1

INTRODUCTION

We start with naming some exemplary areas that need radically new ideologies and technologies aimed at efficient organization of large distributed systems, explaining the existing problems of their understanding and management. Introductory ideas and simple practical examples explaining the essence of WAVE-WP (or World Processing), a novel technology that can rule a variety of distributed artificial and natural systems on a high, semantic level, are presented and discussed. Comparison with the predecessor model WAVE revealed in a previous book (Sapaty, 1999a), as well as the relation to other works in the area, is given, and the general organization of the book is outlined.

1.1 TOWARD COORDINATION AND MANAGEMENT OF LARGE SYSTEMS

1.1.1 Shifting from Computation to Coordination

The use of computers is steadily shifting from computations to coordination and management of complex distributed and dynamic systems, in both civil and military areas. Removing consequences of earthquakes and flooding, recovery after terrorist

Ruling Distributed Dynamic Worlds, by Peter S. Sapaty
ISBN 0-471-65575-9

attacks, or joint military operations in the world's hot spots are examples where distributed computerized systems may be particularly useful and efficient.

Both national and international campaigns of the scale and complexity never seen before may need to be planned, simulated, and managed, often within severe time constraints. These campaigns may use simultaneously within one system many human, technical, and natural resources distributed over large territories.

Radically new integration and coordination models and technologies may help such systems operate efficiently and fulfill their objectives, as traditional ones, stemming, say, from the Turing machine (Herken, 1995; Turing, 1936) or cellular automata (Wolfram, 1994) and oriented on computations, are becoming a real bottleneck.

1.1.2 Overoperability Versus Interoperability

Many current works in the field of integration and simulation of large systems orient on the support of interoperability between system components, where any unit can plug into a computer and see the same picture, say, of a battlefield. It is not easy to do, however, and billions of dollars are being spent on interoperability problems (Erwin, 2002).

This, however, may not be sufficient for the real integration, nor may be needed in general (Sapaty, 2002). For example, a tank driver should operate and make individual decisions within her direct vision only, and a local commander, executing orders from above, has to know the region where troops subordinate to her operate, rather than the whole battlefield.

The common picture is logically (not always practically, however) the simplest way to glue together unstructured or poorly organized systems. It reminds us, in some sense, the first multiprocessor computers which were designed decades ago. They were based on a common memory box, as the first feasible solution. In highly dynamic systems like real or symbolic battlefields, the huge amount of information they possess is an integral and inseparable part of the processes there. Making this information available for many players means that we must regularly create its snapshots and distribute them among the participants.

This may require highly intensive communications and broadband channels, which in severe and hostile environments may become a painful bottleneck. Also, as the information in such systems is changing with great speed, any snapshot of it may potentially be useless and "dead." And for security reasons, especially in military applications, the global system picture may need to be hidden rather than commonly available and shared.

Real integration may be based on more rational system principles, where parts are not necessarily equal to share the common picture (or do not share it at all, in the usual sense). They may rather form altogether a highly organized whole that defines the role of its parts and their interactions and not the other way round. The system's parts may lose their identity within such an organization, with their main goal being to satisfy the whole rather than evolve and prosper themselves.

Usually, such higher level organizations are a prerogative of human intelligence and manned command-and-control infrastructures. But emerging new types of

distributed systems, especially those using automated and unmanned platforms and oriented on solving dynamic tasks in unpredictable environments, may require shifting of an essential part of this "overoperability" to the efficient automatic level.

Traditional agent-based philosophies and approaches (Earnshaw and Vince, 2002; Bigus et al., 2001; Lange and Oshima, 1998; Milojicic et al., 1999; Siegwart et al., 2004; Wooldridge, 2002), first designing parts and then assembling the whole from these parts in the hope it works properly, may not be able to cope with large dynamic systems, and higher level organizational models and technologies, directly supporting the system whole and guaranteeing the needed global behavior, are of growing importance.

1.1.3 Intelligent Systems Versus Intelligent Components

Theoretically, many jobs in dynamic environments can be performed by teams of robots better than by individual robots (Gage, 1993). And, technologically, any number of sophisticated mobile robots can be produced by industry today. But we are still far away from any suitable team solutions that could allow us to use this abundant and cheaper hardware efficiently, as the teamwork is a complex and insufficiently studied phenomenon, with neither universal nor even suitable results proposed so far.

A considerable increase in the robot's individual and group intelligence at the current stage may, however, be achieved by raising the level of language with which robots, especially their teams, are programmed and tasked. This can be used subsequently as a qualitatively new platform for the creation of advanced robotic systems that may integrate different existing control approaches or may be based on radically new, higher level organizational models.

We already have such precedents. Only with the introduction of high-level programming languages such as FORTRAN or LISP did real and massive use (as well as production) of computers began. It became possible to express the semantics of problems directly, abstracting from hardware details and drastically improving application programming productivity, with the system intelligence shifting from computer system components to programming languages and the quality of their implementation.

We can pursue a similar approach for multirobot systems, considering them not as a collection of intelligent individuals (which may never reach human- or even animal-like level), but rather as a parallel machine capable of executing any mission tasks written in a universal high-level scenario language (Sapaty, 2001a). Its automatic implementation, which may be environment dependent and emergent, can be effectively delegated to the distributed artificial brain embedded in, and formed by, multiple robots, as shown in Figure 1.1.

Having designed such a language, we now have the possibility of programming and organizing highly intelligent distributed systems as a whole, rather than assembling them from self-contained intelligent elements, which may be lost indiscriminately, say, on a battlefield, putting at risk the whole system. The needed system intelligence may always be kept *in the mission scenario* regardless of the run time composition of a

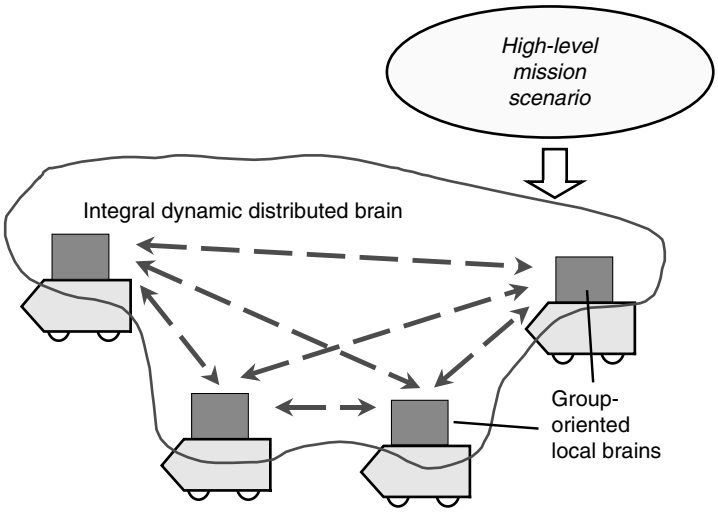

Figure 1.1. Group of robots as a parallel distributed machine.

system executing this scenario, thus enabling us to fulfill the mission objectives under any circumstances.

1.1.4 Directly Operating in Physical World

We also need models that can describe much broader activities than traditional information processing, enabling us to operate in real worlds directly, with physical movement and physical matter and object transference and manipulation.

Let us consider some examples of doing jobs in a physical world that may hint at how world-processing models and languages can be organized. For instance, if we want to perform a job, we first find a proper *place*, say, by a landmark or deviation from the current place (a hill at the right, 600 meters northwest, etc.), then move there with *operations* (machinery or equipment), and *data*, which may comprise both information and physical matter (a written plan of the job and, say, sand, bricks, water, ammunition, etc.).

Having performed the job (building a house, firing ammunition) using the data brought and/or existing there (say, left by other works), we may leave new data and the changes in the environment (the house built), pick up other data (unused materials), and move to other points, together with other operations (fresh or released equipment), to do another job (building a new house or a bridge), and so on. We may also need to bring results back to some point to be analyzed and processed (sand or water collected remotely, weapons or information seized in a reconnaissance operation, soil samples returned by a rover to a lander) to do other jobs or move further in case of success or failure.

Many jobs can be done simultaneously in the same or in different locations in the physical world. In performing jobs, we may cooperate or compete with others, seeing where, what, and how they do (moving in a chain, e.g., needs leveling and keeping distance with neighbors). Moving and doing jobs may depend on results and states reached in other places (deployment of a combat group only after success of another group, or entire mission abortion if the headquarters has been destroyed). Global control over evolving jobs may coexist with autonomous decision making.

