Research Article
Volume 13:6, 2020
DOI: 10.37421/jcsb.2020.13.321

Journal of Computer Science & Systems Biology

ISSN: 0974-7230 **Open Access**

Symbiosis of Virtual and Physical Worlds under Spatial Grasp Technology

Peter Simon Sapaty*

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

Abstract

We are witnessing rapidly growing world dynamics caused by climate change, military, religious and ethnic conflicts, terrorism, refugee flows and weapons proliferation, political and industrial restructuring too. Dealing with frequently emerging crises may need rapid integration of scattered heterogeneous resources into capable operational forces pursuing goals which may not be known in advance. Proper understanding and managing of unpredictable and crisis situations may need their detailed simulation at runtime and even ahead of it. The current paper aims at deep integration, actually symbiosis, of advanced simulation with live system control and management, which can be effectively organized in nationwide and world scale. It will be presenting the latest version of Spatial Grasp Technology (SGT) which is not based on traditional communicating parts or agents, as usual, but rather using self-spreading, self-replicating, and self-modifying higher-level code covering and matching distributed systems at runtime while providing global integrity, goal-orientation, and finding effective solutions. These spatial solutions are often hundreds of times shorter and simpler than with other approaches due to special recursive scenario language hiding traditional system management routines inside its parallel and distributed interpretation. The paper provides basics for deep integration, actually symbiosis, of different worlds allowing us to unite advanced distributed simulation with spatial parallel and fully distributed control, while doing all this within the same high-level and very simple Spatial Grasp formalism and its basic Spatial Grasp Language (SGL). It will also mention various SGT applications including economy, ecology, space research & conquest and security, where effective symbiosis of distributed interactive simulation with live control and management may provide a real breakthrough. SGL can be quickly implemented even within standard university environments by a group of system programmers, similar to its previous versions in different countries under the author's supervision. The technology can be installed in numerous copies worldwide and deeply integrated with any other systems, actually acquiring unlimited power throughout the world.

Keywords: Distributed simulation • Integration of simulation with control • Physical world • Virtual world • Executive world • Spatial Grasp Technology • High level architecture • Parallel and distributed programming • Computational biology

Introduction

We are witnessing rapidly growing world dynamics caused by climate change, military, religious and ethnic conflicts, terrorism, refugee flows, weapons proliferation, political and industrial restructuring, inequality, economic instability, global insecurity, and very recently due to the world-wide pandemic horror [1-10]. Dealing with frequently emerging crises may need rapid integration of scattered heterogeneous resources into capable operational forces pursuing goals which may not be known in advance. Proper understanding and managing of unpredictable and crisis situations need their detailed simulation at runtime and ahead of it [11-30]. This may also need deep integration of advanced simulation with live control and management within united and enriching each other concepts of virtual, physical, and executive worlds, which should be effectively organized in both local and global scale [31-42].

The current paper provides basics for deep integration, actually symbiosis, of different worlds allowing us to unite advanced distributed simulation with spatial parallel and fully distributed control, while doing all this within the same high-level and very simple Spatial Grasp formalism and Technology (SGT), which was patented and revealed in numerous previous publications, Wiley, Springer, and Emerald books including [43-48]. The investigated applications included classical graph and network theory problems, missile defense, massive collective robotics, evolution of space systems, flexible command and control, industrial, social and international security problems, also effectively expressing main gestalt theory laws allowing them to cover any distributed systems rather than just human mind and brain. The developed formalism allows us to directly exist, operate, and move in different worlds and their combinations, while shifting traditional system organizations with numerous routines (DIS and HLA [19-27] including, which needed explicit programming) completely to automatic parallel and networked interpretation of its basic Spatial Grasp Language (SGL), making resulting solutions hundreds of times shorter and simpler.

The rest of the paper is organized as follows. Section 2 briefs the developed SGT which is not based on traditional interoperability principles and communicating parts or agents [49-55], but rather using self-spreading, self-replicating, and self-modifying higher-level code covering and matching distributed systems at runtime, while providing global integrity, goal-orientation, and finding effective solutions; he section also offers the latest version of SGL. Section 3 describes how different worlds (i.e. physical, virtual, executive) can be represented in SGT separately and effectively programmed in SGL. Section 4 shows in SGL how different worlds can be combined with each other, including all three together, and which benefits such integration may offer.

It is shown in Section 5 how the opposite business can be done, in reducing the integration and mutual penetration of different worlds up to their sole representations and even elimination. Section 6 explains how the same scenario in SGL (even simultaneously its different parts), can be executed in different styles (like live, virtual and constructive in traditional terminology) by using special context-setting operational modes, which can provide deepest possible and runtime changeable integration of distributed simulation with live control. Section 7 concludes the paper, sharing plans of the further technology

***Address for Correspondence:** Peter Simon Sapaty, Institute of Mathematical Machines and Systems, National Academy of Sciences, Glushkova Ave 42, 03187 Kiev, Ukraine, Tel: +380-44-5265023, E-mail: peter.sapaty@gmail.com*

Received 08 August 2020; **Accepted** 07 October 2020; **Published** 14 October 2020

development and planned new publications. It again stresses fundamental difference of semantics-based SGT with traditional DIS and HLA, which are trying to standardize and absolutize communication protocols and actually leading away from holistic and intelligent solutions for advanced simulation and management systems.

Spatial grasp technology basics

General SGT Idea: Within Spatial Grasp Technology (SGT), a high-level scenario for any task to be performed in a distributed world is represented as an *active self-evolving pattern* rather than traditional program, sequential or parallel. This pattern, written in a high-level Spatial Grasp Language (SGL) and expressing top semantics of the problem to be solved, can start from any world point. It then spatially propagates, replicates, modifies, covers and matches the distributed world in parallel wavelike mode, while echoing the reached control states and data found or obtained for making decisions at higher levels and further space navigation. This inherently parallel and fully distributed spatial process is very symbolically shown in Figure 1.

Many spatial processes in SGL can start any time and in any places, cooperating or competing with each other, depending on applications. The self-spreading & self-matching SGL patterns-scenarios can create *knowledge infrastructures* arbitrarily distributed between system components which may cover any regions, the whole world including, as in Figure 2.

The created infrastructures, which may remain active any time, can effectively support or express distributed databases, advanced command and control, situation awareness, autonomous and collective decisions, as well as any existing or hypothetical computational and or control models.

Spatial Grasp Language (SGL): General SGL organization is as follows, where syntactic categories are shown in italics, vertical bar separates alternatives, parts in braces indicate zero or more repetitions with a delimiter at the right if multiple, and constructs in brackets are optional:

grasp *constant* | *variable* | [*rule*] [({ *grasp*,})]

From this definition, an SGL scenario, called *grasp*, supposedly applied in some point of the distributed space, can just be a *constant* directly providing the result to be associated with this point. It can be a variable whose content, assigned to it previously when staying in this or (remotely) in other space point (as variables may have non-local meaning and coverage), provides the result in the application point too. It can also be a rule (expressing certain action, control, description or context) optionally accompanied with operands separated by comma (if multiple) and embraced in parentheses. These operands can be of any nature and complexity (including arbitrary scenarios themselves) and defined recursively as *grasp*, i.e., can be constants, variables or any rules with operands (i.e., as grasps again), and so on.

Rules, starting in some world point, can organize navigation of the world sequentially, in parallel or any combinations thereof. They can result in staying in the same application point or can cause movement to other world points with obtained results to be left there, as in the rule's final points. Such results can also be collected, processed, and returned to the rule's starting point, the latter serving as the final one on this rule. The final world points reached after the rule invocation can themselves become starting ones for other rules. The rules, due to recursive language organization, can form arbitrary operational and control infrastructures expressing any sequential, parallel, and hierarchical, centralized, localized, mixed and up to fully decentralized and distributed algorithms. These algorithms, called *spatial*, can effectively operate in, with, under, in between, over, and instead of (as for simulation) large, dynamic, and heterogeneous spaces, which can be physical, virtual, management, command and control, or combined.

SGL full syntax description, as of its latest version, is as follows, with the words in Courier New font being direct language symbols (boldfaced braces including).

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

Constant → *information* | *matter* | *custom* | *special* | *grasp*

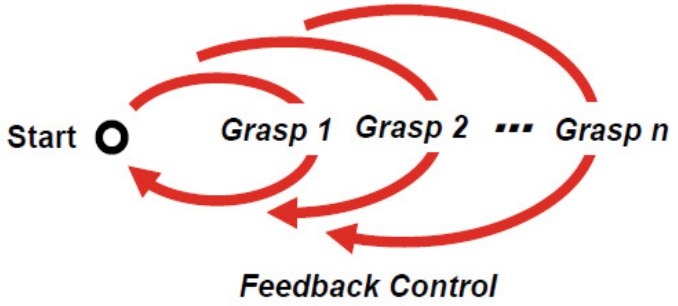

Figure 1. Controlled navigation & matching & grasping of distributed spaces.

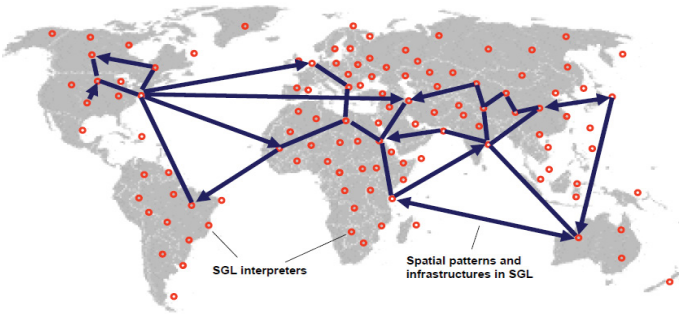

Figure 2. Spreading spatial patterns and creation of distributed infrastructures.

