The Fourteenth International Symposium on Artificial Life and Robotics 2009 (AROB 14th '09),
B-Con Plaza, Beppu, Oita, Japan, February 5 - 7, 2009

Developing High-Level Management Facilities for Distributed Unmanned Systems

Peter Sapaty
Institute of Mathematical Machines and Systems, National Academy of Sciences
Glushkova Ave 42, 03187 Kiev, Ukraine, sapaty@immsp.kiev.ua

Klaus-Dieter Kuhnert
Institute of Real-Time Learning Systems, University of Siegen
Hölderlinstr. 3, D-57068 Siegen, Germany, kuhnert@fb12.uni-siegen.de

Masanori Sugisaka
Department of Mechanical and Electrical Engineering, Nippon Bunri University
1727 Oaza Itiki, Oita, 870-0397, Japan, ms@alife-robotics.co.jp

Robert Finkelstein
Robotic Technology Inc.11424 Palatine Drive, Potomac MD, USA
RobertFinkelstein@compuserve.com

Abstract

Due to increasing complexity of tasks delegated to unmanned vehicles, their collective use is becoming of paramount importance for performing any reasonable jobs. An approach is offered where group behaviors are accomplished automatically rather than set up manually, as usual. Missions in the Distributed Scenario Language (DSL) can be executed jointly by communicating interpreters in robotic units. Scenarios like reconnaissance, camp security, convoy, mule, and EOD in DSL, oriented on different numbers of cooperating vehicles, are demonstrated. The approach may allow us to effectively manage any robotic teams, both homogeneous and heterogeneous, regardless of the number of vehicles in them.

Keywords: unmanned systems, scenario language, robotic teams, swarms, distributed interpretation, reconnaissance, camp security, convoys, EOD, high-level management.

1 Introduction

With the world dynamics increasing due to global warming, numerous natural and manmade disasters, military conflicts, and international terrorism, using unmanned (ground, sea, underwater, and air) systems can alleviate many problems and save lives in hazardous environments. Because of the complexity of tasks delegated to unmanned solutions and still insufficient capabilities of existing robotic vehicles, their simultaneous, collective use may be of paramount importance to perform any reasonable jobs. Operating together, the unmanned groups, often called swarms, can fulfill the required objectives despite possible runtime damages to individual units.

We are offering a novel approach to organization of unmanned swarms, oriented from the very beginning on parallel solutions in physical spaces, with swarm behaviors resulting naturally and accomplished automatically, rather than programmed manually. A mission scenario, written in a special high-level language and reflecting semantics of what to be done in a distributed space rather than details of implementation, is executed in a cooperative manner by dynamically networked unmanned units.

The current paper is inspired by the European Land Robotic Trial ELROB 2008 [1], in which the authors participated. It was conducted to provide trials as close as possible to operational scenarios for UGVs/UAVs with focus on short-term realizable robot systems. The day and night trials were organized within the following five main scenarios: non-urban reconnaissance, camp security, transport convoy, transport mule, and explosive ordnance disposal. Only a limited number of robotic units was engaged in every scenario, just one or two, whereas every scenario could potentially be executed with much higher efficiency if using robotic teams with many units, which cooperate with each other.

The paper reflects an initial state of the international project under sponsorship of Alexander von Humboldt Foundation (AvH) in Germany. Its aim is formalization of known mission scenarios in such a way that they could be performed by any available numbers of robotic vehicles, with the management burden shifted to self-organized robotic teams--thus relieving human operators from tedious routines and allowing them concentrate on mission goals and overall efficiency.

The main features of the Distributed Scenario Language and its parallel interpretation in distributed environments are briefed, with more details on the underlying paradigm and its applications easily obtainable from the previous publications [2-4]. The formalization and expression in DSL of the main ELROB 2008 scenarios is provided on a semantic level, with the scenarios to be potentially executed by any number of mobile robots, and overall management of robotic swarms fully shifted to the self-organized network of DSL interpreters.

The scenario examples in DSL (unfortunately, without detailed comments, due to page limits) show compactness of the DSL code, which can be written on the fly, thus timely responding on dynamics of the

©ISAROB 2009

environment and changing system goals. More on the underlying spatial algorithms and related DSL code interpretation will be in subsequent publications, including the forthcoming project report for AvH.