To make real progress in this direction, we may need the design of a universal model, language and technology that would allow for an efficient description and implementation of a variety of parallel activities in a distributed physical world, similar to those sketched above.

1.1.5 Distributed Artificial Life

Life on Earth is a collective rather than individual phenomenon. It is also a distributed one. Different biological species can survive only if they communicate, and there are enough of them, being efficiently spread over vast territories. Artificial life approach (Langton, 1997; Ward, 2000) is so far considered mostly the creation and reproduction of individual life and single robotic species.

It is the right time to investigate a higher level now where distributed artificial systems and multiple robots can be really useful and can evolve if applied massively and work cooperatively. New advanced approaches for describing distributed autonomous systems on much higher levels than usual may be needed.

This may allow us to comprehend the very sense and basic evolution mechanisms of artificial life on a semantic level and also find, for instance, the *critical robotic mass* for this life to exist and prosper, using for simulation both single and parallel computers as well as computer networks.

The areas listed above are only a few of many where the design, or even invention, of radically new system creation and management models and technologies, like the one described in this book, are of highest interest and importance at present.

1.2 PROBLEMS OF MANAGING LARGE DISTRIBUTED SYSTEMS

Single-machine solutions often exhibit highest possible integrity as a system, with all resources at hand, and control being global, direct and absolute. On the contrary, the creation and management of large distributed systems that should behave as a single controllable entity pursuing global goals and recovering from damages can meet serious difficulties. Let us explain why.

1.2.1 From Localized to Distributed Solutions

In Figure 1.2 a symbolic splitting of the single-machine functionality into pieces for their distribution is shown, which may be based on separating by operations, separating by data, or by both.

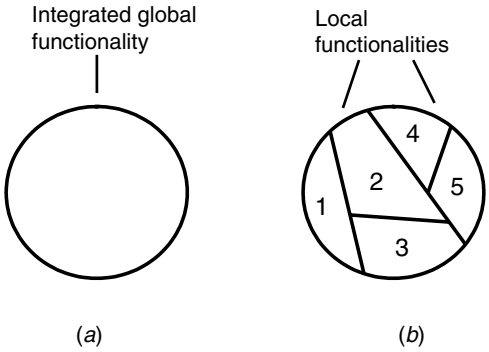

Figure 1.2. Partitioning of the problem for distribution: (*a*) single-machine solution and (*b*) breaking into pieces.

The resultant local functionalities may need to exist in more than a single copy in the distributed system (i.e., they should be replicated and spread throughout the space rather than localized and shared) for performance optimization, say, to become closer and work directly with other parts, and also to operate simultaneously with each other.

To work together as a distributed system, the parts of Figure 1.2*b* may need scores of other things to be additionally set up and integrated with them. First of all, there should be *communication and synchronization*, allowing them to exchange data via the *data channels*, to be established too. Second, the parts should have between (and over) them a sort of *command-and-control (CC) infrastructure*, to provide both local and global coordination and the ability to pursue common goals and obey global instructions. Within this infrastructure, additional *control centers* may be needed along with special *control channels* between them, and the control network may be multilayered for complex systems.

We have mentioned only a few additional things to be set up over the basic, split into pieces, functionality to work as a system, but even these can make the distributed solution very complex, as shown symbolically in Figure 1.3. Many other organizational tools and their interactions may be needed for this distributed system to operate properly (like, e.g., a recovery system, which may be distributed too and deeply integrated with both local functionalities and the distributed control).

1.2.2 More Distribution Problems and Details

Distributed Algorithms May Differ. The other difficulties may be in that the distributed algorithms may differ essentially from the centralized sequential ones (in our case of Fig. 1.2*a*), and they may need updating if not full rewriting of the functional parts shown in Figure 1.2*b* (as well as, possibly, introduction of other parts) to operate in a distributed and parallel manner. To bring the organization of Figure 1.3 to life, we may need to use together quite different existing languages and

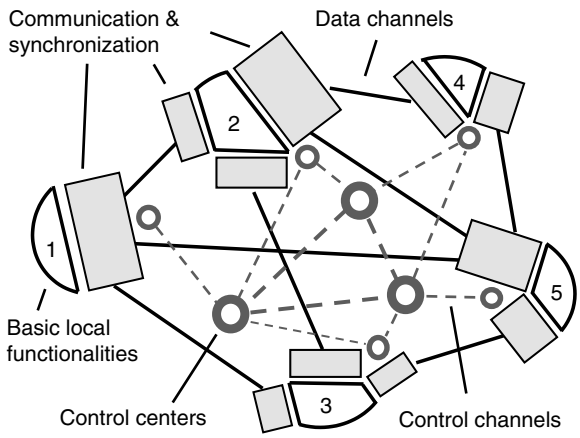

Figure 1.3. Overhead of distributed system solutions.

tools, and the whole system may become hugely overwhelmed with seams, scars, patches, and multiple interfaces.

Working in the Real World. In the real world, these functional pieces may have to acquire bodies and move physically, access and process not only information but physical matter or physical objects as well, also exchange the matter with each other along with the information, and so on. Additional control nodes and their communications with the original functional units and between them may require special (possibly, mobile) hardware units dedicated to command and control in the physical word.

Overwhelming Overhead. Creation, understanding, debugging, and subsequent managing of such distributed and dynamic systems may become a very complex problem, with the complexity growing drastically with the scale of problems to be solved. The organizational overhead, shown in Figure1.3, may considerably overwhelm the useful functionality of integral localized solutions, as in Figure 1.2*a*.

Starting from Scratch for Each Problem. And for each new problem (at least for each class of them), the whole work of creating, assembling, and debugging the complex distributed organization of Figure 1.3 must be repeated, possibly, from scratch. This is because *the local functions and their interactions, also the needed control over them, may be unique for each problem to be solved.*

Other Philosophies Needed. For solving complex problems in real, especially unpredictable and hostile, environments, we may need distributed systems organized in a very different manner than described above, with much higher clarity, flexibility, and also capability to recover after damages. This may require inventing, prototyping, and

testing of quite different organizational ideologies, methodologies, and technologies than usual, as well as a possible revision of the ruling philosophies of existence, evolution, and general behavior of large dynamic and open systems, both manned and unmanned.

1.3 WAVE-WP: BASIC IDEAS

WAVE-WP is a completely different paradigm than any existing approaches oriented on distributed computation, coordination, and control. We provide here a sketch of some of its basic ideas and features.

1.3.1 The Whole First

Any existing approaches to investigation and management of large systems are *analytical* in nature, that is, first breaking the system into self-contained pieces or *agents* (or creating them from the start), studying them in detail, and then trying to assemble from these pieces the system as a whole, *seeing this whole from the level of these pieces.*

However, when the number of agents and interactions between them are getting large enough and these vary at system run time, serious problems of providing and supporting the needed global behavior arise, with the system as a whole becoming clumsy, unpredictable, and obscured by numerous details and routines (as already shown in the previous section).

The dominant parts-to-whole approach is becoming especially inefficient when these systems must operate in rapidly changing and hostile environments, where different parts (agents) can be destroyed, with the necessity of the whole to survive and fulfill the mission objectives.

The WAVE-WP (or World Processing) model attacks the problem by starting from the opposite side: *from the whole.* It allows us to grasp and express it on a *semantic level* in a high-level spatial language, abstracting from possible parts and their interactions (which appear and make sense in the model only in proper time and places), delegating the lower system organization to an efficient automatic implementation.

This, however, is *not a centralized approach*, where this whole as some global program or schedule is associated with a certain control center driving the distributed system, but rather is the *higher level, essence* of the system organization and behavior. This essence, which we symbolically call a *spatial gestalt*, is expressed in a special parallel and distributed formalism, which is quite different from conventional terms. [The term *gestalt* usually stands for a *form, shape, pattern*, as well as *organized whole.* Gestalt psychology (Davidson et al., 1989), see also the previous book (Sapaty, 1999a), represented a revolt from the atomistic outlook on the systems, starting with the organized whole as something more than the sum of the parts into which it can be logically analyzed.]