Information → *string* | *scenario* | *number*

String → ' {*character*}'

Scenario → **{** {*character*}**}**

Number → [*sign*] {*digit*} [. {*digit*} [e [*sign*] {*digit*}]]

Matter → " {*character*}"

Special → thru | done | fail | fatal | infinite | nil | any | all | other | all other | current | passed | existing | neighbors | direct | forward | backward | synchronous | asynchronous | virtual | physical | executive | engaged | vacant | first come | unique | usual | real | simulate

Variable → *global* | *heritable* | *frontal* | *nodal* | *environmental*

Global → G {*alphameric*}

Heritable → H {*alphameric*}

Frontal → F {*alphameric*}

Nodal → N {*alphameric*}

Environmental → Type | Identity | Name | Content | Address | Point | Qualities | Where | Back | Previous | Predecessor | Doer | Resources | Link | Direction | When | Time | State | Value | Identity | In | Out | Status | Mode | Color

Rule → *type* | *usage* | *movement* | *creation* | *echoing* | *verification* | *assignment* | *advancement* | *branching* | *transference* | *exchange* | *timing* | *qualifying* | *grasp*

Type → global | heritable | frontal | nodal | environmental | matter | number | string | scenario | constant | custom

Usage → address | coordinate | content | index | time | speed | name | place | center | range | doer | node | link | unit

Movement → hop | hop first | hop forth | move | shift | pass | return | follow

Creation → create | form | linkup | delete | unlink

Echoing → state | rake | order | unit | unique | sum | count | first | last | min | max | random | average | sort-up | sort down | reverse | element | position | from to | add | subtract | multiply | divide | degree | separate | unite | attach | append | common | withdraw | increment | decrement | access | invert | apply | location

Verification → equal | non equal | less | less or equal | more | more or equal | bigger | smaller | heavier | lighter | longer | shorter | empty | nonempty | belong | not belong | intersect | not intersect | yes | no

Assignment → assign | assign peers | associate

Advancement → advance | slide | repeat | align | fringe

Branching → branch | sequence | parallel | if | or | and | choose | quickest | cycle | loop | sling | whirl | split

Transference → run | call

Exchange → input | output | send | receive | emit | get

Timing → sleep | allowed

Qualification → contain | release | free | blind | quit | abort | stay | lift | seize

SGL Interpreter: The SGL interpreter main components and its general organization are shown in Figure 3. The interpreter consists of a number of specialized functional processors (shown by rectangles) working with and sharing specific data structures. These include: Communication Processor, Control Processor, Navigation Processor, Parser, different Operation Processors, and special (external & internal) World Access Unit directly manageable from SGL. Main data structures (also referred to as stores) with 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, Global Variables, Incoming Queue, and Outgoing Queue. SGL interpretation network generally serves multiple scenarios or their parallel branches simultaneously navigating the distributed world, which can cooperate or compete with each other.

Each interpreter can support and process multiple SGL scenario code which happens to be in its responsibility at different moments of time. More details on SGT, SGL, its implementation and investigated and tested applications can be found elsewhere, including in [44-48]. Implanted into any distributed systems and integrated with them, the interpretation network (having potentially millions to billions of communicating interpreter copies) allows us to form spatial world computer with practically unlimited power for simulation and management of the whole mankind.

Pure world types and their management

Dealing with physical world: Working with pure physical world in SGL is just staying in certain physical locations or P points by given coordinates and moving into new locations by coordinates or their shifts from the previous locations, sequentially or in parallel, the latter becoming another P points. Such points (rather than nodes) identified by their physical coordinates and having no names may access certain local physical parameters (like an established standard list of them) in the respected world locations. P points cannot be visible from other points or nodes in SGL even if are located within the defined and investigated physical region, and cannot be re-entered and shared from other points. After all SGL-defined activities terminate in P points, they (with all temporary data accumulated) disappear from SGT view.

How these P points are reached and how they access related world parameters depends on details of SGT implementation. This, for example, may need physical movement of physical equipment to these locations and measure environmental parameters there (incl. performing certain physical operations), and then terminate or move to other locations. Or just access already existing systems and databases which may provide suitable answers on environmental details in these locations (if such indirect access is not critical for the result, also for obtaining latest, runtime, data in these locations). Let us consider some details on how SGT may deal with pure physical world.

Elementary movement into defined physical locations

Some examples are shown in Figure 4 with the following explanations in SGL.

a) Single move into location with X_Y coordinates (Figure 4, a). move (*X_Y*)

b) Moving into a physical location and then to another one by the coordinate shift given (Figure 4, b). move (*X_Y*); move (WHERE + *dx*) or move (*X_Y*); shift (*dx*)

c) Another shift in space (Figure 4, c). move (*X_Y*); shift (*dx*); shift (*dx*).

Repeated movement

a) Unlimited movement repetition (Figure 5). move (*X_Y*); repeat (shift (*dx*)).

b) Repeated movement with a number of jumps allowed (here 5). move (X_Y); repeat_5 (shift (dx)).

c) Movement repetition limited by a threshold distance to destination Xd_Yd. move (X_Y); repeat (shift (dx_dy); distance (WHERE, Xd_Yd) > threshold).

d) Randomized movement with a given repetition number (Figure 6). move (*X_Y*); repeat_100 (shift (random (*dx_dy*))).

Movement to multiple grid positions

a) Sequential horizontal-vertical space coverage (Figure 7). move (X_Y); repeat_50 (repeat_100 (shift (dx)); shift (dy); repeat_100 (shift (-dx)); shift (dy))

b) Sequential spiral expansion coverage (Figure 8). move (X_Y); frontal (Horizontal, Vertical); repeat_100 (Horizontal+=1; repeat_Horizontal (shift (dx)); Vertical+=1; repeat_Vertical (shift (-dy)); Horizontal+=1; repeat_Horizontal (shift (-dx)); Vertical+=1; repeat_Vertical (shift (dy)))

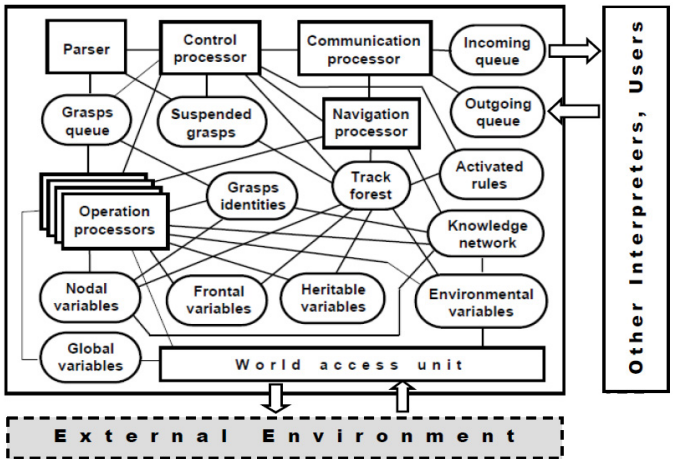

Figure 3. SGL interpreter main components and their interactions.

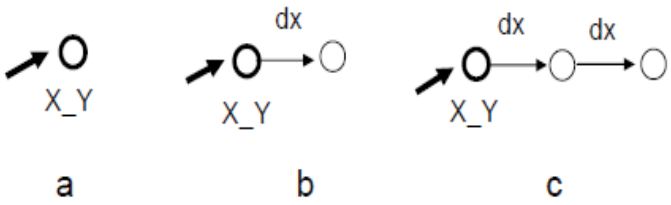

Figure 4. Elementary physical movements.

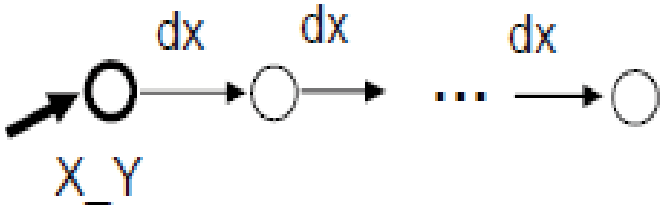

Figure 5. Repeated unlimited movement.

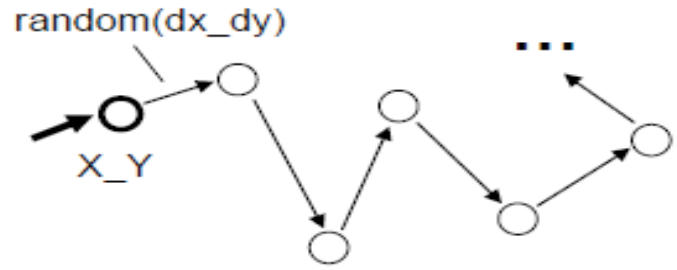

Figure 6. Randomize repetitive movement.

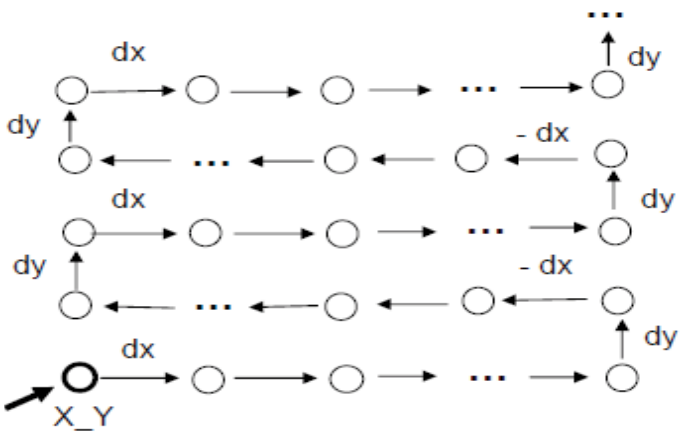

Figure 7. Sequential horizontal-vertical coverage.

