

# The over-operability organization of distributed dynamic systems for asymmetric operations

By Peter S. Sapaty

Institute of Mathematical Machines and Systems, National Academy of Sciences, Kiev, Ukraine

## Abstract

A novel ideology and technology will be presented allowing distributed dynamic and open systems to behave as highly integral organisms pursuing global goals and effectively operating in dynamic and hostile environments. The technology sets a higher, *overoperability*, layer by implanting into important system points a universal control module communicating with other such modules. These, cooperatively and in parallel, without any central resources, interpret compact mission scenarios in a special distributed scenario language, which can start from any component and cover & match the whole system or its needed parts at runtime, establishing proper relations between the system elements, orienting global behaviour, and recovering from indiscriminate damages.

## 1. Introduction

We used to pursue atomistic, parts-to-whole philosophy in system formalization, design and implementation. Originally a system or campaign idea emerges in a very general, integral form in a single mind or close collective of minds (Figure 1a). Then it is mentally decomposed into parts (or atoms, agents) each subsequently detailed, extended, and clarified (Figure 1b).

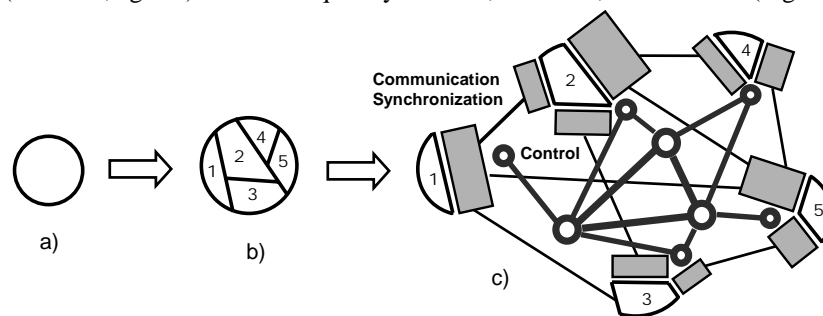


FIGURE 1. Traditional atomistic approach in system design and management. (a) the original system idea; (b) breaking the idea into pieces; (c) system formalization, distribution, and implementation.

Next, the clarified parts and their distribution are implemented in physical or virtual spaces. To make these parts working together as a whole within the original idea, we may need a good deal of their communication and synchronization, also a sophisticated control infrastructure, as in Figure 1c; this resulting in a considerable overhead often outweighing the whole idea itself.

The original idea and even its logically partitioned stage (Figure 1a,b) usually remain in the minds of creators only, whereas actual system formalization and implementation start from the distributed stage of Figure 1c. This often causes the following problems.

- It may be difficult to put the resultant distributed system with many linked parts into compliance with the initial idea.
- The reassembled whole may have quite different features than expected, including unwanted ones.
- The resultant solution is static. If the initial idea changes, the whole system may have to be partially or even completely redesigned and reassembled.
- Adjusting the already existing multi-component system to a new idea may result in a considerable loss of integrity and performance.

In the rest of this paper, we are briefly describing the *overoperability* approach and technology (Sapaty 2002, 2005, 2009b) allowing us to create integral distributed systems that react to changing environments and mission goals in a timely manner. This is in contrast, also supplement, to the traditional atomistic *interoperability* organizations (Desourdis & Rosamilia & et al 2009).

## 2. Higher-Level System Comprehension and Management

The model described here (see also Sapaty 1999 & 2005 & 2009b), following holistic and gestalt philosophies (where the whole is first and greater than the sum of parts, defining their sense or even existence), is based on spreading parallel interdependent waves in open worlds (Figure 2a).

In some sense, it formalizes the process of how human mind grasps distributed spaces by *mentally navigating them*, temporarily concentrating on key points, also scanning between them, for solving local and global problems. This is a highly integral and holistic process, rather than interaction of certain “agents” in the brain, on which many existing AI systems are attempted to be based and built (often stemming from Minsky 1988). For example, it is difficult if not ridiculous to imagine that, say, agent A tells something agent B under control of agent C when we perceive the world—the communicating agents may be wrongly lifted here from much lower (physical rather than mental) organizational layers of our body.

