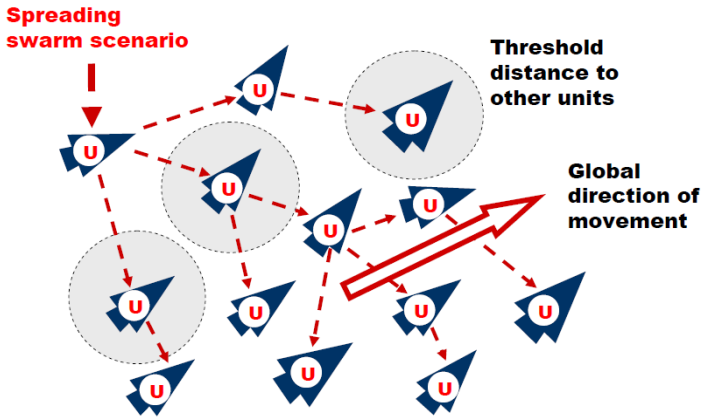

Figure 10: Swarm Movement Snapshot

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

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

The center found by this scenario is shown in Figure 11.

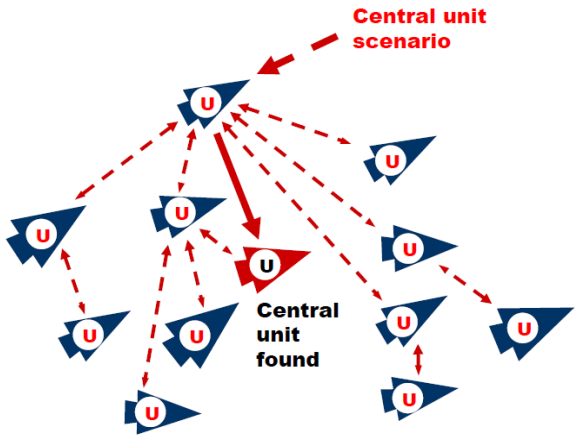

Figure 11: Finding Central Unit

- Creating runtime infrastructure starting from the central unit found (*infra_build*), see Figure 12:

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

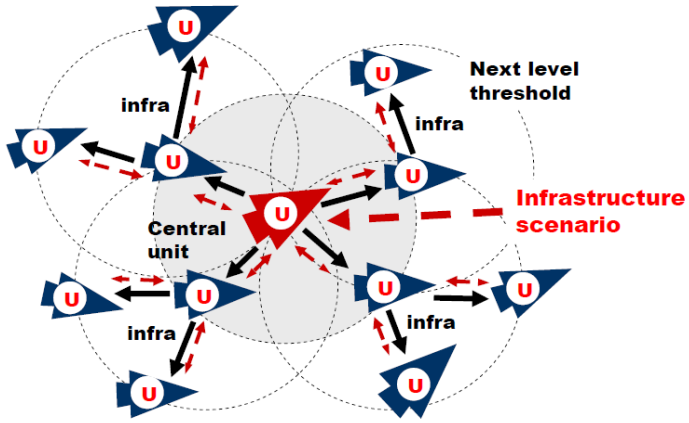

Figure 12: Runtime Infrastructure Creation

- Regular targets collection & distribution & impact (*collect_distribute_impact*) starting from the central unit found, as in Figure 13:

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

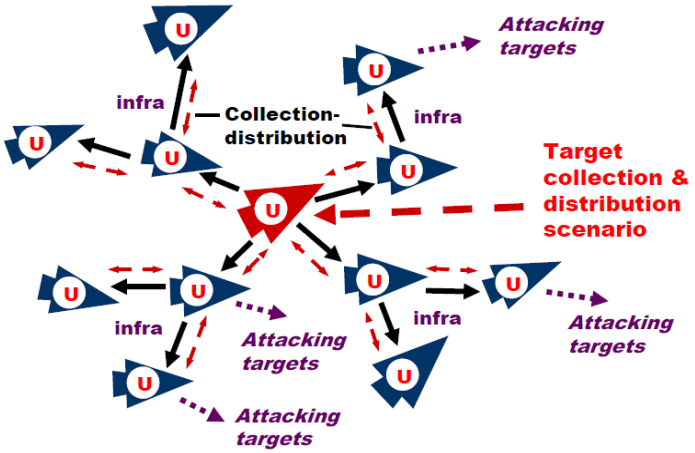

Fig. 13. Targets Collection, Dissemination, and Impact

- Removing previous infrastructure (before creating a new one), starting from any unit (*infra_remove*):