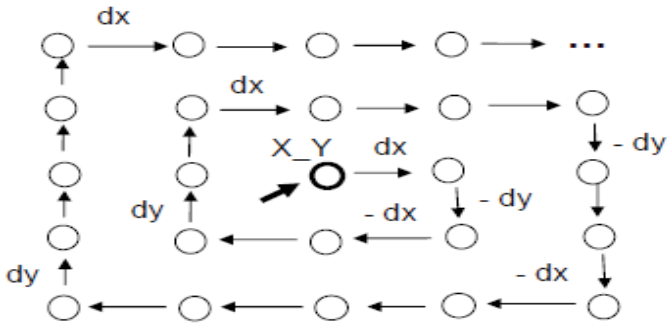

Figure 8. Sequential spiral coverage.

c) Combination of parallel and sequential coverage on different coordinates (Figure 9). nodal (Xstart = …, Ystart = …, Yfinal = …, Dy = …); frontal (Dx = …, Xfinal = …); split (fromto (Ystart, Yfinal, Dy)); move (Xstart_VALUE); repeat (shift (Dx); WHERE [1] < Xfinal)

d) Parallel movement directly to all positions in the grid (Figure 10). nodal (Xstart = …, Xfinal = …, Dx = …, Ystart = …, Yfinal = …, Dy = …); frontal (Xcurrent); split (fromto (Xstart, Xfinal, Dx)); Xcurrent=VALUE; split (fromto (Ystart, Yfinal, Dy)); move (Xcurrent_VALUE)

Application examples

a) By extending the latest scenario, finding position with maximum temperature in the region searched.

nodal (Xstart = …, Xfinal = …, Dx = …, Ystart = …, Yfinal = …, Dy = …);

frontal (Xcurrent);

print (maximum (split (fromto (Xstart, Xfinal, Dx)); Xcurrent = VALUE;

split (fromto (Ystart, Yfinal, Dy)); move (Xcurrent_VALUE);

append (QUALITIES (temperature), WHERE)))

Possible output in the scenario starting position: 42C, Xi_Yi

b) Following the lake's shoreline (Figure 11).

frontal (Start = …, Type = shoreline, Direction = left,

Depth = …, Close = threshold);

move (Start);

repeat (move_follow (Direction, Type, Depth);

if (distance (WHERE, Start) < Close, done))

c) Following the lake's shoreline with output of coordinates of all passed points and also the measured full length of the shoreline.

frontal (Start = …, Type = shoreline, Direction = left,

Depth = …, Close = threshold, Coordinates, Length);

move (Start);

repeat (append (Coordinates, WHERE); follow (Direction, Type, Depth);

Length += distance (Coordinates [last], WHERE);

if (distance (WHERE, Start) < Close,

done (output ('COORD: ' && Coordinates, ' | LENGTH:' && Length))))

Possible output: COORD: (x1_y1, …, xn_yn) | LENGTH: 265 km.

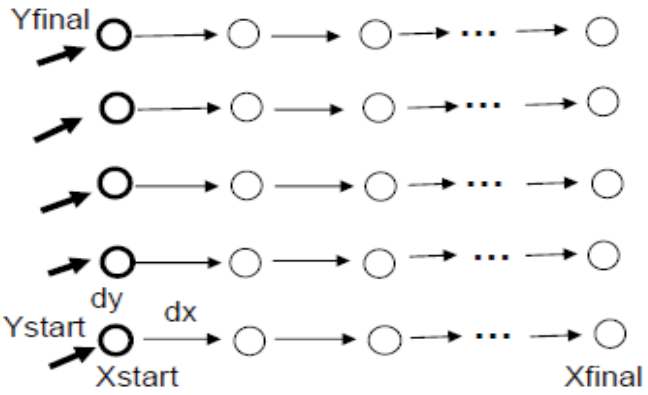

Figure 9. Parallel-sequential total coverage.

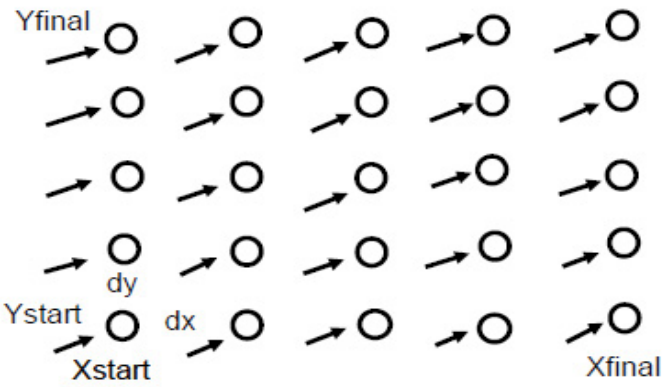

Figure 10. Fully parallel total coverage.

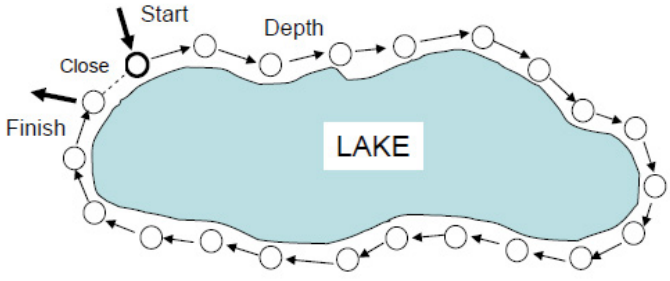

Figure 11. Following lake's shoreline.

Dealing with virtual world

Virtual or V nodes are having names, contents (generally a list), and network addresses. V nodes can be created in SGL and connected with other nodes by named virtual links expressing semantic relations (with orientations if needed). It is possible to create any such virtual networks in SGL and navigate them sequentially or in parallel, also organize parallel and distributed matching of them by graph-like spatial patterns. Any changes to node names and contents, also orientations and names of links between them are possible in SGL, but node addresses are formed automatically and internally by the distributed interpreter on implementation layer, and can only be copied, remembered, and used subsequently to enter these nodes directly.

Virtual nodes are persistent and after creation continue their existence regardless of presence or absence of processes and additional information in them. They may be re-entered by any other and any number of SGL processes which can share and change them (i.e. names and contents, also temporary variables linked to them). Virtual nodes can be removed only explicitly by certain SGL rules (or by just assigning empty value to their names), and only if there are no other processes associated with them at this moment of time. Some examples of dealing with the virtual world follow.

Elementary creation-navigation operations

Some elementary virtual world operations are shown in Figure 12 and explained below.

a) Creating single virtual node with proper name (Figure 12, a). create (direct, A)

b) Extending the single-node network using named link (Figure 12, b).

hop (direct, A); create (link (r1), node (B))

or just:

hop (A); create (r1, B)

c) Another network extension, (Figure 12, c).

hop (A); hop (r1, B); create (r2, C)

d) Combined network navigation with parallel extension (Figure 13, a).

hop (B); hop (r2, C), create (r3, D)

e) Limited repeated network creation (Figure 13, b).

frontal (Name = 1, Nfinal = 30); create (direct, Name);

repeat (Name += 1; create (r, Name); Name < Nfinal)

Or more compact for this particular scenario:

frontal (Name = 1); create (1); repeat_29 (create (r, Name += 1))

Creating tree-like network (Figure 14)

create (A);

(create (r1, B); create (r3, D), create (r4, E)),

(create (r2, C); create (r5, F), create (r6, G))

Or with compact coding and rule create used in a context-like mode:

create (@#A; (r1#B; r3#D, r4#E), (r2#C; r5#F, r6#G))

Arbitrary network creation

Creating arbitrary network (Figure 15, left) based on its depth-first spanning tree (Figure 15, right).

Network creation in SGL will be as follows:

frontal (F1, F2);

create (direct, A); F1 = ADDRESS;

create (r2, C); create (r5, E);

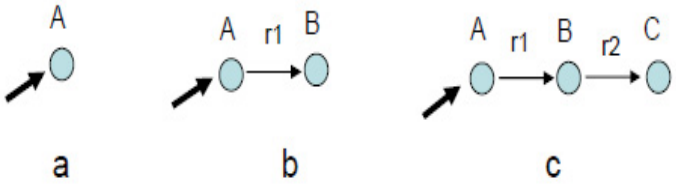

Figure 12. Elementary operations.

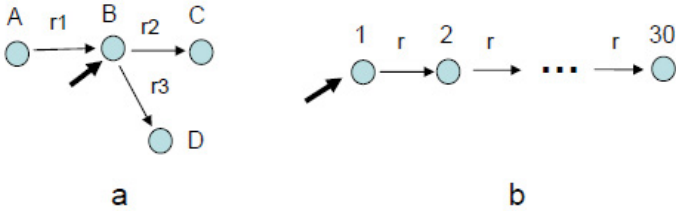

Figure 13. Other virtual network examples.

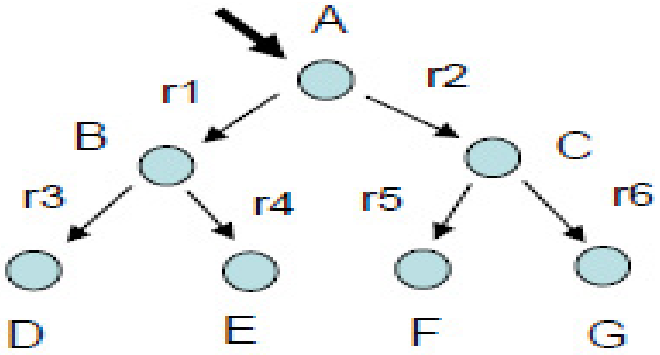

Figure 14. Tree-like virtual network example.