2 Distributed Scenario Language (DSL)

Our approach is based on the Distributed Scenario Language (DSL), describing what to do in distributed spaces rather than how to do, and by which resources, leaving the latter to an effective automatic interpretation in networked environments. The DSL main features include:

- Association of different actions (which may be performed in parallel) with positions in physical, virtual, or combined spaces
- Working with both information and physical matter
- Runtime creation of distributed knowledge networks
- Distributed decision making
- Automatic command and control

DSL has a recursive syntax that can be expressed on the top level as follows (square brackets are for an optional construct, braces mean construct repetition with a delimiter at the right, and vertical bar separates alternatives).

wave	→	*constant* | *variable* | [*rule*] ({*wave* , })
constant	→	*information* | *matter*
variable	→	*nodal* | *frontal* | *environmental*
rule	→	*evolution* | *fusion* | *verification* | *essence*
evolution	→	*expansion* | *branching* | *advancing* | *repetition* | *granting*
fusion	→	*echoing* | *processing* | *constructing* | *assignment*
verification	→	*comparison* | *membership* | *linkage*
essence	→	*type* | *usage*

A rule is a general construct which can be:

- Elementary arithmetic, string or logic operation
- Hop in a physical, virtual, or combined space
- Hierarchical fusion and return of (remote) data
- Parallel and distributed control
- Special context for navigation in space
- Sense of a value for its proper interpretation

Different types of variables, especially when used together, allow us to create efficient spatial algorithms which work *in between* components of distributed systems rather than *in* them. The *nodal* variables can store and access local results in the system points visited, while other ones can transfer data in space together with the evolving control (*frontal* variables) or access and impact the internal and external world (*environmental* variables).

Due to peculiar syntax and semantics, the language parallel interpretation in distributed systems is transparent and straightforward, and does not need any central resources. DSL can dramatically simplify application programming in distributed environments, which is often much more concise and simple than in traditional programming languages.

3 Distributed DSL Interpreter

The DSL interpreter may be embedded in internet hosts, robots, mobile phones, or smart sensors (the interpreter can also be a human being herself, understanding and executing high-level orders in DSL and communicating with other humans or robots via the language syntax). The interpreter copies may be concealed, if needed (say, to work in a hostile environment); they can also migrate freely, collectively executing (mobile too) mission scenarios, resulting altogether in a flexible and ubiquitous system organization.

The interpreter [3] consists of a number of specialized modules working in parallel and handling & sharing specific data structures, which are supporting both persistent virtual worlds and temporary hierarchical control mechanisms. The heart of the distributed interpreter is its spatial track system enabling hierarchical command and control and remote data and code access, with high integrity (or "consciousness") of emerging parallel and distributed solutions, achievable without any central facilities.

In application to robotic communities, the approach allows us to convert any group of mobile robots into a goal-directed cooperative system by integrating copies of the DSL interpreter, as a universal control module U on Fig. 1, with traditional robotic functionalities, like the ones described in [5]. (The figure uses pictures of mobile robots participated in the ELROB 2008 trial.)

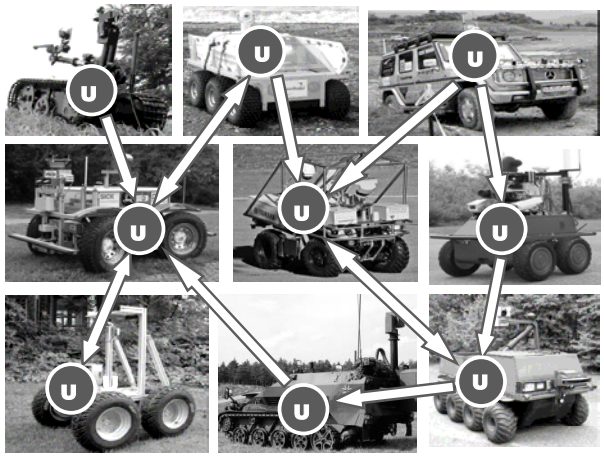

Figure 1. Heterogeneous robotic teaming using embedded DSL interpreters.