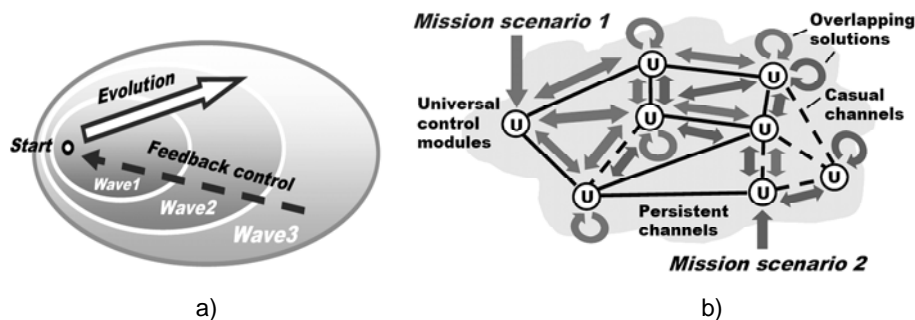


FIGURE 2. The waves paradigm. (a) controlled grasping of distributed world with spatial waves; (b) self-spreading high-level mission scenarios in distributed networked worlds.

The waves model underlying our approach is in line with known investigations (see Wilber 2009) that certain aspects of cognition, morals, needs, object relations, motor skills, and language acquisition proceed in developmental stages. These stages appear to be *fluid, flowing, and overlapping waves*. This is also consistent with the ideas of *self-actualization* and *person-centered approach* (Rogers 1988, Kriz 2008), where *the self* is considered as an organized, consistent, conceptual gestalt exhibiting *active forward thrust*—against tension reduction, equilibrium, or homeostasis (as in Freud 1997), the latter being ideologically close to the current agent-based models.

We have put the above mentioned ideas of wave-like mental coverage of physical or virtual spaces in the forward thrust mode onto highly parallel and fully distributed basis, which effectively allows any distributed systems to behave as an integral and self-conscious mind, comparable to that of a single individual. Atomism in our case emerges during automatic implementation only, and when required, allowing us to get flexible, easily changeable, high-level formal definitions of systems and operations in them, while omitting numerous organizational details (as in Figure 1c) and concentrating on global goals and global behaviour instead.

Materialization of this approach is carried out by the network of universal intelligent modules (U), embedded into important system points, which collectively interpret integral mission scenarios expressed in the waves formalism, starting from any point and covering the distributed system at runtime, in a viruslike mode (as in Figure 2b). Different scenarios can cooperate or compete in the networked space as overlapping fields of solutions.

The spreading scenarios, being extremely compact (can be created and modified on the fly), are forming dynamic knowledge infrastructures arbitrarily distributed between system components (robots, sensors, humans). Navigated by same or other scenarios, they can effectively support distributed databases, advanced command and control, also provide overall situation awareness and autonomous decisions.

### 3. Distributed Scenario Language

The Distributed Scenario Language, or DSL (similar to its previous versions, WAVE and WPL, as in Sapaty 1999 & 2005), reflects the waves model proposed, allowing us to directly express semantics of problems to be solved in distributed worlds, also the explicit local and global system behaviour to solve the problems, if needed. The language operates with:

- Virtual World (VW), which is discrete and consists of nodes and links connecting them.
- Continuous Physical World (PW), points in which are accessible by physical coordinates.
- Virtual-Physical World (VPW), as an extension of VW, where nodes additionally associate with coordinates in PW.

DSL also has the following key features:

- A DSL scenario develops as a transition between sets of progress points (or *props*) in the form of parallel waves.
- Starting from a prop, an action may result in one or more props (the resultant set of props may include the starting prop too).
- Each prop has a resulting value (which can be multiple) and a resulting state, being one of the four: *thru* (full success, also allowing us to proceed further), *done* (success with planned termination), *fail* (regular failure, with local termination), and *abort* (emergency failure, terminating the whole distributed process, associated with other props too).
- Different actions may evolve independently or interdependently from the same prop, forming altogether the resultant set of props.
- Actions may also spatially succeed each other, with new ones applied in parallel from all props reached by the preceding actions.
- Elementary operations can directly use local or remote values of props obtained from other actions (the latter including the whole scenarios).
- Elementary operations can result in open values that can be used by other operations in an expression or by the following operations in a sequence. They can also be directly assigned to local or remote variables (an access to which may invoke scenarios of any complexity).
- Any prop can associate with a node in VW or a position in PW, or both.
- Any number of props can be simultaneously linked with the same points of the worlds.
- Staying with world points (virtual, physical, or combined) it is possible to access and update local data in them.
- DSL can also be used as a usual universal programming language.