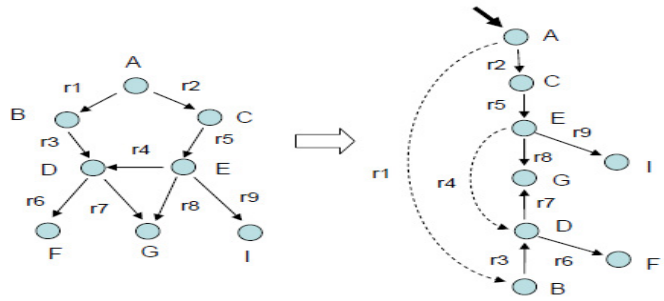

Figure 15. Virtual network and its depth-first tree coverage.

F2 = ADDRESS; create (r9, I), (create (r8, G); create (-r7, D);

create (r6, F), link (-r4, F2), (create (-r3, B); link (-r1, F1)))

Using compact coding version

create (@#A; r2#C; r5#E; r9#I, (+r8#G; -r7#D; +r6#F, -r4##E, (-r3#B; -r1##A)))

Network pattern-matching

Finding node names (of the network of Figure 15) in a graph pattern matching, with pattern structure and its variables X1-X6 shown in Figure 16a, and their correspondence to network nodes in Figure 16b.

This matching, sown in SGL below, can use a linear search template (Figure 17) based on a path through all nodes and links of the pattern of Figure 16a.

frontal (X); hop (direct, all); X [1]=NAME;

+any#any; X [3]=NAME; +any#any; X [5]=NAME;

+any#any; X [6]=NAME; -any#any; X [4]=NAME;

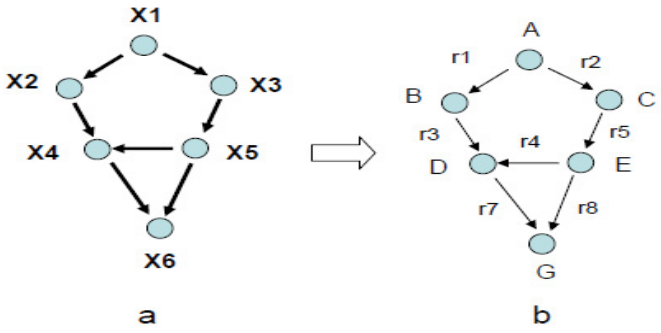

Figure 16. Graph pattern with correspondence of variables to network nodes.

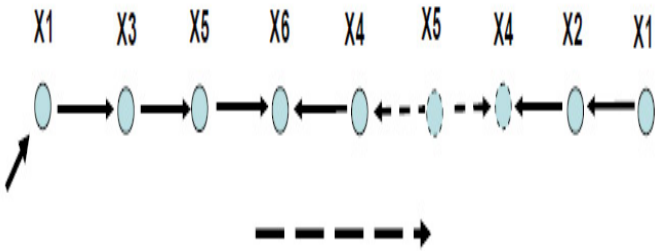

Figure 17. Path-based search template for the pattern of Figure 16a.

-any#X [5]; +any#X [4]; -any#any; X [2]=NAME;

-any#X [1]; output (X)

Using compact code version:

frontal (X);@#;X [1]=NAME;+#;X [3]=NAME;+#;X [5]=NAME;

+#;X [6]=NAME;-#; X [4]=NAME;-#X [5];+#X [4];-#;X [2]=NAME;

-#X [1];output (X)

Result: A, B, C, D, E, G

Similarly can be done for collection of all link names for this pattern matching.

More on complex network creation

a) Creating network of Figure 16b by a similar to Figure 17 linear template shown in Figure 18.

create (direct, A);

create (+r2, C); create (+r5, E); create (+r8, G); create (-r7, D);

hop (-r4, E); hop (+r4, D); create (-r3, B); create (-r1, A)

Or with denser coding, using rule create in a context mode:

create (@#A; +r2#C; +r5#E; +r8#G; -r7#D); -r4#E; +r4#D; create (-r3#B; -r1#A)

b) Tree-based creation of network of Figure 16b using sequencing of branches to avoid competition during nodes creation (Figure 19).

create (direct, A);

sequence (

(create (+r2, C); create (+r5, E); create (+r8, G)),

(create (+r1, B); create (+r3, D); linkup (-r4, E), linkup (+r7, G)))

Or with more compact coding

create (@#A);

sequence ((create (+r2#C; +r5#E; +r8#G), (create (+r1#B; +r3#D; -r4##E, +r7##G))

Managing executive worlds

Executive or E nodes, may be represented by people, robots, sensors, computers, any electronic and mechanical machinery, the whole organizations, any web-based units including, which can be addressed, entered, and tasked. E-nodes can have personal names-identities, allowing them to be found and addressed on a request from SGT via existing channels (like voice, telephone, paper, post, internet, etc.) directly or with possible help of external catalogues, databases, etc.

SGL allows us to move to and between them via the channels chosen, sequentially or in parallel, give them executive orders, also establish any command and control infrastructures between them. Executive nodes are supposed to exist beyond SGT and cannot be created or changed in SGL explicitly. But they, at least some, may be formed, created, registered and changed indirectly on a request from SGT to other systems and authorities. E-nodes can be directly accessed from any other nodes by their names-identities. Examples of dealing with E nodes are shown below.

Elementary operations

Some elementary E node operations are shown in Figure 20 and explained in SGL as follows.

a) Entering executive node A (Figure 20, a).

pass (A)

b) Entering executive nodes first A then B (Figure 20, b).

pass (A); pass (B)

c) Entering A then B and returning to A (Figure 20, c).

pass (A); pass (B); return (A)

Managing executive hierarchy

a) Spreading top-down via chosen executive hierarchy (Figure 21).

pass (A); (pass (B); pass (D), pass (E)), (pass (c); pass (F), pass (G))

b) Forming and fixing executive hierarchy (Figure 22).

nodal (Up, Down);

pass (A); Down = (B, C);

(pass (B); Up = A; Down = (D, E); (pass (D); Up = B), (pass (E); Up = B)),

(pass (c); Up = A; Down = (F, G); (pass (F); Up = C), (pass (G); Up = C))

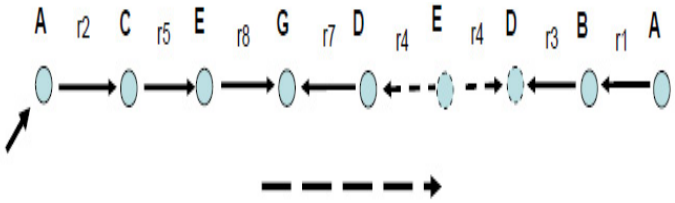

Figure 18. Linear network-creation template.

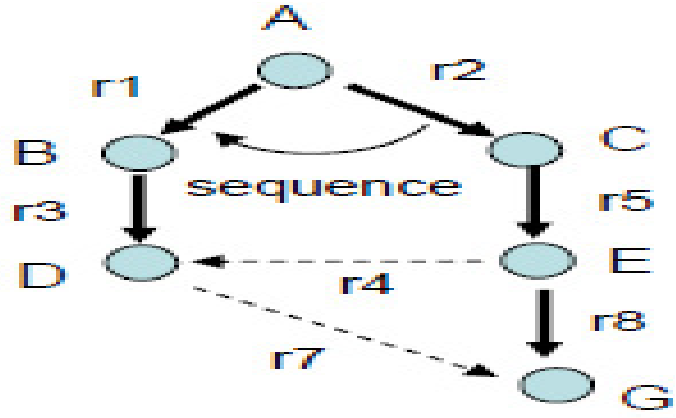

Figure 19. Tree-based network creation with sequencing of branches.

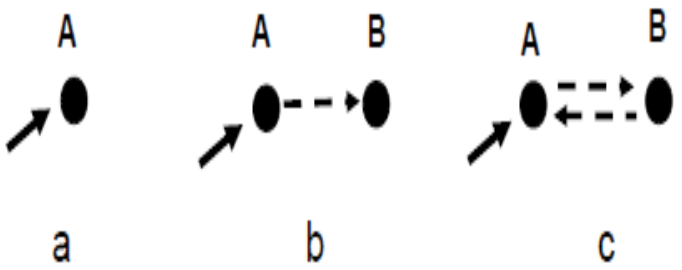

Figure 20. Some elementary E node operations.

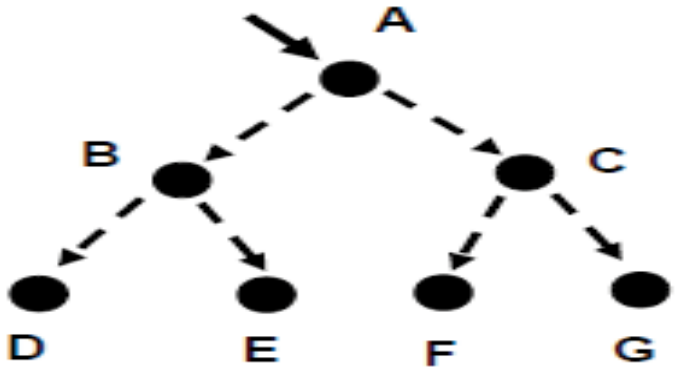

Figure 21. Top-down movement via executive hierarchy.