Any mission scenario in DSL can start from any robot, covering and tasking the whole system (or its parts needed) at runtime and in parallel. Subordination between the units and dynamic command and control are established automatically--as a derivative of the mission scenario and current state of environment.

Due to fully interpretive nature of the technology, the scenarios can self-recover from any point, timely reacting on failures of robots. The whole group may remain fully functional and global-goal-oriented even in case of indiscriminate damages to individual units.

4 Non-urban / Reconnaissance

For this scenario, it is supposed that a group of unknown vehicles is located in some distance in a non-urban area (defined, say, with the position of a center and area's radius), with security situation unclear there, so

the reconnaissance should be done by robotic vehicles for not risking own personnel. The objective is to go to this target area and search for vehicles with specific characteristics. If found, they should be examined in detail, with their parameters collected and reported to the control station.

The general picture is shown in Fig. 2, where the reconnaissance facilities should first go to the target area (i.e. its center), observe the area by cameras/sensors to roughly locate most probable targets (by their size, for example). The next will be to move directly to these selected targets and sense & collect their detailed parameters, with sending the results to the control point where they are stored and analyzed.

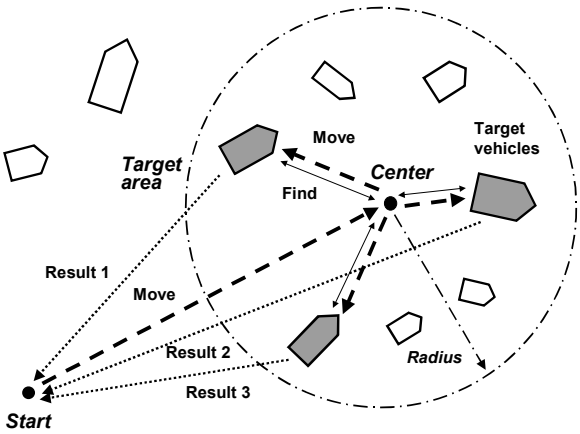

Figure 2. The reconnaissance scenario.

Parallel solution. This solution in DSL may allow us to use as many reconnaissance vehicles as possible (a single one including), potentially involving individual vehicles for each target identified, for their detailed examination.

```
USER = (
 move (start); WHERE = center;
 Targets = recognize (radius, features);
 split (Targets); WHERE = VALUE;
 collect (size, type, speed))
```

Explicitly sequential solution. The following DSL program just details navigation and organization procedures to execute the reconnaissance scenario in a strictly sequential way, which may be useful for optimization of the use of a single vehicle only.

```
move (start); WHERE = center;
Targets = recognize (radius, features);
loop (
 (Next = withdraw (Targets, 1)) != nil;
 WHERE = Next;
 Result &= collect (size, type, speed));
USER = Result
```

Avoiding obstacles. The movement to the target area and inside it may be complicated due to presence of obstacles, as shown in Fig. 3. The following DSL program, for the move from *Start* to *Center*, uses an external procedure `approach_or_stop` to detect obstacles and stop to avoid collision, and the procedure `suitable` to find next suitable waypoint on the way to the destination, from which the move should continue.

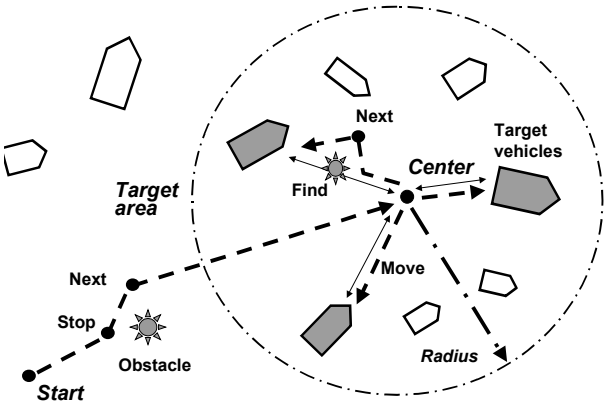

Figure 3. Avoiding obstacles.

```
move (start);
loop (approach_or_stop (center);
      WHERE ! = center;
      Next = suitable (depth, center);
      WHERE = Next)
```

5 Camp Security

For the camp security scenarios, a defined urban area has to be monitored (think military camp) and this should be executed by robotic vehicles too, to minimize risk to human personnel. The objective is to detect and report irregularities in the area, like intruders, while acquiring their positions and imagery, and transmitting to control station.