WAVE-WP *converts the whole distributed world into a universal spatial supercomputer*, which may operate with both information and physical matter and can

include humans as elementary "processors" as well. The paradigm often enables us to program this world with the clarity and easiness we program tasks in single computers, using traditional programming languages.

1.3.2 WAVE-WP Spatial Automaton

The WAVE-WP automaton effectively inherits the integrity of traditional sequential programming over localized memory, but for working now with the real distributed world, while allowing its parallel navigation in an active pattern flow and matching mode—as a single and integral spatial process.

The automaton may start from any point of the distributed world or system to be controlled, dynamically covering its parts or the whole, and mounting a variety of parallel and distributed knowledge and control infrastructures. Implanting distributed "soul" into the system organization, the automaton may increase the system's integrity, capability of pursuing local and global goals, assessing distributed situations, making autonomous decisions, and recovering from indiscriminate damages.

In quite different applications, the automaton may need to destroy the very system it propagates through and/or operates in (or certain, e.g., malicious incursions or infrastructures in it). Many spatially cooperating or competing parallel WAVE-WP automata may evolve on the same system's body serving, say, as deliberative, reactive, and/or reflective spatial processes.

1.3.3 Implementation Basics

On the implementation layer, the WAVE-WP automaton widely uses high-level mobile cooperative program code that is self-spreading and replicating in networks and can be easily implemented on any existing software or hardware platform. As the automaton can describe direct movement and processing in the physical world, its implementation may need to involve multiple mobile hardware—with or without human participation.

A network of communicating (hardware or software) WAVE-WP language interpreters (or WI), which can be mobile if installed in manned or unmanned vehicles, should be embedded into the distributed world to be controlled, placing WIs into its most sensitive points, as shown in Figure 1.4.

During the spatial execution of system scenarios in WAVE-WP, individual interpreters can make local information and physical matter processing as well as physical movement in space. They can also partition, modify, and replicate program code, sending it to other interpreters (along with local transitional data), dynamically forming track-based distributed interpretation infrastructures.

The automaton can also exploit other systems as computational and control resources, with or without preliminary consent, that is, in a (controlled) viruslike mode. For example, existing network attacks (especially Distributed Denial of Service, or DDoS) may be considered as a possible malicious (simplified, degenerated, and distorted) implementation of the automaton.

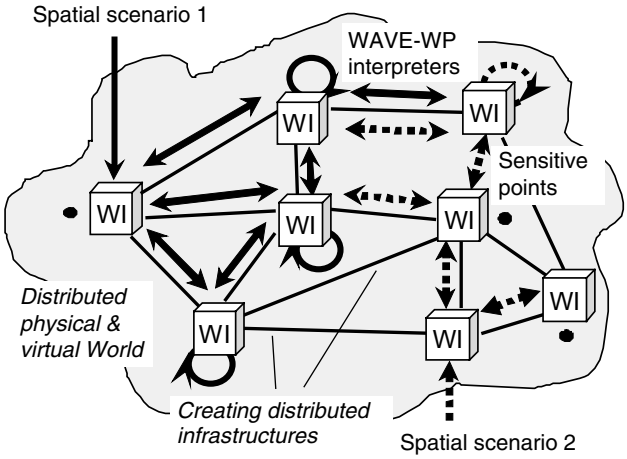

Figure 1.4. Network of interpreters with self-spreading spatial scenarios in WAVE-WP.

To explain the key WAVE-WP ideas further, let us consider the detailed expression in it of some well-known problems for which parallel and distributed solutions are of particular significance.

1.4 EXAMPLE: THE SHORTEST PATH PROBLEM

The first example relates to the finding of a shortest path between certain nodes in a network of weighed links (an example of which is shown in Fig. 1.5). This problem had been described in the previous book in relation to WAVE (Sapaty, 1999a), but due to its high practical importance and convenience in explaining the very essence of the paradigm, it is repeated in an extended form and with more details for WAVE-WP too, and within a broader context than before.

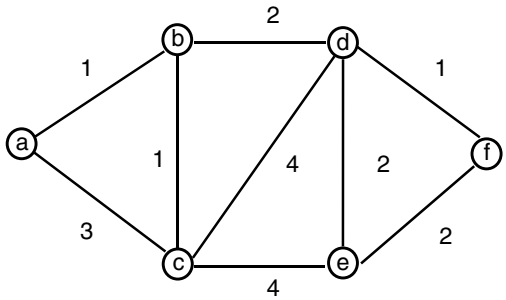

Figure 1.5. Network with weighed links.

What is shown in Figure 1.5 may be a computer network in which the link weights may define the time of sending of a standard information packet between adjacent nodes, or a road network with the weights defining distance (say, in kilometers) between the road junctions, represented by nodes. Many other important optimization problems can also be converted into finding shorted paths within different time and space continuums or their combinations.

1.4.1 Importance of Distributed and Parallel Solutions

Distributed solutions of the shortest paths may be highly important in large dynamic spaces, where communications between computers can vary at run time (caused, say, by mobility of nodes, line congestions, or as a result of virus attacks). This may also relate to road networks (with embedded local computers serving parts of them), where roads or their junctions may get damaged in natural or human-caused disasters.

Collecting all needed information in one point and then solving the shortest path problem there may not be possible, even in principle, as the network topology and weights of links may be rapidly changing over time. So the solutions must be found directly where the information resides, as the latter is inseparable from the world it represents, losing its worth when accessed remotely. And parallel solution, in addition to the distributed one, may speed up the obtaining of the result considerably.

Let us find a shortest path (which may be more than one, in general) between nodes `a` and `e` of the network of Figure 1.5.

1.4.2 Finding Shortest Path Tree

We begin with finding a shortest path tree (or SPT) starting from node `a` and covering the whole network. This tree will directly show the shortest paths from the starting node to all other nodes in the network (node `e` included), and every such path can be easily traced by just moving from each of these nodes up the tree to its root.

The most natural expression of the general solution of this problem may be as follows. By starting in node `a`, we should move to all neighboring nodes through the links to them, setting in these nodes a distance from the start equal to the weight of the passed link, also naming the start node as their predecessor in an SPT.

Then we should move from these reached nodes in a similar way to their own neighbors, carrying into them the distance from the starting node, which is the sum of the distance brought to the predecessor node and the weight of the new link passed, also naming the previous node as their predecessor in SPT. This should take place only if the distance brought is smaller than the one already associated with the nodes, thus redefining it and the predecessor in these nodes (or if these nodes have been reached for the first time), otherwise we should stop—in these nodes only.

We should continue this throughout the whole network until possible. This spatial process easily (and always) converges to a solution, as in each step we either (a) come to the nodes for first time or (b) bring into them a shorter distance than before or (c) stop—and this cannot last indefinitely. Terminating globally, we will have one of the

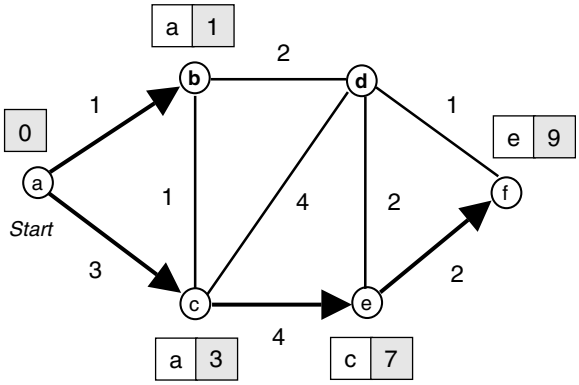

Figure 1.6. Initial SPT steps.

possible SPT with its links fixed in the network nodes by naming their final SPT predecessors.

SPT Finding Snapshots. Some initial development of the SPT finding process is shown in Figure 1.6. Starting from node `a` and propagating through its links, nodes `b` and `c` have been reached with setting distance in them from the start equal to the weights of links passed (shaded boxes) and establishing same predecessor `a` for both (clear boxes at the left of the shaded ones).

Further on, allowing arbitrary development of processes, let us assume that links from node `c` to node `e`, and then from `e` to `f` were passed, establishing corresponding distances from the start (as sums of weights of the passed links) and predecessors for the nodes reached.