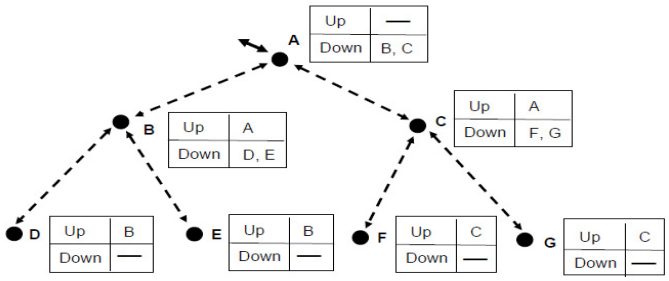

Figure 22. Fixing executive hierarchy.

c) Staying at the top and issuing modifying order downward the whole hierarchy.

frontal (Command = …);

repeat (pass (Down); execute_update (Command))

d) Staying at the bottom level and issuing modifying reply upward the whole hierarchy.

frontal (Reply = …);

repeat (return (Up); inform_update (Reply))

e) Endlessly combining top down and bottom up control cycle

pass (A); frontal (Command = …); nodal (Beneath);

release (repeat (

repeat (pass (Down); Beneath = count (Down); execute_update (Command));

frontal (Reply = …);

repeat (return (Up); inform_update (Reply); decrement (Beneath) == 1)))

Managing ring infrastructures

a) Following ring command route (Figure 23).

frontal (Ring= (A, B, C, D, E, F, G), Start);

Start = Ring (first);

repeat (

Current=withdraw (Ring); append (Ring, Current); pass (Current);

nonequal (Start, Ring [first]))

b) Forming and fixing ring command structure (Figure 24).

frontal (Ring = (A, B, C, D, E, F, G), Start, Current);

nodal (Left, Right);

Start = Ring (first);

repeat (

Current = withdraw (Ring); pass (Current);

Left = Ring [last]; Right = Ring [first];

append (Ring, Current); nonequal (Start, Ring [first]))

c) Endlessly circulating control via the ring infrastructure, starting in any ring unit.

pass (any); frontal (Command = …);

release (repeat (pass (Right); execute_update (Command)))

d) Endlessly circulating awareness starting in any ring unit.

pass (any); frontal (Aware = …);

release (repeat (return (Left); inform_update (Aware)))

Combining vertical and horizontal control infrastructures

Any combinations of hierarchical and horizontal command and control organizations can be easily managed in SGL, by integrating hierarchical and ring scenarios shown above (Figure 25).

Forming and fixing this combined infrastructure in SGL will be as follows:

nodal (Up, Down, Left, Right);

frontal (Ring, current, Start);

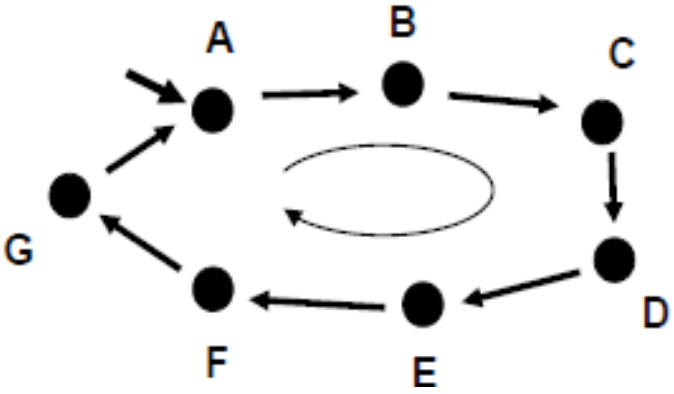

Figure 23. Following ring command route.

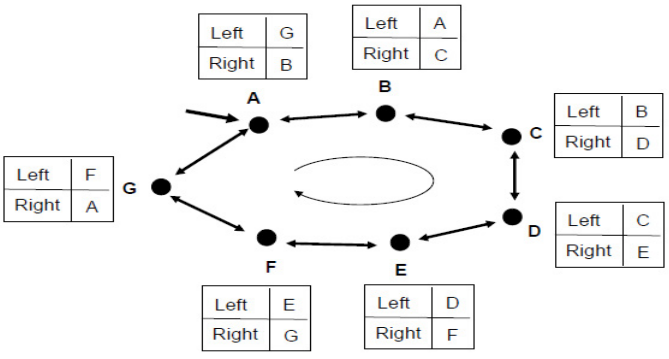

Figure 24. Fixing ring command structure.

Ring = reorder (

pass (A); Down = (B, C);

(pass (B); Up = A; Down = (D, E); (pass (D); Up = B), (pass (E); Up = B)),

(pass (c); Up = A; Down = (F, G); (pass (F); Up = C), (pass (G); Up = C));

identity);

Start = Ring [first]);

repeat (

Current = withdraw (Ring); pass (Current);

Left = Ring [last]; Right = Ring [first];

append (Ring, Current); nonequal (Start, Ring [first]))

Or shorter for this particular case, as ring nodes are only four, and they can be processed explicitly within the hierarchical stage.

nodal (Up, Down, Left, Right);

frontal (Ring, current, Start);

pass (A); Down = (B, C);

(pass (B); Up = A; Down = (D, E); (pass (D); Up = B; Left = E; Right = G),

(pass (E); Up = B; Left = F; Right = D)),

(pass (c); Up = A; Down = (F, G); (pass (F); Up = C; Left = G; Right = E),

(pass (G); Up = C; Left = D; Right = F))

The resultant combined command and control infrastructure will be recorded and fixed in different E nodes as summarized in Figure 26.

Combined world types

Physical-Virtual: Allows physical points reached to have virtual names, contents and addresses as in a pure virtual world, also have virtual links or relations between them, thus becoming visible and persistent PV nodes. And from the other side, such integration allows virtual nodes to be associated with locations in the physical world and have access to the local world parameters similar to pure physical points. Also such combined PV nodes are persistent unlike pure P points, can be shared by different SGL scenarios and continue to exist unless removed explicitly by SGL rules. PV nodes can be created by first moving to the needed physical location by given coordinates, thus getting P points, and then assigning them virtual dimension with the name and contents. Or by first creating V node and then associating it with the given physical coordinates providing access to related local world parameters.

This integration can be also done from the very beginning, by creating PV node in a single breath and supplying it with virtual name & contents (the latter if needed) and physical coordinates, and immediately moving into the given physical location. PV nodes are visible and reachable in physical space from P and other PV nodes (say, by outlining a physical region where they may be located rather than their exact coordinates due to limited coordinate precision), also from V and PV nodes by their virtual names and semantic links between them.

PV nodes, preserving their identities and links with other V and PV nodes, also with all accumulated information and processes in them, can migrate in physical world by setting new full coordinates or shifts from current positions in space, rather than creating new PV nodes in the reached destinations as in case of pure P points. Similar to P points, PV nodes associated with certain physical locations can also have indirect access to them via other systems, if this does not impede the needed problem solutions.

We have chosen the logo of PV nodes as shown in Figure 27, a, with examples of their usage explained below.

Forming PV nodes (Figure 27, b)

a) Forming nodes in a single breath.

form (TYPE = physical_virtual; WHERE = X_Y; NAME = A; CONTENT = ...)

Or shorter, without explicit usage of environmental variable TYPE as redundant:

form (WHERE = X_Y; NAME = A; CONTENT = ...)

b) Forming PV nodes stepwise.

Starting from physical world:

move (X_Y); NAME = A; CONTENT = ...

Or starting from virtual world:

create (A); CONTENT = ...; WHERE = X_Y

Organizing PV nodes movement in space: PV nodes have unique names and can exist indefinitely unless removed explicitly; they can also move and shift in space while preserving identities and links with other nodes. A repeated shift in physical space of a combined node A is shown in Figure 27, c, also below in SGL (limited to N repetitions).

hop (A); repeat_N (shift (dx_dy))

Using PV nodes visibility in physical and virtual spaces: Such nodes can be visible in and from both physical and virtual spaces by their current physical locations, also by names and via relations between nodes (the latter as in a pure virtual space), (Figure 28).

a) Visibility examples (Figure 28).

– Replying exact coordinates of nodes inside the given physical area:

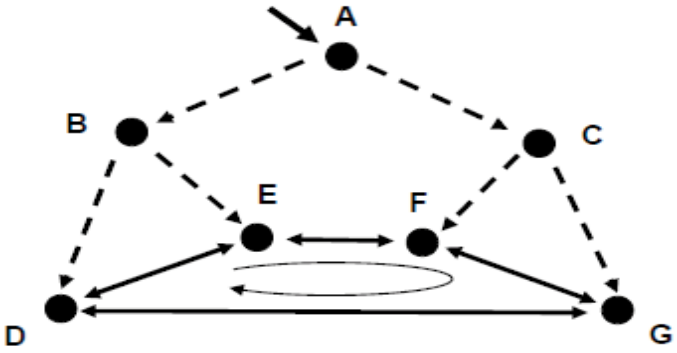

Figure 25. Combining hierarchical and horizontal executive infrastructures.

Node	A	B	C	D	E	F	G
Up	—	A	A	B	B	C	C
Down	B, C	D, E	F, G	—	—	—	—
Left	—	—	—	E	F	G	D
Right	—	—	—	G	D	E	F

Figure 26. Summarized hierarchical-horizontal infrastructure data.