DSL has a recursive syntax, which on top level may be expressed as follows (programs are called *waves*, braces show repetition, and vertical bar delimits alternatives):

```
wave      → phenomenon / rule ( { wave , } )
phenomenon → constant / variable / special
constant  → information / matter
variable  → heritable / frontal / environmental / nodal
rule      → movement / creation / elimination / echoing / fusion / verification / assignment /
           construction / advancing / branching / transference / timing / granting / type / usage
```

The basic construct, *rule*, can represent any definition, action or decision, being for example:

- elementary arithmetic, string or logic operation;
- hop in a physical, virtual, or combined space;
- hierarchical fusion and return of (remote) data;
- distributed control, both sequential and parallel;
- a variety of special contexts for navigation in space, influencing operations and decisions;
- type or sense of a value, or its chosen usage, guiding automatic interpretation.

There are different types of variables in DSL:

- *Heritable variables* – these are starting in a prop and serving all subsequent props, which can share them in both read & write operations.
- *Frontal variables* – are an individual and exclusive prop's property (not shared with other props), being transferred between the consecutive props, and replicated if from a single prop a number of props emerge.
- *Environmental variables* – are accessing different elements of physical and virtual words when navigating them, also a variety of parameters of the internal world of DSL interpreter.
- *Nodal variables* – allow us to attach an individual temporary property to VW and VPW nodes; they can be accessed and shared by any props associated with these nodes.

These variables, especially when used together, allow us to create efficient spatial algorithms which work *in between* components of distributed systems rather than *in them*.

Elementary DSL programming examples are shown in Figure 3. (Traditional abbreviations of operations and delimiters can be used too, as in further examples throughout this text, to simplify and shorten DSL programs.)

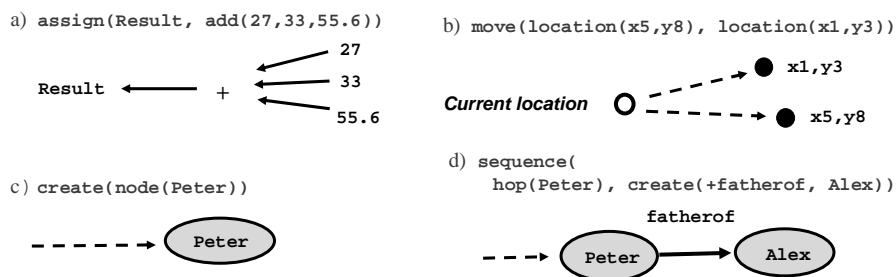


FIGURE 3. Elementary examples in DSL. (a) assignment of a sum of values to a variable; (b) parallel hop into physical locations; (c) creating a node in a virtual space; (d) extension with a new link-node pair.

#### 4. Distributed Interpreter

The DSL interpreter (as from the previous language versions) has been prototyped on different platforms. Its key features (see Figure 4) are as follows:

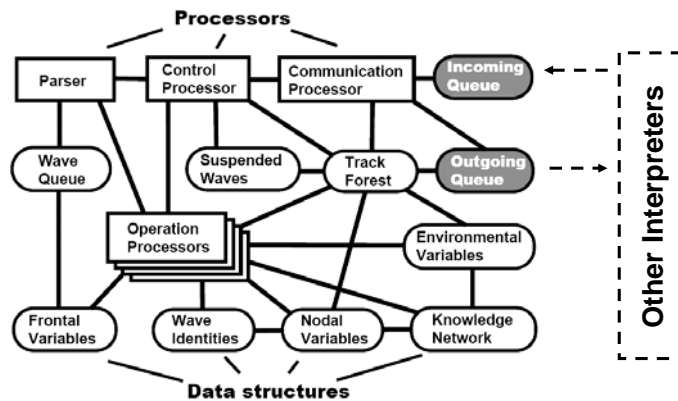


FIGURE 4. Organization of DSL interpreter.