An intermediate stage is shown in Figure 1.7, where distances from the start and predecessors for nodes `c`, `e`, and `f` have been redefined, as compared to Figure 1.6,

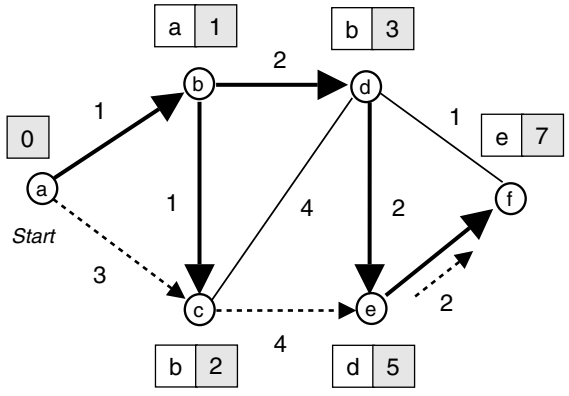

Figure 1.7. Intermediate SPT finding stage.

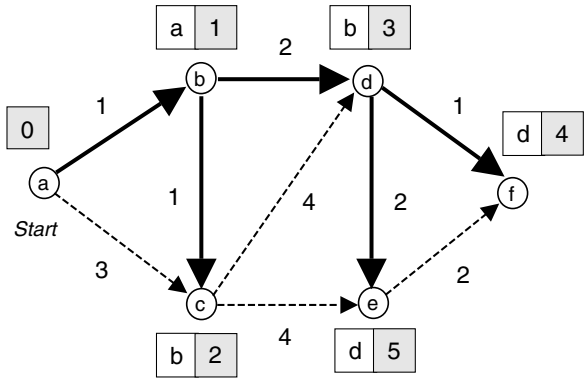

Figure 1.8. Final SPT solution.

with the previous path a–c–e–f discarded as not optimal. The link between nodes e and f was passed again, carrying shorter distance from the start and reinstating predecessor e in node f.

The final solution for the SPT (which in general may be one of the possible shortest path trees from the same start node) is shown in Figure 1.8. In comparison with Figure 1.7, an attempt to bring a shorter path to node d from node c failed, and the move from node d to node f redefined the distance and predecessor (as, correspondingly, 4 and d) in the latter for the final, optimal solution.

As follows from Figure 1.8, the resultant shortest path tree is recorded directly on the body of the network in a fully distributed form, by naming the SPT predecessor node in each of its nodes.

Spatial SPT Finding Pattern. A spatial pattern expressing the semantics, essence, of these shortest path tree finding processes, omitting any implementation details, is shown in Figure 1.9.

The pattern reveals maximum possible parallelism for solving the SPT problem, where all links from the nodes reached can be passed simultaneously, and operations in nodes do not need any synchronization with processes in other nodes that may cause delays. The pattern also characterizes an absolute distribution of SPT finding activities in the network space, and all operations it offers are purely local, being associated exclusively with the nodes reached.

A variety of concrete algorithms for sequential or parallel, as well as centralized or distributed, implementation can be designed and realized, all based on this most generic, semantic representation of the problem.

1.4.3 Collecting the Shortest Path Between Nodes

Tracing the Recorded Predecessors. To collect the shortest path from node a (from which we have already built SPT leading to all other nodes) and node e, we may

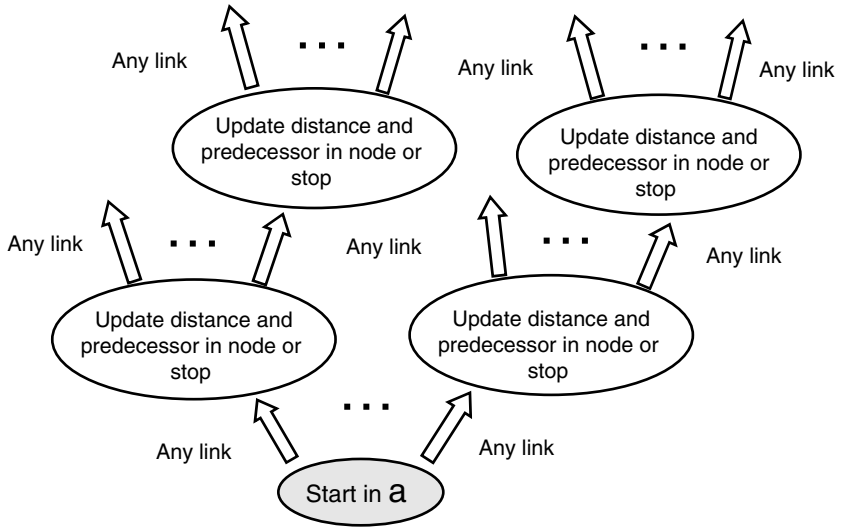

Figure 1.9. Spatial pattern of finding SPT.

start from node e and go to the neighbor node, which is its SPT predecessor, and then form the node reached to its own predecessor, and so on, unless we come into the node for which no SPT predecessor has been defined (and this will be node a) as shown in Figure 1.10.

During this movement between the network nodes by the recorded SPT predecessors in them, we can collect their names in the proper order and output the resultant path upon reaching the root of the tree, which is node a.

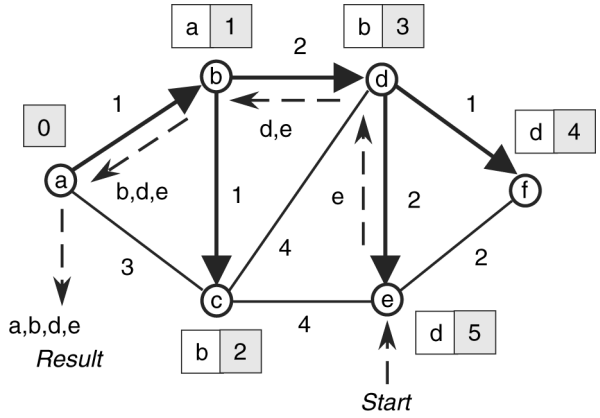

Figure 1.10. Collecting shortest path from node a to node e.

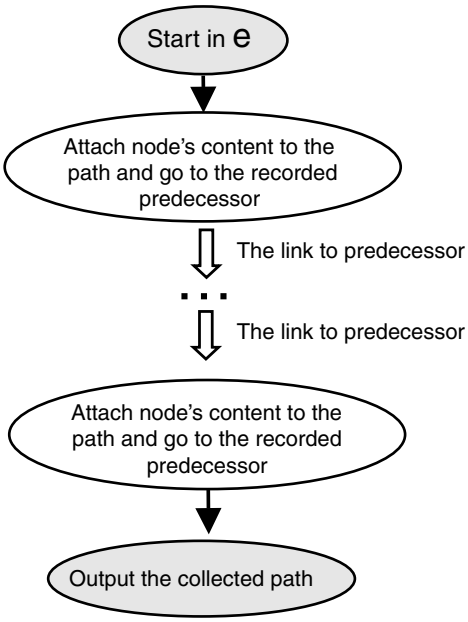

Figure 1.11. Spatial pattern for collecting the shortest path.

Shortest Path Collection Pattern. The spatial pattern expressing general semantics of this shortest path collection by moving between nodes using their prerecorded SPT predecessors is shown in Figure 1.11.

Similar to the SPT finding pattern of Figure 1.9, it may have different implementations in centralized or distributed environments too, all following, however, its sequential general nature.

Different algorithms can be possible to follow the semantics of finding shortest paths in networks, described above and expressed in the spatial patterns of Figures 1.9 and 1.11. For a sequential single-machine implementation, the first pattern has to be "sequentialized," and multidimensional arrays representing the network topology and different pointers in it can be used as data structures. In general, a single-machine implementation is trivial as all resources are in the same memory, and in view of the same program having an absolute control over the whole solution process.

1.4.4 Main Problems of Distributed Implementation

If we have a fully distributed representation of the network space of Figure 1.5, where each node may be in a different computer and links may happen to be between the computers, the shortest path solutions will be complicated drastically if traditional ideologies and technologies are used.

The following is only a fraction of what should be done in nodes of the network in addition to the operations exhibited in spatial patterns of Figures 1.9 and 1.11. Some of these service operations may be rather complex and may themselves need global supervision of the network.