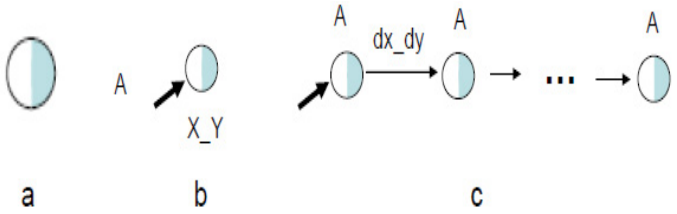

Figure 27. Combined physical-virtual nodes and their elementary networks.

output (hop_nodes (center (X_Y), radius (*radius*)); WHERE)

-- Replying exact coordinates of nodes given by their names in virtual space:

output (hop (B, C, D, E); WHERE)

Result in both cases: xB_yB, xC_yC, xD_yD, xE_yE

-- Replying names of all nodes inside the given physical area:

output (hop_nodes (center (X_Y), radius (*radius*)); NAME)

Or just (by default, as the reached node names are always representing the result of search):

output (hop (center (X_Y), radius (r)))

Result: B, C, D, E

b) Some more visibility examples (Figure 28).

output (hop (A); hop_links (r1, r6); WHERE)

Result: xB_yB, xD_yD

output ((hop (F); hop_link (r3)),

(hop (G); hop_link (r5)); NAME && WHERE)

Result: (C, xC_yC), (E, xE_yE)

Executive-Physical: Combining E nodes with P points as EP nodes (having E node identities as the united node names) will need the related executive units to physically appear in (or move to) the locations corresponding to P points and made them capable of directly accessing local world parameters there. But similar to pure P points, also PV nodes, this access with obtaining needed data and even providing physical impact there could also be in certain cases implicit, indirect, with the possible use of other systems associated with these points or already located there. EP nodes can be accessed directly from any other nodes by their names-identities using existing channels between E nodes, also by naming/outlining a physical region where they may be located, thus being visible in physical world similar to PV nodes.

Will be using the logo shown in Figure 29, a for the united EP nodes, with some examples of their creation and usage following.

Forming EP nodes in a single step

form (TYPE = executive_physical; IDENTITY = E; WHERE = X_Y)

Or shorter, as explicitly using environmental variable TYPE may be optional here:

form (IDENTITY = E; WHERE = X_Y)

Stepwise forming EP nodes

-- Starting from E nodes:

pass (E); WHERE = X_Y

-- Starting from P points:

move (X_Y); IDENTITY = E

Swarm randomized movement of PV nodes with impact of discovered targets (Figure 29,b)

Executives = (E1, E2, …, En); pass (Executives);

repeat (

Shift = random (dx_dy);

if (empty (WHERE + Shift),

(shift (Shift); if (seen (targets), impact (targets))),

sleep (*delaytime*)))

Executive-Virtual: E nodes in combination with V nodes, as EV nodes,

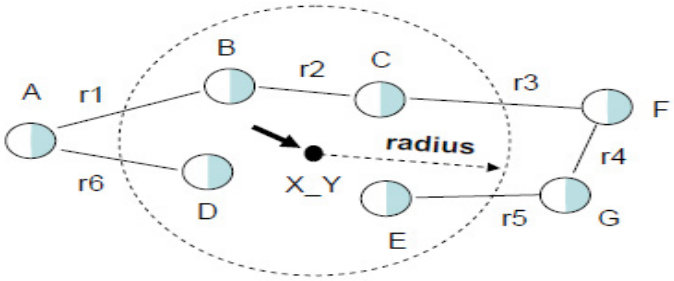

Figure 28. Networked examples for PV nodes.

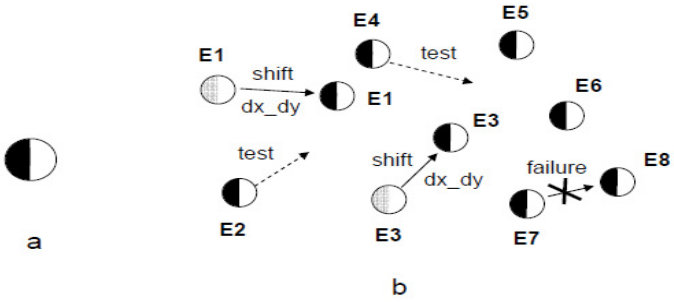

Figure 29. Executive-physical nodes and their usage.

additionally to their E-identities can have virtual names and contents like pure V nodes. Already existing E nodes can be converted to EV nodes by assigning V-status for them with proper names and contents, also automatically obtaining resulting network addresses afterwards. EV nodes may have semantic links with other EV nodes, also with PV and V nodes, via which they can be entered from other V-related nodes or, in turn, access them. EV nodes can be directly contacted / accessed / entered by both their E-identities and assigned V-names. If virtual names not assigned to combined EV nodes, they may be directly accessed by their E identities, which will also be treated as V-type node names, so environmental variables NAME and IDENTITY will be used as the same.

The EV nodes logo is depicted in Figure 30, a, with some examples of their usage following.

Single step EV node forming

form (TYPE = executive_virtual; IDENTITY = E; NAME = A; CONTENT = …)

Or, if IDENTITY used as NAME:

form (TYPE = executive_virtual; IDENTITY = E)

Or, if both IDENTITY and NAME present and different, with using TYPE optional:

form (IDENTITY = E; NAME = A; CONTENT = …)

Stepwise EV node forming

pass (E); NAME = A; CONTENT = …

Or, starting from virtual dimension:

create (A); CONTENT = …; IDENTITY = E

Fixing centralized infrastructure (Figure 30, b)

Boss = E1; Subordinates = (E2, E3, E4, E5, E6, E7, E8);

pass (Boss); linkup (+command, nodes (Subordinates))

Issuing a command order (Figure 30, b)

Boss = E1; pass (Boss); frontal (Order = …);

pass_link (+command); execute (Order)

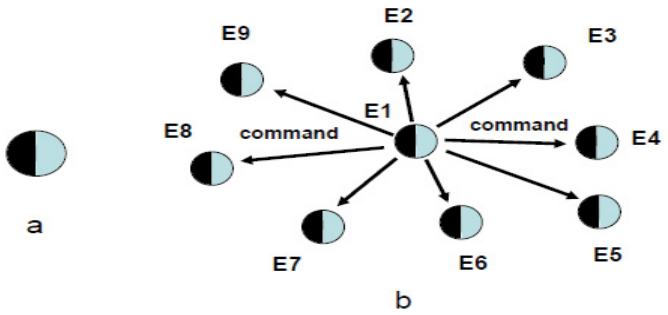

Figure 30. Executive-virtual nodes and their usage.

Or, starting from virtual dimension:

hop_node (E1); frontal (Order = …);

hop_link (+command); execute (Order)

Executive-Physical-Virtual: Executive-physical-virtual or EPV nodes combine features of E nodes with physical locations where they are or have to be located, also with additional virtual names, contents, and resulting network addresses. They will also acquire possibility of creating semantic links with other V, PV, EV, and EPV nodes. EPV nodes may be formed stepwise, starting from E nodes, and then assigning them P-coordinates where they are currently located or by moving into them, if different, as well as V-features. Similarly to EV nodes, they can be directly accessed by both E-identities and V-names.

The EPV nodes logo is depicted in Figure 31, a, with examples of their applications explained below in SGL.

Forming EPV nodes

a) Simultaneously, in a single breath.

form (TYPE = executive_physical_virtual, IDENTITY = E, WHERE = X_Y,

NAME = A, CONTENT = …)

Or with a shortened coding:

form (IDENTITY = E, WHERE = X_Y, NAME = A, CONTENT = …)

b) Stepwise forming of EPV nodes.

-- Starting from executive dimension:

pass (E); WHERE = X_Y; NAME = A; CONTENT = …

-- Staring from physical dimension:

move (X_Y); IDENTITY = E; NAME = A; CONTENT = …

-- Or starting from virtual dimension:

create (A); CONTENT = …; IDENTITY = E; WHERE = X_Y

Fixing hierarchical infrastructure taking into account allowed physical distances between nodes (Figure 31, b)

pass_firstcome (E1); NodeDistance = …;

repeat (linkup_firstcome (+command, distance (NodeDistance)))

Issuing an order via command hierarchy

pass (E1); frontal (Order = …);

repeat (hop_link (+command); execute_update (Order))

Creating vertical-horizontal infrastructure with any nodes available, using threshold physical distance between directly connected nodes (Figure 32)

frontal (NodeDistance = …, Ring);

pass_firstcome (E1);

Ring = adjustorder (

repeat (linkup_firstcome (+vertical, distance (NodeDistance)));

IDENTITY);

hop (Ring [first]); frontal (Next = first);

repeat_linkup (horizontal, Ring [increment (Next)]);

linkup (horizontal, Ring [first])

Node type reductions

Reducing combined node types can be easily done by the following operations in nodes (Figure 33).

- Physical-virtual to physical only (Figure 33, a)

TYPE = physical or NAME = nil

- Physical-virtual to virtual only (Figure 33, b)

TYPE = virtual or WHERE = nil

- Executive-physical to physical only (Figure 33, c)

TYPE = physical or IDENTITY = nil

- Executive-virtual to virtual only (Figure 33, d)

TYPE = virtual or IDENTITY = nil

- Executive-physical-virtual to physical-virtual (Figure 33, e)

TYPE = physical_virtual or IDENTITY = nil

- Executive-physical-virtual to executive-virtual (Figure 33, f)

TYPE = executive_virtual or WHERE = nil

- Executive-physical-virtual to executive-physical (Figure 33, g)

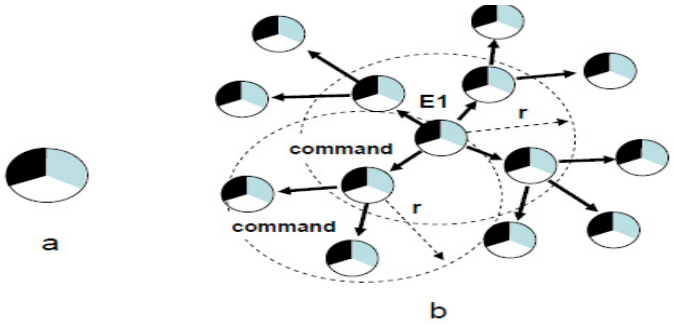

Figure 31. Executive-physical-virtual nodes and their usage.