The general picture is shown in Fig. 4, where the camp units (numbered 1 to 6) are simultaneously patrolled by a number of robotic vehicles moving along the paths between and around the buildings.

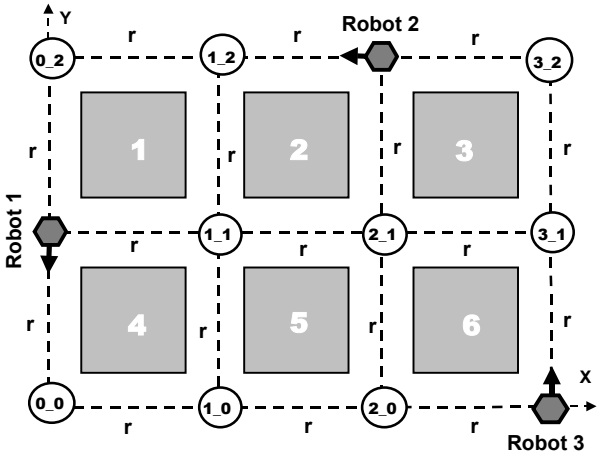

Figure 4. Camp security scenario.

Distributed campus map. The proper routing of vehicles and resolution of possible conflicts between them (like collision avoidance) can be assisted by the creation of a distributed map of the campus area (just reflecting Fig. 4) by the following DSL program (with node names reflecting X-Y coordinates of the crossings, and all links named `r`):

```
create (#3_1; F1=A; r#2_1; F2=A;
 r#1_1; F3=A; r#0_1;
```

```
(r#0_2; r#1_2; r#F3, (r#2_2; r#F2,
 (r#3_2; r#F1))),
(r#0_0; r#1_0; r#F3, (r#2_0; r#F2,
 (r#3_0; r#F1))))
```

Random movement. The next program organizes duty performance by three parallel processes (which may be executed by three robots) using the created distributed map, with random choice of the next-hop crossing and activation of the external service procedure `move_check_report` to analyze the local security situation while on the move.

```
hop (0_1, 2_2, 3_0);
WHERE = CONTENT;
repeat (
 or (
  (hop (link (random));
   grasp (Mark == nil; Mark = 1);
   (hop (BACK); Mark) = nil;
   move_check_report (CONTENT)),
  stay))
```

Movement via predetermined routes. If to use predetermined routes only, like shown in Fig. 5 (one route using links named `r1` and another one `r2`), the collisions between robots can be avoided in full.

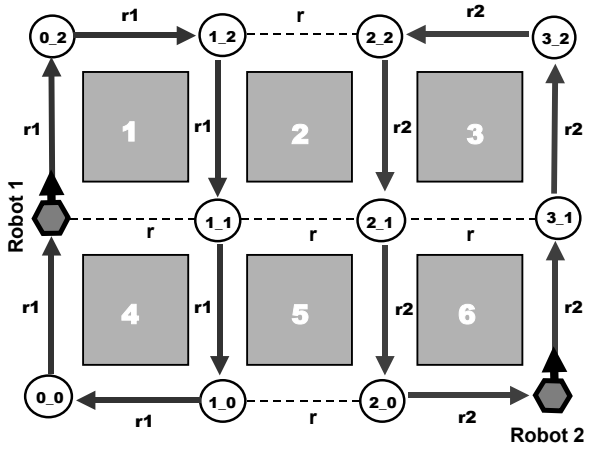

Figure 5. Using predetermined routes

Additional links `r1` and `r2` in the campus map can be installed by the following DSL program:

```
Linkup (
 (#0_2; r1#1_2; r1#1_1; r1#1_0; r1#0_0;
  r1#0_1; r1#0_2),
 (#3_2; r1#2_2; r1#2_1; r1#2_0; r1#3_0;
  r1#3_1; r1#3_2))
```

And two independent spatial processes navigating the campus via the new links (which may engage two robots) can be organized by the following parallel DSL code:

```
(hop (0_1); Flink = +r1),
(hop (3_0); Flink = +r2);
WHERE = CONTENT;
repeat (
  hop (link (Flink));
  move_check_report (CONTENT))
```

Any imaginable combinations of different types of simultaneous movement through the camp (like those by predetermined routes and/or by free, random, wandering) with collision avoidance can also be easily organized in DSL.