- Loading the shortest path algorithm into *all nodes* of the network, checking the loading completion.
- Global search for the nodes by their names throughout the whole distributed network (here nodes `a` and `e`), starting sequentially two different distributed processes from these nodes, with assigning identities to these processes.
- Passing messages between nodes, interrupting processes for the data exchanges as these cannot be scheduled in advance in a distributed asynchronous system (the messages can come from any neighbors at any time and simultaneously from more than one neighbor).
- Broadcasting messages to all neighbors carrying distance from the start and the predecessor network address.
- Queuing messages coming simultaneously from different nodes, queuing replicated messages to be sent to successor nodes.
- Determining the overall termination of the SPT distributed process, in order to launch the path collection process, which can work only after the SPT is found.
- Informing the whole distributed network that the task is completed and the network should be cleaned up (i.e., variables for the path length and predecessor node address, still kept in all network nodes, should be removed) to be ready for execution of other distributed algorithms.

So we may be having something even more complex and ugly than what is exhibited in Figure 1.3, where parts `1` to `5` may be similar to nodes `a` to `f` of Figure 1.5, and additional control centers may need to be set up to solve the distributed coordination and management problems on different stages of the distributed shortest path solutions.

1.4.5 Universal WAVE-WP Interpreters

In WAVE-WP, all the routines listed above for distributed solutions are carried out automatically, with highest possible efficiency (as this is done on the internal system implementation, rather than application, level) by networks of the interpreters from WAVE-WP language. The interpreters may be associated with each node of the network, as shown in Figure 1.12, and reside in different computers, and the computers may have any communication infrastructure (which, e.g., can match the one of Figure 1.5 or can be quite different from it).

In WAVE-WP, the network of Figure 1.5 can also be arbitrarily distributed between the interpreters, with a possible partition between two interpreters (and two

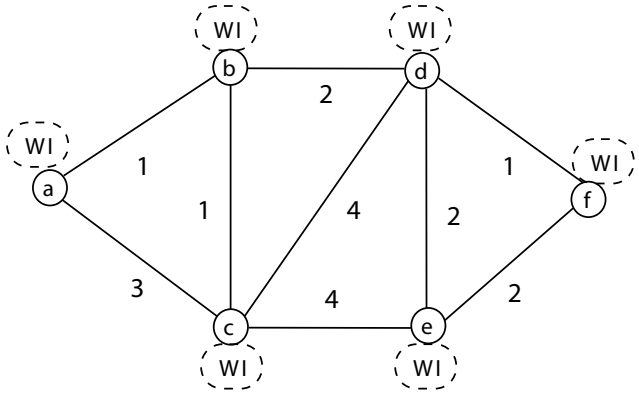

Figure 1.12. Associating the WAVE-WP interpreters with each node.

computers as well) depicted in Figure 1.13, with some links connecting nodes in the same interpreters while others being between nodes located in different interpreters (computers).

Direct Expression of the Semantics in WAVE-WP. In WAVE-WP, we can express the semantics of highly parallel and distributed solutions directly, omitting tedious system details symbolically exhibited in Figure 1.3 and also clarified above in relation to the shortest path problem.

For example, we can *straightforwardly* convert the spatial patterns of Figures 1.9 and 1.11 into active programs in WAVE-WP language dynamically self-navigating, self-replicating, and self-matching the distributed network in a pursuit of the global optimization solution, as follows.

1.4.6 Shortest Path Tree Finding in WAVE-WP

For finding a possible SPT in a network of weighed links in a maximum parallel and fully distributed mode, starting from node `a` and directly expressing the spatial pattern of Figure 1.9, the following WAVE-WP program will be sufficient:

```
direct # a;
repeat(
  any # noback; Fdistance += LINK;
  Ndistance == nil, Ndistance > Fdistance;
  Ndistance = Fdistance; Npredecessor = BACK
)
```

We will give below only some general explanation of how this program works, as the WAVE-WP language syntax and semantics are described in full details later in the book.

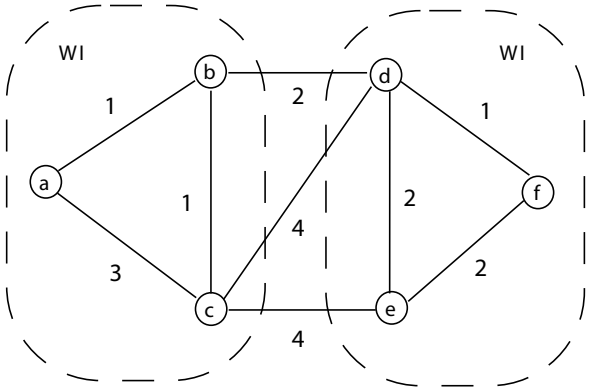

Figure 1.13. Partitioning the network between two interpreters.

The operations separated by a semicolon are executed in a sequence, while a comma separates parallel or independent parts. Symbol `#` is a hop act allowing us to move between nodes of the network, with the left operand giving information about links to pass and the right one detailing nodes to which these links should lead. This act permits both selective and broadcasting propagation in networks.

The operations of this program have the following meanings (given in the order written).

`direct # a`**.** Hops directly into node `a` of the network from an arbitrary point of the (distributed) world where this program is applied (which may also be any of the network nodes in Fig. 1.5).

`repeat(...)`**.** The `repeat` (belonging to nonlocal language constructs called *rules*) allows us to make spatial looping of the embraced part of the program, where the next iteration of this part starts in parallel from all nodes reached by the previous iteration—until such nodes exist.

`any # noback`**.** Makes hop from the current node (at the start this is node `a`) through *any* existing links *to all neighboring nodes, excluding the node it has been reached from*, using corresponding keywords as the left and right operands. The following part of the program will replicate, starting in parallel in all such nodes reached, if any.

`Fdistance += LINK`**.** The variable `Fdistance` (its name starting with `F`) belongs to the class of spatial variables called *frontal.* Being an exclusive property of the navigation program, this variable spreads and replicates during the space navigation (each program branch operating with its own copy of the variable).

`Fdistance` accumulates and carries the distance from the start node `a`, as the sum of weights of the links passed by the current branch. Having come into a node, this

variable (automatically created by the first access to it) is incremented by the content (here weight) of the link passed, which is accessible by special variable `LINK` (belonging to the class of *environmental* variables).

`Ndistance == nil, Ndistance > Fdistance`**.** These two statements, operating independently from each other, check if the variable `Ndistance` (belonging to `N`-prefixed *nodal* variables associated with network nodes) is undefined (treated as `nil`) or has the value that is greater than the one brought to the current node in variable `Fdistance`. If one of these statements results with `true`, the following part of the program will work. Otherwise the program in this particular node will stop (with the node failing to be classified as *reached*), not influencing, however, its development in other branches operating in other nodes.

`Ndistance = Fdistance; Npredecessor = BACK`**.** The sequence of these two statements redefines (or sets up if undefined, i.e., if the node is visited first time) the distance from the start in nodal variable `Ndistance` by the content of `Fdistance` brought into the current node. It also redefines (or sets up) in `Npredecessor` the network address of the predecessor node (lifted by environmental variable `BACK` in the current node). Nodal variables `Ndistance` and `Npredecessor` are automatically created by the first access to them (which is common to variables of WAVE-WP).

1.4.7 Shortest Path Collection in WAVE-WP

For collecting the shortest path using the SPT predecessors recorded in each node of the network in variable `Npredecessor`, while directly expressing the pattern of Figure 1.11, the following program will be sufficient:

```
direct # e;
repeat(Fpath = CONTENT & Fpath; any # Npredecessor);
USER = Fpath
```

Similar to the previous program, we will consider its work by analyzing different operations too, in the order written, as follows.

`direct # e`**.** Will hop directly to node `e` from some initial position in the world, which may be the same one we hopped from to node `a`, or any other.

`repeat(...)`**.** This spatial loop construct works similarly to the loop of the previous program. If the current iteration terminates with failure, the rest of the program (following the whole `repeat` construct) will work from the current node (i.e., the one from which this last, failed iteration started).

`Fpath = CONTENT & Fpath`**.** This statement adds the content (same as name) of the reached node (lifted by environmental variable `CONTENT`) to the beginning of the

path being collected in frontal variable `Fpath` (the latter, automatically created on the first access to it, is moving between nodes too, accumulating the shortest path).

`any # Npredecessor`**.** Using the SPT predecessor node address recorded in nodal variable `Npredecessor` by the previous program, this statement hops to this predecessor node via any existing link to it. If no predecessor is recorded in a node (and this may only be the start node `a`), control state `fail` will result, terminating the spatial loop and also allowing the next operation to take place.