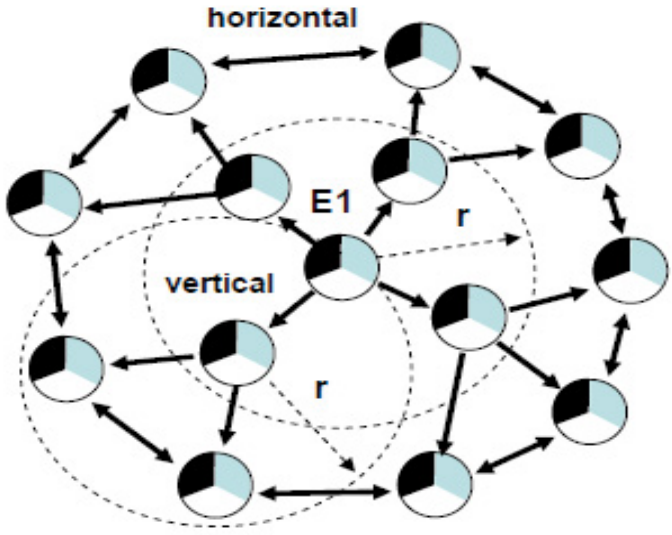

Figure 32. Creating vertical-horizontal infrastructure with EPV nodes.

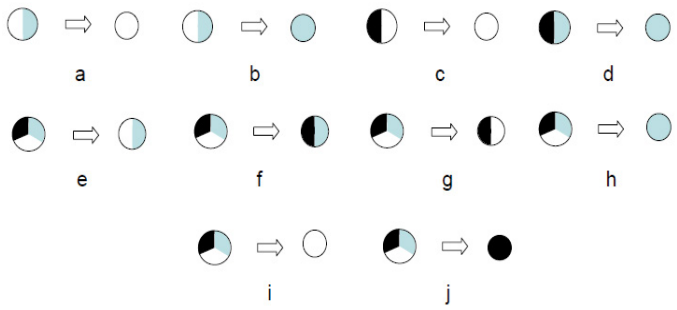

Figure 33. Variants of node type reductions.

TYPE = executive_physical or NAME = nil

- Executive-physical-virtual to virtual only (Figure 33, h)

TYPE = virtual or IDENTITY = nil; WHERE = nil

- Executive-physical-virtual to physical only (Figure 33, i)

TYPE = physical or IDENTITY = nil; NAME = nil

- Executive-physical-virtual to executive only (Figure 33, j)

TYPE = executive or WHERE = nil; NAME = nil

The nodes complete removal while staying in them can be done by assigning nil to WHERE for pure physical nodes, to NAME for virtual nodes, and to IDENTITY for executive nodes, or stepwise for combined nodes similarly to the above mentioned cases. Any nodes can also be immediately deleted by assigning nil to their environmental variable TYPE:

TYPE = nil

Modes: Usual, Real, Simulate

Different modes semantics: These modes can be established in frontal environmental variable MODE, where usual, reflecting all described features above, is the default option. If to set up real in MODE, all P, PV, EP, and EPV nodes will need absolute physical presence (direct physical impact including) in the respected physical locations, not allowing this being done remotely by other systems or using existing databases to obtain/change needed local world parameters. If to set up simulate in MODE, all world access by P, PV, EP and EPV nodes in the subsequent scenario will be interpreted only in a modeling regime, by using information in existing records or databases related to these word points. Also, all executive units will be considered as simulated rather than real ones, and when accessed by their E-identities will be redirected to their V-dimension if have it (i.e. for EV and EPV nodes).

Assigning usual to MODE will restore normal interpretation of the following scenario. By runtime changing the content in MODE in the same scenario (there may be many independent MODE variables spreading in distributed spaces as the scenario may have different and many branches evolving sequentially or in parallel) we can effectively combine real and simulation styles of its execution. By this, the same parts can be changing their style during repeated execution and at any time, thus leading to deep symbiosis of real and simulation mode in solving complex problems, as will be shown in the planned subsequent publications, new books including.

Using fluent symbiotic example: Let us consider the following scenario. There are two types of groups or swarms (Figure 34), one composed from unmanned (say, aerial) units spreading in physical space in a coordinated randomized manner, and the other one being group/swarm of "alien" objects considered as targets (the latter spreading in space in a randomized manner too). For both groups, each unit or object can make next randomly chosen move only if the expected destination location is not occupied.

Each unit (represented as EPV node) has a unique identity or name, while all target objects symbolically having just same identities-names as 'target' (being simplified EPV nodes too). Unmanned units, moving in space, are searching for target objects to a certain depth from their current locations (limited by their sensors), accumulating information on the objects seen and each time trying, if possible, to destroy the nearest registered alien object.

This scenario can be made more advanced (Figure 35), by introducing a certain parallel automatic command and control infrastructure for the unmanned swarm, with some unit appointed as central one (here E7) having direct access to all other units via virtual "infra" links.

Having such central unit, all other units may regularly supply it with all locally seen targets with centralized accumulation of their summary. This summary may then be broadcast back to all units and enhance their awareness of what is being seen by the whole swarm, also providing opportunity of attacking targets picked up by other units (and even doing this cooperatively).

The whole scenario containing all mentioned features is provided in SGL below, where parts related to the unmanned swarm with the mentioned additional centralized service are shown in bold, with the rest relating to the target swarm, also top management of the united scenario. The following SGL scenario is presented in a maximum parallel and fully distributed mode and may be effectively used for organizing and management of real hardware units and objects with their physical cooperation and fighting each other, or for distributed simulation in distributed computer networks, as well as any combination thereof.

nodal (Units, Targets, Shift, Objects; frontal (Seen, Summary);

stay (sequence (

(Units = ((E1, X1_Y1), (E2, X2_Y2), …, (Em, Xm_Ym));

split (Units); form (IDENTITY = NAME = VALUE [1], WHERE = VALUE [2])),

(hop (E7); linkup (infra, all_other_nodes)),

(Targets = (x1_y1, x2_y2, …, xn_yn);

Split (Targets); form (IDENTITY = NAME = 'target'; WHERE = VALUE))));

parallel (

(hop (all_units);

repeat (

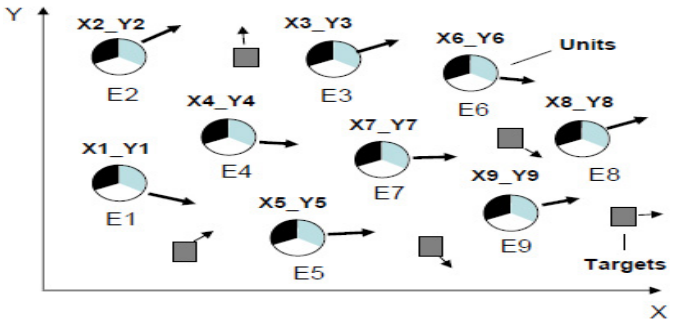

Figure 34. Two types of swarms propagating in space.

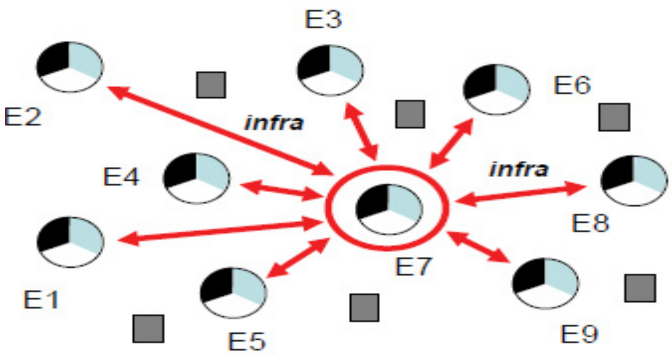

Figure 35. Using additional centralized infrastructure.

Shift = random (*d1x_d1y*);

if ((empty (WHERE + Shift), shift (Shift));

Seen = hop (threshold_distance); NAME = 'target');

stay (hop_link (infra), stay); append_new (Objects, Seen));

remove (select_closest (Objects)); sleep (*delay1*))),

(hop (E7); Summary = Objects;

repeat (

stay (hop (infra); append_new (Ojects, Summary)); sleep (*delay2*))),

(hop (all_targets);

repeat (

Shift = random (*d2x_d2y*);

if (empty (WHERE + Shift), shift (Shift), sleep (*delay3*))))

We did not explicitly use the frontal environmental variable MODE assuming that on default this is equivalent to its meaning as usual, so this scenario may be exploited with a good deal of flexibility like combining live operations and simulation models, which may relate to both live and virtual modeling concepts [28,29]. If to set up real in MODE before the start of the scenario, the existence of all units and objects and their behavior should completely and absolutely correspond to the physical and live organizations. If to set up simulate in MODE at the beginning, then all units and their interactions will correspond to the complete modeling in parallel, distributed or centralized representation, depending on the implementations, which may fully relate to the *constructive* simulation [28,29].

Taking into account the frontal, mobile nature of MODE, by using it in different parts of the same scenario will be possible to organize their simultaneous execution in different modes, from live to virtual to fully simulated, even with runtime changing modes for the same parts during their different repetitions. Thus providing very flexible combination and integration, actually symbiosis, of live and simulated implementation of complex systems.