- It consists of a number of specialized modules working in parallel and handling and sharing specific data structures supporting persistent virtual worlds and temporary hierarchical control mechanisms.
- The whole network of the interpreters can be mobile and open, changing at runtime the number of nodes and communication structure between them.
- The heart of the distributed interpreter is its *spatial track system*, with its parts kept in the Track Forest memory of local interpreters (see Figure 4); these being logically interlinked

with such parts in other interpreter copies, forming altogether indivisible space coverage. This enables hierarchical command and control and remote data and code access, with high integrity of emerging parallel and distributed solutions, without any centralised resources.

- Copies of the interpreter can be concealed (as in hostile systems), allowing us to impact them overwhelmingly (finding & eliminating unwanted infrastructures including).

The dynamically crated track trees (Figure 5a) spanning the systems in which DSL scenarios evolve, are used for supporting spatial variables and echoing and merging different types of control states and remote data, being self-optimized in the echo processes (Figure 5b). They also route further waves to the positions in physical, virtual or combined spaces reached by the previous waves, uniting them with the frontal variables left there by preceding waves.

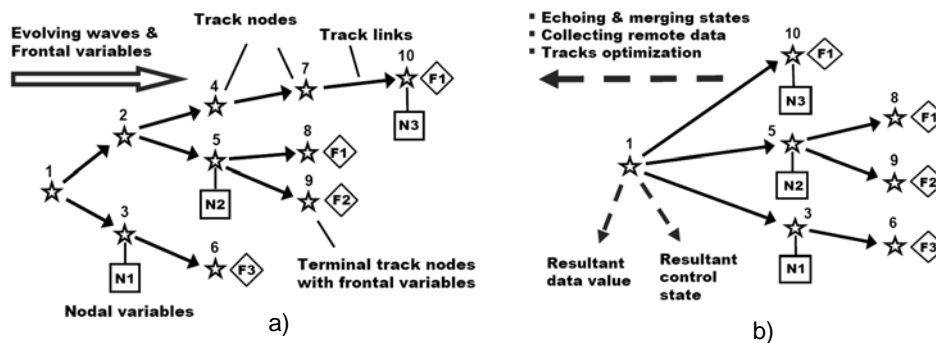


FIGURE 5. Distributed track system. (a) forward operations; (b) optimization in backward operations.

## 5. Programming Examples: Topology Analysis

DSL allows us to directly analyze and process distributed topologies in a parallel and concise way. To find the weakest nodes in a network (like *articulation points*, see Figure 6a) which, when removed, split it into disjoint parts, we only need the following program (resulting in d).

```
hop(allnodes); IDENTITY = CONTENT; Nmark = 1;
and((random(hop(alllinks));
  repeat(grasp(Nmark == nil; Nmark = 1); hop(alllinks))),
  (hop(alllinks); Nmark == nil),
  output(CONTENT))
```

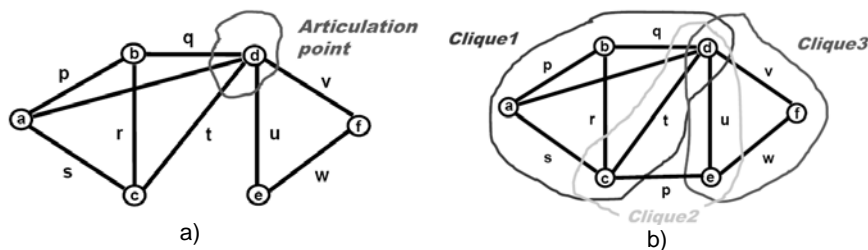


FIGURE 6. Discovering distributed topological features. (a) articulation points; (b) cliques.