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

The resultant combined solution (integrating previous SGL programs named in italics), starting from any unit, will be as (where *time* is meant to be a certain time interval):

```
parallel(
  swarm_move,
  repeat(
```

```
find_hop_center;
infra_remove; infra_build;
remain(time_delay, collect_distribute_impact)))
```

The obtained resultant scenario combines loose, randomly oriented swarm movement in a distributed space with periodic updating of topologically central unit (as units are changing distances and relative positions) and updating runtime hierarchical infrastructure between them. This infrastructure controls observation of distributed territory while collecting potential targets, distributing them back to the vehicles for local assessment selection and impact.

4.2 Collective Patrol of Coastal Waters

This scenario may be suitable for both surface and varying depth underwater search of intrusions in the coastline zone, but for simplicity we will be assuming here only two dimensional space to be navigated.

At the beginning let us create a distributed coastal waypoint map in the form of embedded semantic network, as in Figure 14 (r being arbitrary name of links between nodes-waypoints). The corresponding DSL solution is as follows.

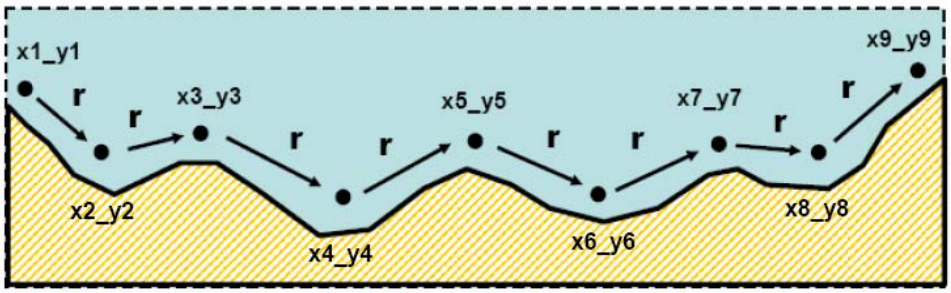

Figure 14: Coastal Waypoint Map

```
create_physical(
   #x1_y1; +r#x2_y2; +r#x3_y3; … +r#x9_y9)
```

where xi, yi represent concrete coordinates.

A single USV (or UUV) solution repeatedly navigating all coastal area by the map created is shown in Figure 15 and DSL program that follows (searching the water space for alien objects by the depth available by vehicle's sensors).

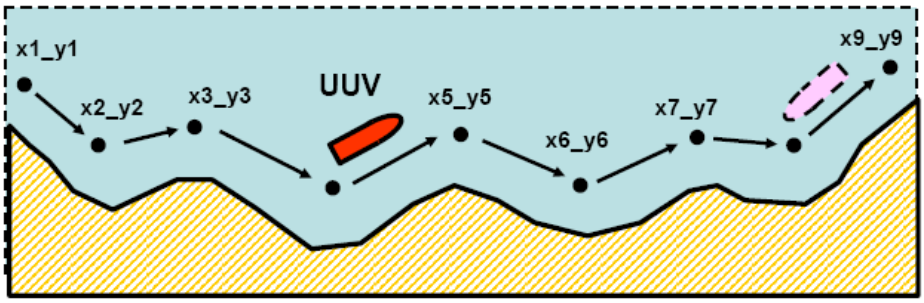

Figure 15: Patrolling Coastal Waters with a Single Vehicle

```
frontal(Link =+r, SearchDepth = …);
move(x1_y1);
repeat(
  repeat(move(Link); check_report(SearchDepth, alien));
  invert(Link))
```

Two-vehicle simultaneous solution is shown in Figure 16 and by the following program, with vehicles moving according to the coastal map independently, assuming each having automatic procedures for avoiding possible collisions with the other vehicle.