More information on traditional meaning of live, virtual, and constructive may be found in [28], where *live* is a simulation involving real people operating real systems, virtual is a simulation involving real people operating simulated systems, and constructive involves simulated people operating simulated systems. These types usually have different implementation costs, where live has relatively high one since it is very human resource/materiel intensive and not particularly repeatable, virtual has relatively medium cost with less human resource/materiel intensive, and constructive has relatively low cost as being the least human resource/materiel intensive [55-62].

Discussion and Conclusion

We have described basics of representation and management of different worlds, like continuous physical world, discrete and networked virtual world, and executive world consisting of active human and technical resources. Different possibilities of merging of these worlds (even symbiosis) with each other were demonstrated, which may bring clear benefits and advantages for their simulation, management and control, from local to global. And all operations on different worlds and their combinations were expressed in a simple recursive formalism supported by high-level Spatial Grasp Language (SGL).

This formalism, expressing top semantics of the overall simulation and control, allows us to directly see, stay, comprehend, move and operate in distributed environments while shifting traditional system organization and management routines, used to be programmed explicitly, to effective automatic language interpretation. SGL offers a much higher level of expressing distributed simulation and control systems, considering traditional DIS and HLA organizations just as lower-level implementation systems only. The author raised this question many years ago, but this direction continues to develop using huge resources and believably not in the best way.

SGT continues its development in different areas, including advanced mosaic-type operations in distributed systems, in trying to understand and simulate such extremely complex features as awareness and consciousness, in providing philosophical and technological support of space conquest and advanced terrestrial and celestial missions, and many others. New conference and journal papers as well as two books are being prepared, new patent on the technology too.

The latest SGL version can be implemented even within standard university environments, similar to its previous versions in different countries (named WAVE at the past) under the author's supervision. The technology can be installed in numerous copies worldwide and deeply integrated with any other systems, actually acquiring unlimited power for simulation and management of the whole world.

References

1. https://en.wikipedia.org/wiki/World-systems_theory.
2. https://www.researchgate.net/publication/328333429_Understanding_Dynamics_of_Modern_World.
3. Burke, Anthony, and Rita Parker. "Global Insecurity: Futures of Global Chaos and Governance." *Springer*, 2017.
4. https://www.un.org/development/desa/dspd/wp-content/uploads/sites/22/2020/02/World-Social-Report2020-FullReport.pdf.
5. https://en.wikipedia.org/wiki/List_of_ongoing_armed_conflicts.
6. https://www.acclimatise.uk.com/2020/01/20/environmental-threats-dominate-2020-global-risks-report-for-the-first-time-in-history/.
7. https://www.weforum.org/agenda/2020/01/what-s-missing-from-the-2020-global-risks-report/.
8. https://www.climatechangenews.com/2020/01/15/climate-change-tops-risks-for-world-in-2020-davos-report/.
9. Lai, Allen Y. "Organizational collaborative capacity in fighting pandemic crises: a literature review from the public management perspective." *Asia Pacific J Pub Health* 24 (2012): 7-20.
10. Hamal, Pawan Kumar, Ganesh Dangal, Pradip Gyanwali, and Anjani Kumar Jha. "Let Us Fight Together against COVID-19 Pandemic." *J Nepal Health Res Coun* 18 (2020): I.
11. http://www.sympoetic.net/Simulation_Models/GSS.html.
12. http://www.sympoetic.net/Simulation_Models/GSS_files/2007%20Hoffman%20et%20al%20Simul%20for%20Sustain.pdf.
13. https://en.wikipedia.org/wiki/Simulation_hypothesis.
14. https://en.wikipedia.org/wiki/Simulated_reality.
15. https://unigis.at/weiterbildung/spatial-simulation/.
16. https://www.researchgate.net/publication/230802219_Multi-level_spatial_simulation.
17. O'Sullivan, David, and George LW Perry. "*Spatial simulation: exploring pattern and process*." John Wiley & Sons, 2013.
18. Darvishi, Mehdi, and Gholamreza Ahmadi. "Validation techniques of agent based modelling for geospatial simulations." *Int Arch Photogrammetry, Remote Sensing & Spatial Info Sci* (2014).
19. https://www.informs-sim.org/wsc15papers/004.pdf.
20. Anagnostou, Anastasia, and Simon JE Taylor. "A distributed simulation methodological framework for OR/MS applications." *Simul Mod Pra Theory* 70 (2017): 101-119.
21. D'Angelo, Gabriele. "Parallel and distributed simulation from many cores to the public cloud." In *2011 Int Conf High Perf Comp Simul* pp: 14-23. IEEE, 2011.
22. Frohn, Christian, Petyo Ilov, Stefan Kriebel, and Evgeny Kusmenko, et al. "Distributed Simulation of Cooperatively Interacting Vehicles." In *2018 21st Int Conf Intel Transp Sys (ITSC)*, pp: 596-601. IEEE, 2018.
23. https://en.wikipedia.org/wiki/Distributed_Interactive_Simulation.

24. Topçu, Okan, and Halit Oğuztüzün. *Guide to distributed simulation with HLA*. Springer International Publishing, 2017.
25. Hakiri, Akram, Pascal Berthou, and Thierry Gayraud. "Addressing the challenge of distributed interactive simulation with data distribution service." (2010).
26. https://www.researchgate.net/publication/251422110_Overview_about_the_High_Level_Architecture_for_Modelling_and_Simulation_and_Recent_Developments.
27. https://en.wikipedia.org/wiki/High_Level_Architecture.
28. https://en.wikipedia.org/wiki/Live,_virtual,_and_constructive.
29. https://www.scalable-networks.com/live-virtual-constructive-simulation-software.
30. Teng, Teck-Hou, Ah-Hwee Tan, and Loo-Nin Teow. "Adaptive computer-generated forces for simulator-based training." *Expert Systems with Applications* 40 (2013): 7341-7353.
31. https://en.wikipedia.org/wiki/Command_hierarchy.
32. Burgess, Alan, and Peter Fisher. "A Framework for the Study of Command and Control Structures." (2008).
33. https://www.thebalancecareers.com/chain-of-command-1918082.
34. http://c4isys.blogspot.com/2013/11/basics-of-information-operations-24.html.
35. https://en.wikipedia.org/wiki/Virtual_world.
36. https://en.wikipedia.org/wiki/Persistent_world.
37. Bell, Mark W. "Toward a definition of "virtual worlds"." *J Virtual Worlds Res*. (2008).
38. https://www.merriam-webster.com/dictionary/real-world.
39. https://en.wikipedia.org/wiki/Real_life.
40. https://en.wikipedia.org/wiki/Nature.
41. https://en.wikipedia.org/wiki/Executive.
42. https://en.wikipedia.org/w/index.php?title=Special:Search&search=intitle%3A%22Executive%22&ns0=1.
43. Sapaty, Peter Simon. "A Distributed Processing System." European Patent Office, Munich, 1993.
44. Sapaty, Peter Simon. "Mobile Processing in Distributed and Open Environments." John Wiley & Sons, New York, 1999.
45. Sapaty, Peter Simon. "Ruling Distributed Dynamic Worlds." John Wiley & Sons, New York, 2005.
46. Sapaty, Peter Simon. "Managing Distributed Dynamic Systems with Spatial Grasp Technology." Springer, 2017.
47. Sapaty, Peter Simon. "Holistic Analysis and Management of Distributed Social Systems." Springer, 2018.
48. Sapaty, Peter Simon. "Complexity in International Security: A Holistic Spatial Approach." Emerald Publishing, 2019.
49. Baumann, Joachim, Fritz Hohl, Kurt Rothermel, and Markus Strasser, et al. "MOLE: A mobile agent system." *Software: Practice and Experience* 32 (2002): 575-603.
50. https://link.springer.com/book/10.1007/978-981-13-8679-4.
51. https://en.wikipedia.org/wiki/Mobile_agent.
52. https://en.wikipedia.org/wiki/Telescript_(programming_language).
53. https://en.wikipedia.org/wiki/Code_mobility.
54. https://en.wikipedia.org/wiki/Interoperability.
55. Lange, Danny B, and Mitsuru Oshima. "Seven good reasons for mobile agents." *Communications of the ACM* 42 (1999): 88-89.
56. Sapaty, Peter Simon, Mj. Corbin, and Pm. Borst. "Mobile WAVE programming as a basis for distributed simulation and control of dynamic open systems. Report at the 4th UK SIWG National Meeting, SGI Reality Centre, Theale." *Reading* 11 (1994).
57. Sapaty, Peter Simon, Mj. Corbin, and Pm. Borst. "Towards the development of large-scale distributed simulations." In *Proc. 12th Workshop on Standards for the Interoperability of Distributed Simulations, IST UCF*, pp: 199-212. 1995.
58. Sapaty, Peter Simon, Mj. Corbin, and Pm. Borst. "Towards the intelligent infrastructures for distributed federations." In *Proc. 13th DIS Workshop*. 1995.
59. Sapaty, Peter Simon, 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.
60. Sapaty, Peter Simon. "Mosaic Warfare: From Philosohpy to Model to Solution, Mathematical Machines and Systems." 2019.
61. https://eagle.sbs.arizona.edu/sc/report_poster_detail.php?abs=3696.
62. Sapaty, Peter Simon. "Integral spatial intelligence for advanced terrestrial and celestial missions." J Aeronautics & Aerospace Engineering. (2015).

How to cite this article: Peter Simon Sapaty. "Symbiosis of Virtual and Physical Worlds under Spatial Grasp Technology". *J Comput Sci Syst Biol* 13 (2020) doi: 10.37421/jcsb.2020.13.321