`USER = Fpath`**.** This will output the result by assigning the collected path in frontal variable `Fpath` to a symbolic user (using environmental variable `USER`), which may be any terminal inside or outside the WAVE-WP system or belonging to any other system.

1.4.8 Full Program for Finding Shortest Path

We can easily integrate the above two programs (for finding SPT from node `a` and collecting the shortest path from `a` to `e` using this SPT) within a single program starting from any point of the distributed world, as follows:

```
sequence(
  (direct # a;
  repeat(
    any # noback; Fdistance += LINK;
    Ndistance == nil, Ndistance > Fdistance;
    Ndistance = Fdistance; Npredecessor = BACK
  )
  ),
  (direct # e;
    repeat(Fpath = CONTENT & Fpath; any # Npredecessor);
    USER = Fpath
  )
)
```

Similar to the predecessor language WAVE, we can use abbreviations of the WAVE-WP language keywords (also short names of variables) to make WAVE-WP programs extremely compact and suitable for direct interpretation, without any compilation, in both software and hardware, as well as for quick spreading via communication channels between computers in a network. For the previous program, its short representation may be as follows:

```
sq(
(@#a;rp(#nb;F+=L;N==,N>F;N=F;Np=B)),
(@#e;rp(F=C&F;#Np);U=F)
)
```

Using some additional features of WAVE-WP, this program can be made even shorter, for example, as:

```
@#a;[rp(#;F+=L;N==,N>F;N=F;M=B)];
@#e;rp(F=C&F;#M);U=F
```

It essentially is (up to two, or even three, orders of magnitude!) shorter than in any other languages, being at the same time fully distributed and maximum parallel solution of the shortest path problem.

In WAVE-WP, we can express the semantics of spatial problems directly, ignoring implementation details, delegating the latter to an efficient automatic interpretation system. This is the main distinction and advantage of WAVE-WP paradigm in comparison with any other models and technologies.

1.5 EXAMPLE: DISTRIBUTED KNOWLEDGE REPRESENTATION AND PROCESSING

In the previous section, we made an attempt to explain what WAVE-WP means on an example of the important computation and optimization problem on distributed networks. Let us now try to touch another very important direction: *knowledge representation and processing*, especially parallel and distributed ones, which underlies a great variety of intelligent systems of different natures and with diverse applications.

1.5.1 Knowledge Network

The most general knowledge representation may be in the form of a knowledge network, or KN, with nodes reflecting different concepts or facts, and links being relations (oriented as well as nonoriented) between the nodes, with an example shown in Figure 1.14. In this KN, persons named `Masanori`, `Peter`, `Bob`, `John` and `Doug` are shown in their various relations (symmetric as well asymmetric, directed) to each other and also to such substances as `whiskey` and `beer`.

There are different strategies for creating any knowledge networks in parallel and fully distributed mode in WAVE-WP, inheriting these from its predecessor WAVE. Very efficient and compact ones usually based on depth-first tree templates covering the network.

Without explaining the details in this introductory presentation (which may be easily found in the previous book as well as later in this one), we are showing below how the WAVE-WP program creating the KN of Figure 1.14 in a "single breath" may look like. (The symbol `##` identifies hops to the already existing nodes, up the depth-first tree, and the language rule `create` supplies the program it embraces with a global creative power when navigating in space).

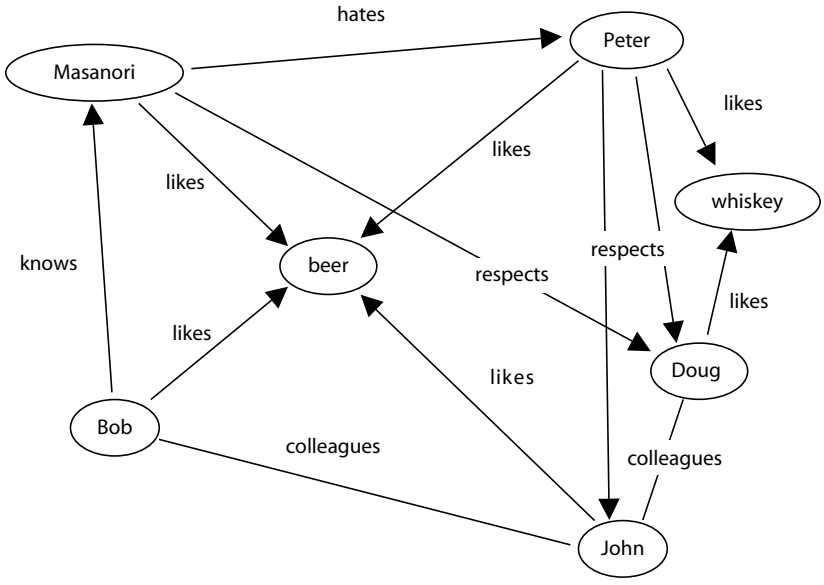

Figure 1.14. A knowledge network.

```
create(
  direct # masanori; + likes # beer; - likes # john;
  (colleagues # bob; + likes ## beer, + knows ## masanori),
  (colleagues # doug; - respects ## Masanori,
    (+ likes # whiskey; - likes # Peter;
    + respects ## (john, doug), + likes ## beer,
    - hates ## masanori)))
```

During the creation, this network may be arbitrarily (or in any other needed way) distributed between interpreters installed, say, in robots, with a possible partition shown in Figure 1.15. We can also easily supply the program with explicit hints of which hardware or software resources should be used for the implementation, or even with a detailed mapping on particular hardware (robots, in our case).

Let us show a couple of examples of elementary tasks that can be solved on this created KN in a spatial pattern-matching mode, which is effectively provided by WAVE-WP (as well as by its predecessor WAVE).

1.5.2 Elementary Query 1

"*Find everybody who likes beer*"

The spatial pattern for this request is shown in Figure 1.16, where the solution to be found corresponds to nodes named `X`.

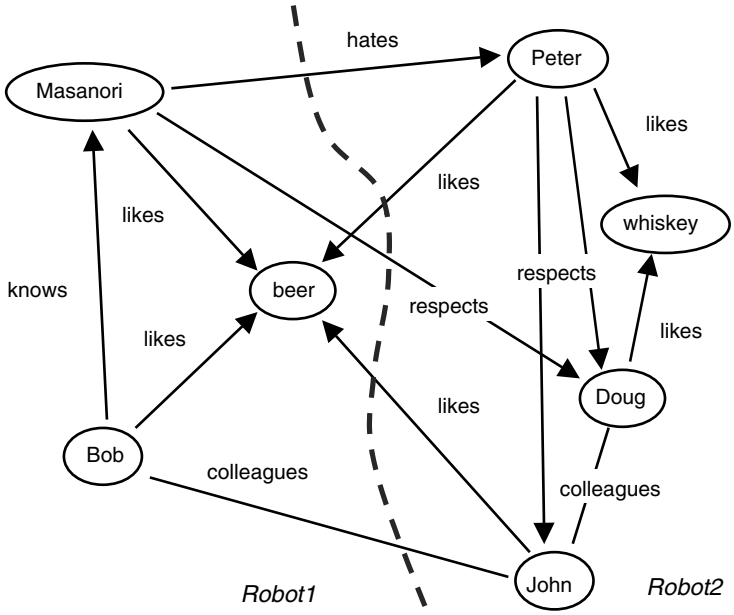

Figure 1.15. Possible network distribution between two interpreters.

In WAVE-WP, we can write a program straightforwardly following this pattern and providing on output of the contents (names) of all `X`-type nodes reached in a hop from node `beer` opposite all oriented links named `likes`.

```
USER = (direct # beer; - likes # any)
```

So in WAVE-WP the pattern of Figure 1.16 converts into a self-navigating parallel program that does all the job by itself, including the output of the (possibly, multiple) result. Starting from node `beer`, the program above finds the four resultant nodes: `bob`, `john`, `masanori`, and `peter`, as shown in Figure 1.17 (the nodes matching the pattern are shaded and matching links are shown in bold).