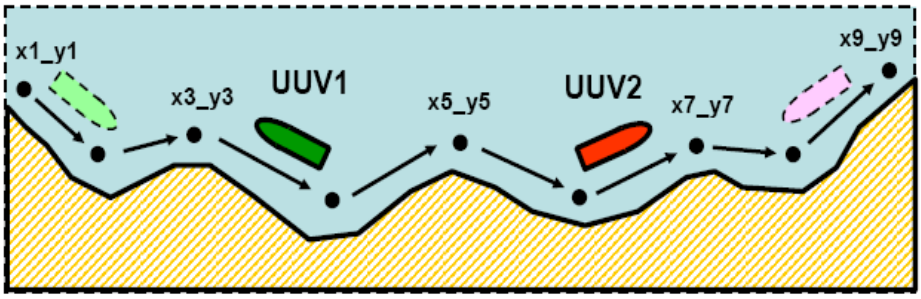

Figure 16: Patrolling coastal waters with two vehicles

```
frontal(Link, SearchDepth = …);
branch(
  (Link = +r; move(x1_y1)),
  (Link = -r; move(x9_y9)));
repeat(
  repeat(check_report(SearchDepth, alien); move(Link));
  invert(Link))
```

Another solution for the two-vehicle case may be when each vehicle turns back if discovers another patrol vehicle on its way, checking for this its vicinity with depth given.

```
frontal(Link, SearchDepth = …, CollisionDepth = …);
branch(
  (Link = +r; move(x1_y1)),
  (Link = -r; move(x9_y9)));
repeat(
  repeat(none_seen(CollisionDepth, patrol);
         check_report(SearchDepth, alien));
         move(Link));
  invert(Link))
```

For the both cases, the whole coastline will always be searched in full if at least a single vehicle remains operational. The current scenario can be easily extended to more then two vehicles searching space cooperatively.

4.3 Cooperative Finding of Oil Spill Center

This scenario tries to find oil spill center at sea by cooperating multiple surface or underwater unmanned vehicles distributed initially throughout the polluted region, as shown in Figure 17.

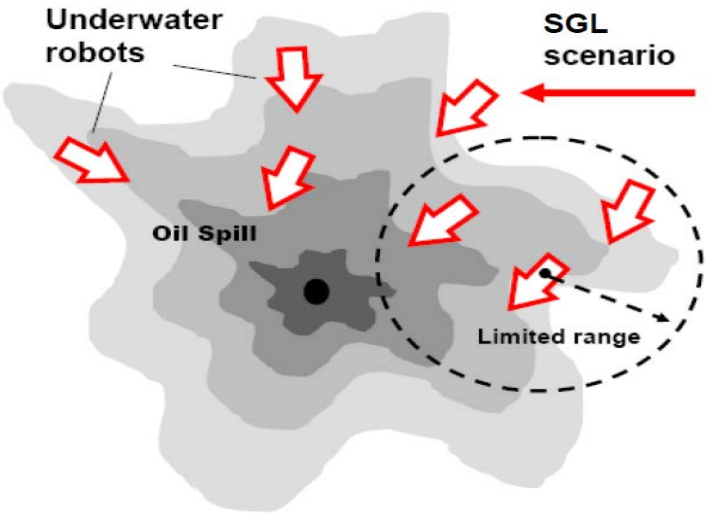

Figure 17: Cooperative Finding of the Spill Center

The distributed scenario operates as follows.

- It starts from any robot, covering the whole group by limited local channels.
- Each robot randomly tries to move toward increasing spill level,
- Each movement is allowed only if some other robots remain in reach, to preserve connectivity between robots as a network.
- By regular local communications between robots the maximum recorded oil spill level is updated in all robots.
- The robot(s), in which maximum level of the whole region corresponds to locally checked level for a threshold period of time, report the center of spill.