Cliques (or fully connected sub-graphs of a graph, as in Figure 6b), on the contrary, may be considered as strongest parts of a system. They can be found in parallel by the following program, resulting for Figure 6b in: (a, b, c, d), (c, d, e), (d, e, f).

```
hop(allnodes); Fclique = CONTENT;
repeat(hop(alllinks); notbelong(CONTENT, Fclique);
  and(andparallel(hop(anylink, nodes(Fclique)); done),
  or((BACK > CONTENT; done), append(Fclique, CONTENT))));
output(Fclique)
```

The above programs work without any central resources, even if every node of the network is located in a different computer, with programs starting from any node (Sapaty 1999).

## 6. Programming Example: Collective Robotics

Installing DSL interpreter into mobile robots (ground, aerial, surface, or underwater, as in Figure 7) allows us to organize effective group solutions (incl. any swarming) of complex problems in distributed physical spaces in a concise way, shifting traditional management routines to automatic levels. Expressing explicit system behaviours at any levels is easy too.

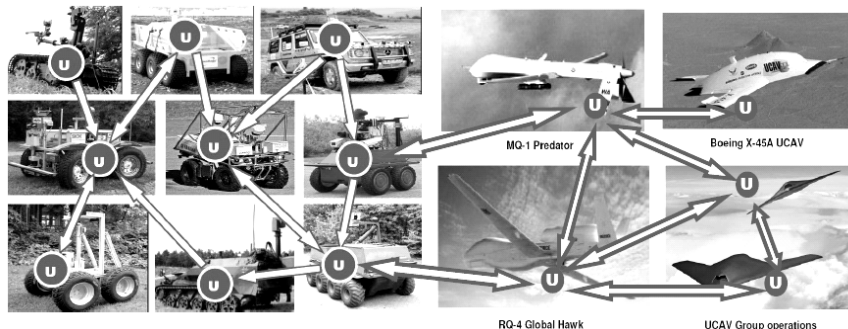


FIGURE 7. Grouping unmanned ground and aerial vehicles.

An example task here: *Go to physical locations of the disaster zone with coordinates (50.433, 30.633), (50.417, 30.490), and (50.467, 30.517). Evaluate damage in each location, find and transmit the maximum destruction value, together with exact coordinates of the corresponding location, to a management centre.* The DSL program will be as follows:

```
transmit(max(move((50.433, 30.633), (50.417, 30.490), (50.467, 30.517)));
         append(evaluate(destruction), WHERE)))
```

Details of automatic implementation of this scenario by different numbers of mobile robots can be found elsewhere (Sapaty 2009b).

## 7. Other Applications

Many other applications of the presented paradigm can be found in the previous publications (like in Darling 1998, Dragos & Collier 2004, Gonzales-Valenzuela & Vuong 2002, Sapaty 1999 & 2005 & 2009). We will mention here only the most active ones at present.

- *Emergency Management.* Using the interpreters installed in massively wearable devices may allow us to assemble workable systems from any wreckage after the disasters, using remaining communication channels, manual including (Sapaty & Sugisaka & et al 2006).
- *Directed Energy Systems.* The technology can provide high flexibility in organizing directed energy systems and weapons, especially in crisis situations, making automatic distributed decisions with the “speed of light” too (Sapaty & Morozov & Sugisaka 2007).
- *Distributed Avionics.* Implanting the interpreter copies into main control points of the aircraft may provide a higher, intelligent layer of its self-analysis and recovery, by the spreading recursive scenarios starting from any point and collecting & fusing key data from other points (Sapaty 2008).
- *Sensor Networks.* Wireless sensors may be dropped from the air massively, as “smart dust”. With a limited communication range, they must operate in a network to do nonlocal jobs in a distributed environment. With the technology offered, we can convert their emergent networks into a universal parallel computer operating in DSL (Sapaty 2009a).
- *Battlespace Dominance.* The technology can provide superiority over an adversary by flexible runtime adaptable system organizations, rather than by troop numbers or power of weapons (Sapaty 2009c), countering asymmetric situations by highly asymmetric solutions.

## 8. Conclusions

The overoperability technology presented can be readily incorporated into different national and international organizations, campaigns, and forces. At the beginning, for example, this can

serve as an intelligent extension to usual tactical communications (Sapaty 2009d), providing guidance and support for their runtime reorientation and restructuring and supplementing classical command and control. In a longer term, it can direct and support gradual transition toward robotized (and even fully unmanned) armies or civil crisis reaction forces operating in hazardous environments, with unified mission scenarios executed by both manned and unmanned components, having situation dependent subordination between the two.