These nodes are found in a parallel broadcast from node `beer` and happened to reside in different robots, with `bob` and `masanori` in `Robot1`, and `john` and `peter`

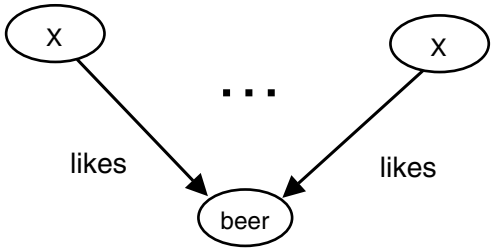

Figure 1.16. Spatial pattern for query 1.

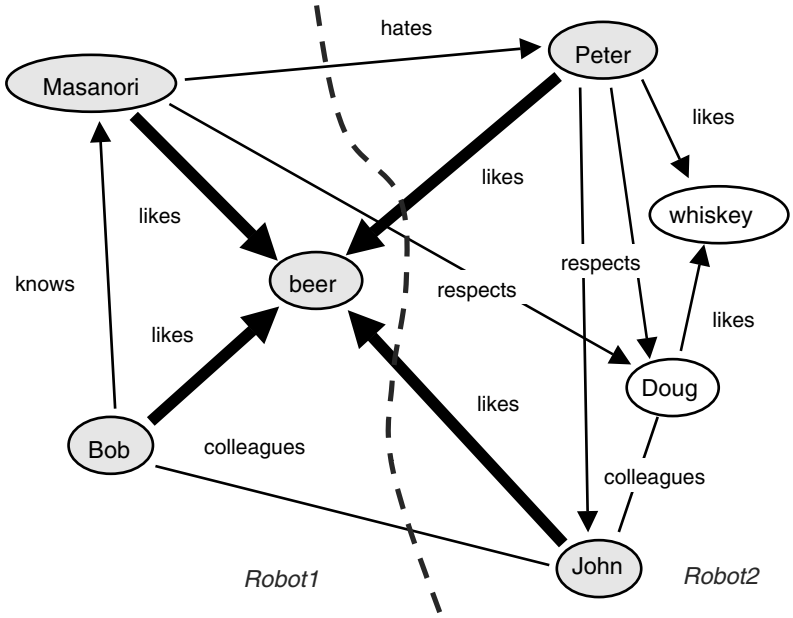

Figure 1.17. Spatial solution for query 1.

in `Robot2`, but their names are collected in one list and output in the world point where the pattern was applied, which can be any location, including one of the robots engaged.

1.5.3 Elementary Query 2

"*Who from John's colleagues is respected by both Masanori and Peter?*"

This request corresponds to the spatial pattern of Figure 1.18, which is to be matched with the KN of Figure 1.14 (or 1.15) to provide the solution needed.

This pattern can, too, be easily converted into an active WAVE-WP program solving the task in parallel and fully distributed mode, regardless of the KN distribution between interpreters (robots), as follows:

```
direct # John; colleagues # any;
and(
  andparallel(- respects # (Masanori, Peter)),
  USER = CONTENT
)
```

Starting from node `john`, the program moves from it in parallel via all links named `colleagues`, and checks independently in the nodes reached (i.e., `bob` and `doug`) whether they have oriented links `respects` coming *from both* `peter` and `masanori`

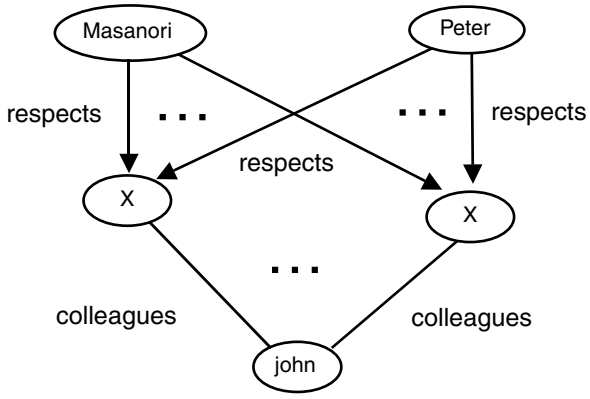

Figure 1.18. Spatial pattern for query 2.

nodes. The program provides the final single answer `doug` found in `Robot2`, as shown in Figure 1.19, which is copied and sent outside (using `CONTENT` and `USER` environmental variables).

The program uses `and` as well as `andparallel` (parallel and) rules of the language providing the needed spatial logic in finding the distributed solution. The statement:

```
andparallel(- respects # (Masanori, Peter))
```

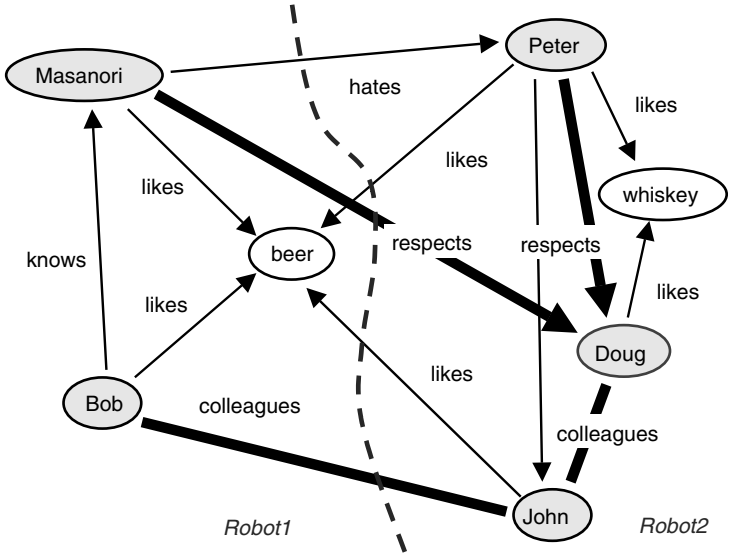

Figure 1.19. Spatial solution for query 2.

will be automatically (due to the WAVE-WP language syntax and semantics) split into:

```
andparallel(- respects # Masanori, - respects # Peter)
```

The whole program works regardless of the KN distribution between the interpreters (robots), which may vary from associating of a separate copy of the interpreter with each KN node (like what we have shown for the network in Fig. 1.12), to keeping the whole KN within the single interpreter, thus degenerating into a single-machine solution, as usual.

1.6 SYSTEM ORGANIZATION AS A FUNCTION OF THE APPLICATION SCENARIO

As is demonstrated by the above programming examples, WAVE-WP allows us to express the semantics of distributed problems straightforwardly, in a very compact and clear way, abstracting from the system implementation details. At the same time, the space-navigating algorithms in WAVE-WP are highly parallel and maximum distributed in nature, requiring conceptually no central resources, thus providing the impetus for most efficient implementations in distributed environments.

The overall system organization, similar to what has been shown in Figure 1.3, may not be predetermined in advance in our case, but may rather represent a *function of the application scenario*, the latter written on the layer well above the traditional partitioning into system components and their communications.

Materialization of the system executing these high-level spatial scenarios may be provided at run time and may depend on the current external environment in which these scenarios operate, as well as on the currently available software and hardware resources. The automatic implementation may support the integrity of these scenarios and their ability to behave as a whole under any circumstances, despite failures or damages of the system components.

Having radically raised the distributed problem description level, we have effectively shifted the extremely complex and tedious routines of the internal system organization and overall management and control to the efficient automatic layer, as will be clear from of rest of this book.

1.7 RELATION TO THE PREVIOUS BOOK

In the previous book (Sapaty, 1999a) a new distributed processing model, WAVE, was introduced. It allowed us to dynamically create virtual, or knowledge, networks in distributed computer networks and process them in parallel by multiple navigating processes.

Being just a formal spatial graph machine, with graphs universally representing any knowledge and systems, WAVE was capable of solving complex problems in

computer networks from many different classes. The solutions were integral, seamless, and compact due to higher level programming in WAVE, with traditional distributed control and management routines performed automatically on the language interpretation level.

In WAVE-WP, along with more advanced knowledge processing in open computer networks, which is also done with the help of knowledge networks, we can move directly in a continuous physical world and organize multiple activities there, including physical matter (or objects) processing. The implementation can be performed fully automatically too, by multiple mobile manned or unmanned hardware.

By using a similar to WAVE spatial automaton matching now continuous physical worlds too, this enables us to cover much broader and very important areas (practically all) of working in real environments, for both humans and robots. This includes, for instance, organization and management of human collectives, cooperative behavior of groups of robots (up to robotic armies), as well as cooperation and symbiosis between humans and robots in the advanced society—all as a derivative of high-level scenarios expressed in WAVE-WP.