The expression of this scenario in SGL will be as follows (words in italics to be substituted by real values):

```
move(coord1, coord2, …, coordn);
nodal(Level = check(spill), Direction,
      Current, Count, Max);
parallel(
   loop(Max = maximum(
          (hop(range, all); Max), Max, Level);
        Level == Max != 0; Count += 1;
        if(Count >= threshold,
```

```
            output('Center', WHERE));
        sleep(Delay)),
    loop(Current = WHERE;
        or((WHERE += random(shift, Direction);
            nonempty(hop(range, all));
            New = check(spill) > Level;
            Level = New; Count = 0;
            Direction = angle(Current, WHERE)),
           WHERE = Current)))
```

5. CONCLUSIONS

We have described a new type of distributed management philosophy and supporting networking technology based on a completely different type of management language than usual. This language, SGL, is oriented on programming and processing of distributed both physical and virtual systems and worlds directly and on a high semantic level, allowing at the same time to express organization, management, and control details on any other levels too, if needed, which are usually shifted to automatic SGL interpretation in collaborative environments.

With the use of SGL, the whole distributed world can be considered as an integral and universal spatial machine capable of solving arbitrary complex problems (*machine* rather than *computer* as it directly operates with physical matter/objects too). Multiple communicating "processors" or "doers" of this machine can include humans, computers, robots, smart sensors, any mechanical/electronic equipment capable of cooperatively solving complex problems formulated in SGL.

Being understandable and suitable for both manned and unmanned components, the language offers a real support for unified transition to robotized systems as within execution of operational scenarios in it any components can easily change, at runtime including, their manned to unmanned status and vice versa, also enabling *fully unmanned solutions* if these may happen to be needed for specific applications.

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]. 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.

[4]. Smuts JC (2007), Holism and evolution. Kessinger Publishing, LLC.

[5]. Wertheimer M (1924), Gestalt theory. Erlangen, Berlin.

[6]. Sapaty P (2009), Gestalt-based ideology and technology for spatial control of distributed dynamic systems. Proc. International Gestalt Theory Congress, 16th Scientific Convention of the GTA. University of Osnabrück, Germany.

[7]. Sapaty P (2009), Gestalt-based integrity of distributed networked systems. SPIE Europe Security + Defence, bcc Berliner Congress Centre, Berlin Germany.

[8]. Schade U, Hieb MR (2006), Formalizing battle management language: a grammar for specifying orders. 06S-SIW-068, presented at the 2006 Spring Simulation Interoperability Workshop, April 2006, Huntsville, AL.

[9]. Tyler T (2011), Memetics: memes and the science of cultural evolution. CreateSpace Independent Publishing Platform.

[10]. Sapaty PS (2013), The world as an integral distributed brain under spatial grasp paradigm. Proc. International Science and Information (SAI) Conference, 7-9 October, London, UK.

[11]. Sapaty P (2012), Logic flow in active data. Book chapter in: VLSI for Artificial Intelligence and Neural Networks, Springer; Softcover reprint of the original 1st ed. 1991 edition.

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

[13]. Sapaty PS (2011), 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.

[14]. Sapaty PS (2009), Providing spatial integrity for distributed unmanned systems. Proc. 6th International Conference in Control, Automation and Robotics ICINCO 2009, Milan, Italy.

[15]. Sapaty P, Sugisaka M, Delgado-Frias MJ, Filipe J, Mirenkov N (2008), Intelligent management of distributed dynamic sensor networks. Artificial Life and Robotics, Volume 12, Numbers 1-2 / March, Springer Japan.

[16]. Sapaty P (2008), Distributed technology for global dominance. In: Raja Suresh (ed): Proc. of SPIE, Vol. 6981, Defense Transformation and Net-Centric Systems.

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

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

[19]. Sapaty PS, Corbin MJ, Seidensticker S (1995), Mobile intelligence in distributed simulations. Proc. 14th Workshop on Standards for the Interoperability of Distributed Simulations, IST UCF, Orlando, FL, March.

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