6 Transport Convoy

Imagine there is a delivery for a camp located in some distance. The objective is to move at least two vehicles to the target location, where only the first one can be manned and the second should follow the route of the first one, on a certain distance from it. We will consider a fully robotic solution for such a convoy, with two and also any number of vehicles, where only the first vehicle knows (and follows) waypoints toward the target location, while others dynamically chaining with, and following the previous ones on the move.

Two-unit convoy. It is represented by the communicating `Leader` and `Follower`, where the first one defines its movement by a sequence of waypoints, and the second one, regularly requesting the Leader, moves to the positions previously occupied by it, while keeping a certain threshold distance. This is shown in Fig. 6, and by the DSL program that follows

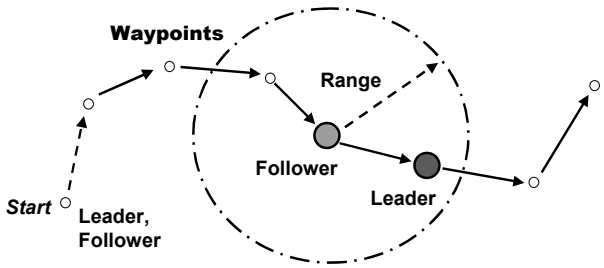

Figure 6. Two-unit convoy.

```
move (start);
(create (Leader);
 Waypoints = (w1, w2, w3, …);
 loop (
  (Next = withdraw (Waypoints,1))!= nil;
   WHERE = Next)),
(create (Follower);
 sling (
  Lcoord = (hop (range, any); WHERE);
  distance (WHERE, Lcoord) > threshold;
  WHERE = Lcoord))
```

Multiple-unit convoy. A scenario for the convoy with any number of chained processes (to be materialized by robotic units) is described by the following DSL program and depicted in Fig. 7. For this case, only the first process is a pure leader and the last process is a pure follower, while all other processes combine both functionalities, i.e. being followers for the previous processes and leaders for the subsequent ones.

```
move (start);
cycle (N < number; create (N += 1));
(NAME == 1; Waypoints = (w1, w2, w3, …);
 loop (
  (Next = withdraw(Waypoints, 1))!= nil;
   WHERE = Next)),
(NAME != 1;
 sling (
  Lcoord = (hop (range, NAME-1); WHERE);
```

```
distance (WHERE, Lcoord) > threshold;
WHERE = Lcoord))
```

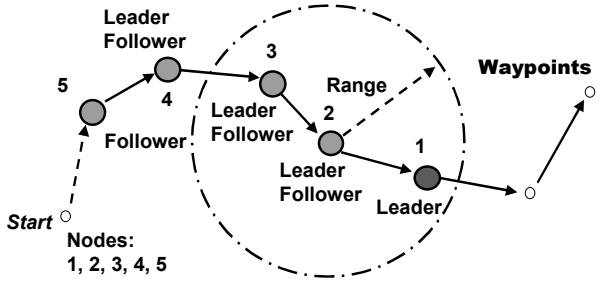

Figure 7. Multiple-unit convoy.

7 Transport Mule

Fir this scenario, there are two camps with a certain distance in between, and a cargo with a given weight should be transferred between the camps. We will consider here different possibilities to deliver payload between the camps, using unmanned vehicles as "mules".

In a single piece. This may be the case if cargo's weight allows it to be put on a single vehicle, as shown ion Fig. 8.

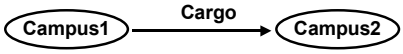

Figure 8. Single piece cargo delivery.