## Acknowledgments

The author is grateful for the support of these ideas by: John Clifford at the Electronic Warfare Conference in London, Huaglory Tianfield at the International Transactions on Systems Science and Applications, Edward Carapezza at SPIE DSS Symposia in the US and Europe, Masanori Sugisaka from the AROB community and numerous symposia in Japan, and Juergen Kriz from the Scientific Convention of GTA in Germany. Special thanks to James Darling for his belief in this paradigm and resultant brilliant PhD in the virtual reality field, as well as to Mykola Mazuruk for invaluable help with solving numerous practical problems in the UK.

## REFERENCES

- DARLING, J.C.C. 1998. *The application of distributed and mobile computing techniques to advanced simulation and virtual reality systems*, PhD Thesis, Dept. EEng, University of Surrey.
- DESOURDIS, Jr.R.I., & ROSAMILIA, P.J. & JACOBSON, C.P & SINCLAIR, J.E. 2009. *Achieving interoperability in critical IT and communication systems*. Artech House Publishers, 411p.
- DRAGOS, S. & COLLIER, M. 2004. Macro-routing: a new hierarchical routing protocol. *Proc. 47th annual IEEE Global Telecommunications Conference, Globecom 2004*, Dallas, Texas, USA.
- FREUD, S. 1997. *General psychological theory*. Touchstone, 500p.
- GONZALEZ-VALENZUELA, S. & VUONG, S.T. 2002. Evaluation of migration strategies for mobile agents in network routing. *Proc. of the 4th International Workshop MATA'02*, Barcelona, Spain.
- KRIZ, J. 2008. *Self-actualization: person-centred approach and systems theory*. PCCS Books, 380p.
- MINSKY, M. 1988. *The society of mind*. Simon and Schuster, New York, 500p.
- ROGERS, C.R. 1978. *Carl Rogers on personal power: inner strength and its revolutionary impact*. Trans-Atlantic Publications, 40p.
- SAPATY, P. 2009. Distributed technology for global control. *Lecture Notes in Electrical Engineering 37*, Springer Berlin Heidelberg.
- SAPATY, P. 2009a. Remote control of open groups of remote sensors. *Proc. SPIE Europe Security + Defence*, Berlin, Germany.
- SAPATY, P. 2009b. Meeting the world challenges: from philosophy to information technology to applications. *Proc. 6th International Conference ICINCO 2009*, 1, Milan, Italy.
- SAPATY, P. 2009c. Distributed capability for battlespace dominance. *Proc. Electronic Warfare 2009 Conference & Exhibition*, London, UK.
- SAPATY, P. 2009d. High-level communication protocol for dynamically networked battlefields. *Proc. International conference Tactical Communications 2009*, London, UK.
- SAPATY, P. 2008. Grasping the whole by spatial intelligence: a higher level for distributed avionics. *Proc. International Conference Military Avionics 2008*. London, UK.
- SAPATY, P. & MOROZOV, A. & SUGISAKA, M. 2007. DEW in a network enabled environment. *Proc. International conference Directed Energy Weapons*, London, UK.
- SAPATY, P. & SUGISAKA, M. & FINKELSTEIN, R. & DELGADO-FRIAS, J. & MIRENKOV, N. 2006. Advanced IT support of crisis relief missions. *Journal of Emergency Management* 4, 4, 29-36.
- SAPATY, P.S. 2005. *Ruling distributed dynamic worlds*. Wiley, 256p.
- SAPATY, P.S. 2002. Over-operability in distributed simulation and control. *The MSIAC's M&S Journal Online*, Winter Issue 4, 2, Alexandria, VA, USA.
- SAPATY, P.S. 1999. *Mobile processing in distributed and open environments*. Wiley, 436p.
- SAPATY, P. 1993. *A distributed processing system*. European patent No.0389655.
- WILBER, K. 2009. *Waves, streams, states, and self--a summary of my psychological model (or, outline of an integral psychology)*. Shambhala Publications, 60p.