The related DSL program will be as follows:

```
move (Campus1);
frontal (Cargo) ="substance";
move (Campus2); Store = Cargo
```

Shuttling between camps. For this option, the process shuttles as often as possible between the two camps after partitioning the cargo into portions for the weight allowed, unless all the cargo is delivered, as shown in Fig. 9 and by the following program.

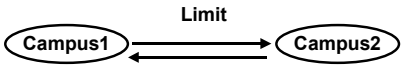

Figure 9. Shuttling delivery.

```
move (Campus1); frontal (Load);
Cargo = "substance"; Limit = 50;
loop (
 or (
 (weight (Cargo) > Limit;
  Load = withdraw (Cargo, Limit)),
 (weight (Cargo) > 0; Load = Cargo));
 hop (Campus2); Store += Load;
 hop (Campus1))
```

Multiple, parallel delivery. For this case, different processes (vehicles) are considered to be independent from each other, each moving to the destination as quickly as possible on its own (see Fig. 10 and the following program).

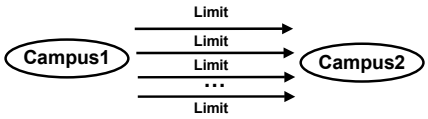

Figure 10. Parallel cargo delivery.

```
move (Campus1); frontal (Load);
Cargo = "substance"; Limit = 50;
cycle (
 or (
 (weight (Cargo) > Limit;
  Load = withdraw (Cargo, Limit)),
 (weight (Cargo) > 0; Load = Cargo)));
move (Campus2); Store += Load
```

Multiple, convoy delivery. For this scenario, the vehicles, each with a limited partition of cargo, are dynamically chaining in a column for a cohesive movement towards the destination (see Fig. 11 and the subsequent DSL program).

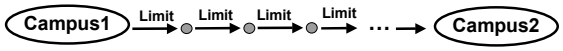

Figure 11. Delivery in a convoy.

```
move (Campus1); frontal (Load);
Cargo = "substance"; Limit = 50;
cycle (
 Or (
  (weight (Cargo) > Limit;
   Load = withdraw (Cargo, Limit)),
  (weight (Cargo) > 0; Load = Cargo));
 create (N += 1));
(NAME == 1; move (Campus2)),
(NAME != 1;
 loop (WHERE != Campus2;
       WHERE = (hop (NAME-1); WHERE));
Store += Load
```

8 Explosive Ordnance Disposal

Explosive Ordnance Disposal (EOD) means the detection, identification, onsite evaluation, rendering safe, recovery, and final disposal of Unexploded Ordnance (UXO) including detonation and burning. It is often said that the EOD operation is a 3 Ds one, which is Dangerous, Dirty and Demanding (or Difficult) job. Using robotic vehicles, especially multiple ones, is therefore becoming the most promising EOD option.

Various kinds of EOD scenarios for navigation and examination of the target territory may be offered. We will just hint here on the simplest two options, easily expressible in DSL.

Sequential territory search. This represents a single-thread process (oriented on a single vehicle), where the whole territory is incrementally scanned unless all being searched, as described by the following program and depicted in Fig. 12.

```
X1 =…, X2 =…; Y = Y1; Y2 =…; DY =…;
loop (WHERE = (X1, Y); (Y += DY) < Y2;
      WHERE = (X2, Y); (Y += DY) < Y2)
```

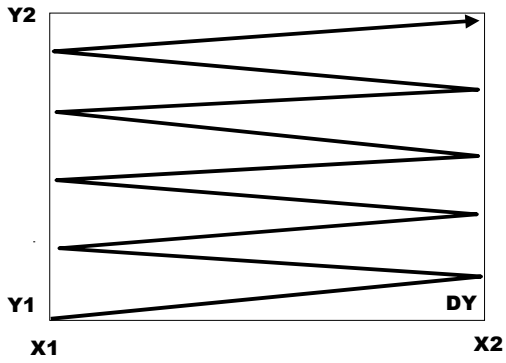

Figure 12. Sequential navigation.

Parallel territory search. This can be represented by a number of independent processes, each starting from a different location, and navigating altogether the whole region in parallel, as depicted by Fig. 13 and explained by the DSL program that follows.

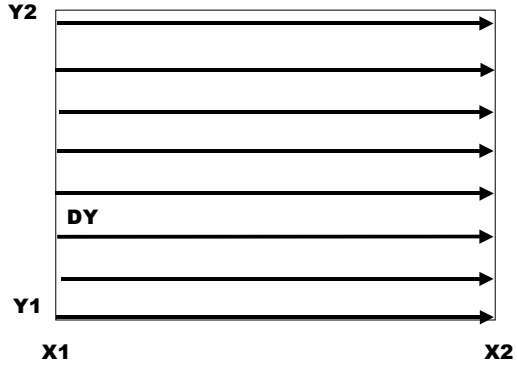

Figure 13. Parallel navigation.

```
X1 = …; X2 = …; Y1 = …; Y2 = …; DY = …;
frontal (Y) = Y1; DDY = 0;
cycle ((Y += DDY) < Y2; DDY += DY);
WHERE = (X1, Y); WHERE = (X2, Y)
```

9 Conclusions

Programming multi-robot scenarios in distributed and dynamic environments using DSL may not be more complex than, say, programming routine data processing tasks in traditional languages like Fortran, C, or Java, where the latter can run on parallel computers using any available number of processors. In our case, any multi-robot system may also be treated as a universal parallel computer (more correctly: parallel machine, as it operates not only with information but with physical matter or objects too).

The mission scenario in DSL may set up top level semantics of what, and generally how, should be done in a distributed space, and which key decisions should be taken in different spatial locations, to fulfill the objectives, regardless of the number of available robotic units, which may vary overtime (as some robots can be destroyed while others entering the operational field at runtime). This may essentially (and in many cases completely) relieve the human manager from traditional tedious routines of handling distributed multi-component systems, shifting the organizational burden to parallel scenario interpretation in the self-organized network of DSL interpreters.

This also gives us new opportunities for self-recovery in hostile environments as, first, the DSL scenarios may be free from mentioning any hardware robotic components (like computational problems in Fortran not mentioning computer registers or functional units and data transfers between them), and failures or recovery of particular robots may not be the business of the application program but rather of the internal system organization. And, second, due to fully interpretative nature of DSL, self-spreading scenarios in it may themselves recover from any point (robot), or even be self-relaunching from the beginning in most unfavorable situations. Using DSL, the human operator may effectively control distributed robotic swarms regardless of the number of units in them, just like controlling a single robot remotely, due to high self-organizational level of the swarms within the technology offered.

The current work is in progress, and in parallel with re-implementation of the technology on new, robotic, platforms (its existing public domain is mostly used for intelligent network management), we are considering various scenarios of engagement of heterogeneous unmanned systems for solving complex tasks. One of these is investigation and development of exemplary behaviors integrating energy (e.g. biofuel) seeking foraging robots [6] with other types of vehicles having specific payloads (the latter supposedly consuming the energy produced by the former). The foraging robots are dedicated to operate autonomously and for a long time in remote unpopulated areas, and heterogeneous swarms with them may represent a promising approach for advanced applications.

We also hope that the current project may help the next, ELROB 10, event to use more cooperating robots and even their swarms within the scenarios discussed, and, possibly, quite new ones.

Acknowledgment. This work has been funded by the Alexander von Humboldt Foundation in Germany.

References

[1] Military European Land-Robot Trial**,** M-ELROB 2008: 30 June – 03 July 2008, Hammelburg, Germany, http://www.m-elrob.eu/.

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

[3] P. S. Sapaty, Ruling Distributed Dynamic Worlds, John Wiley & Sons, New York, May 2005, ISBN 0-471-65575-9.

[4] P. Sapaty, Distributed Technology for Global Dominance, Proceedings of SPIE – Volume 6981, Defense Transformation and Net-Centric Systems 2008.

[5] K.-D. Kuhnert, M. Krödel: Autonomous Vehicle Steering Based on Evaluative Feedback by Reinforcement Learning, MLDM 2005.

[6] R. Finkelstein, EATR: Energetically Autonomous Tactical Robot, DARPA Contract W31P4Q-08-C-0292.