1.8 COMPARISON WITH OTHER WORKS IN RELATED AREAS

For the 5 years after the publication of the first book on WAVE (Sapaty, 1999a), this new paradigm, having evolved now into a much broader and more powerful WAVE-WP, still remains unique, neither matching other approaches nor falling into other categories or classes. It, however, has features and applications that may intersect with other areas. Some possible links are outlined below.

1.8.1 Parallel Computing

WAVE-WP strongly relates to parallel computing [see, e.g., Gupta et al. (2003) and Dongarra et al. (2002)] as it allows spatial processes to develop with maximum freedom and minimum ordering, revealing all potential parallelism for any implementation. The spatial algorithms in WAVE-WP often resemble the spread of smog or flooding, and they are often more natural, simple, and compact than the related ones programmed in conventional languages, Java, for example. Of course, there may be cases where spatial constraints like synchronization and ordering are vital, and WAVE-WP has very efficient spatial mechanisms for providing these, but only if absolutely necessary.

1.8.2 Distributed Systems and Distributed Computing

WAVE-WP fully orients on distributed systems and distributed computing (Attiya, 2004; Bruce and Dempsey, 1996; Tanenbaum, 2002). Representing a completely different paradigm from conventional terms, it allows us to design fully distributed algorithms where any elementary operation and any locus of control are always associated with local points in distributed physical or virtual spaces. This brings the

distributed computing (processing) philosophy to its extreme, keeping at the same time the highest integrity of distributed solutions comparable to single-machine implementations.

1.8.3 Parallel and Distributed Computing

Integration of the ideas of parallel and distributed computing, especially with the efficient use of distributed computer networks, and for such important applications as, say, distributed interactive simulation (Hughes, et al., 2003; Fujimoto, 1999), is of growing interest and importance. In WAVE-WP these two directions integrate most naturally and are inseparable from each other, as parallelism naturally appears during the code replication and free spreading in a distributed world.

1.8.4 Computer Networking

Computer networking (Comer, 2000; Tanenbaum, 2002) is another strong link with WAVE-WP, as originally this paradigm, through WAVE, was designed to work in distributed and open network spaces and was effectively used for distributed network management; see Sapaty (1999a). Any network algorithms and protocols can be expressed in WAVE-WP as in the universal spatial automaton. Fighting crisis situations in computer networks, especially withstanding virus attacks, is one of the promising applications of WAVE-WP, to be discussed later in the book.

1.8.5 Intelligent Agents

Design of intelligent agents and efficient composition of complex systems from them (Bigus et al., 2001; Wooldridge, 2002) is one of the hottest topics in computer science and artificial intelligence. In WAVE-WP, a copy of the language interpreter, which can communicate with other such copies, is actually a universal programmable agent, and a static or dynamic network of such interpreters forms a spatial machine capable of executing any distributed algorithms represented in WAVE-WP.

1.8.6 Mobile Agents

Mobile agents (Earnshaw and Vince, 2002; Lange and Oshima, 1998; Milojicic et al., 1999; Siegwart et al., 2004) allow us to move through the distributed network and perform operations and collect data remotely. WAVE-WP, too, allows us to navigate distributed systems and execute operations where data resides, but this is organized on a much higher level than traditional mobile agents, without dividing the program into agents and their interactions.

Such things, equivalent to traditional mobile agents and also advanced distributed command and control over them (even unthinkable to be done using traditional mobile agents) are done exclusively on the implementation layer, allowing us to have application programs hundreds of times simpler and shorter that if written on the level of mobile agents directly.

1.8.7 Grid Computing

The grid computing (Carey, 1997; Abbas, 2004; Foster and Kesselman, 2003; Joseph and Fellenstein, 2003), inheriting the idea from electrical grids, tries to organize a shared use of all distributed computational resources, not only distributed information. WAVE-WP, providing this too but much cheaper and also more efficient, allows us to set up a higher level of the system organization. It converts the whole world into a powerful spatial machine that may be effectively driven by a single spatial program, dynamically covering the whole computerized world, if needed.

Embedding WAVE-WP interpreters into sensitive points of the artificial or natural distributed systems, we may obtain a unique possibility to coordinate, or *rule*, these systems in any needed way, with grid computing being years away from these, already existing, global possibilities.

1.8.8 Spatial Programming

The term *spatial programming* has been recently used for programming computers embedded into the physical world (Borcea et al., 2004; Iftode et al., 2003). Separately from this, *spatial* also relates to handling information with a geographical element (Wood, 2002). WAVE-WP, supporting all these features too, also allows us to program arbitrary complex operations in physical, virtual, or combined worlds *directly*, abstracting from computers and their networks. We also say that WAVE-WP programs are "spatial" in nature, meaning they *exist and evolve in space* (like, say, holograms) rather than in particular points. Throughout this book, we will be using the term in this broader sense, as a key feature of the programming philosophy.

1.8.9 Mobile Robotics, Cooperative Robotics

Allowing us to move in physical worlds directly and process both information and physical matter there, WAVE-WP, unlike WAVE has become very close to the topic of mobile robots (Braunl, 2003; Dudek and Jenkin, 2000; Holland, 2003) on the implementation level, where the language interpreters can be placed on top of the robotic functionality. And their ability to form a universal distributed computer driven by high-level scenarios in the spatial language allows us to cover with this philosophy any distributed multirobot systems (Asama et al., 2002; Butenko et al., 2003; Parker et al., 2000) and their collective or cooperative behavior.

1.8.10 System Management

Complex system management (Laudon et al., 2003; Wheelen et al., 2003) is one of the main applications of distributed WAVE-WP control model and technology, and the rest of this book is mainly devoted to the organization and management of a variety of large dynamic and open systems.

1.9 ORGANIZATION OF THE BOOK

In Chapter 2, key features of the WAVE-WP model are considered with examples and illustrations. Three distributed worlds can be integrated within the same formalism: continuous physical world, discrete virtual world, and the world of communicating doers, or execution world. The united world can be navigated by parallel spatial programs, *waves*, which can establish any authority over it.

In Chapter 3, the WAVE-WP language is described. Detailed syntax and semantics of the language are presented along with elementary programming examples explaining them. WAVE-WP operates on values that may represent information, physical matter, or any combination of the two. It is a higher level language for spatial processing and control, which can directly access any points of distributed worlds, process local or remote data, as well as assign local or remote results to local or remote variables.

In Chapter 4, we consider the peculiarities of distributed interpretation of WAVE-WP language by dynamic networks of "doers" represented by the language interpreters embedded in communicating vehicles of any origin. The solutions discussed can allow spatial WAVE-WP scenarios to evolve in various environments and set up advanced command and control over other systems, with automatic decision making at any levels.

Chapter 5 contains examples of the practical use of WAVE-WP language ranging from traditional programming to expressing higher level abstractions and cooperative actions in the united physical–virtual space, with implementation in computer networks and multiple mobile robots. Included are elements of the new agent-free methodology of distributed and parallel programming of dynamic systems in a spatial pattern-matching mode.

In Chapter 6, an expression in WAVE-WP of a number of exemplary mission scenarios in a distributed physical world is considered. Different forms of movement of organized groups, including the movement with physical payload, are presented. Distributed space search and processing are discussed too. Optimized solutions can be found in the virtual world first in a look-ahead simulation mode, and subsequently used for guiding parallel movement and operations in the physical world.

In Chapter 7, we consider the run time creation and modification of different kinds of distributed command-and-control infrastructures for mobile dynamic systems, with orientation on crisis management, and also solving some basic control and management problems in a spatial mode, navigating these infrastructures in parallel. Rapid composition of an effective crisis reaction force and providing its high operability in hostile environments are extremely important problems to be solved.

Chapter 8, as a continuation of Chapter 7, concentrates on a variety of dynamic situations and scenarios that may take place in multirobot, computer network, and united global systems, especially with the relation to advanced crisis management. All these are investigated and expressed using the WAVE-WP language. Distributed cognitive systems, robotized hospitals, future combat systems, and fighting virus attacks in computer networks are among the topics and problems discussed, with examples of distributed solutions presented.

Chapter 9 concludes the book, summarizing main features of the WAVE-WP model and technology, also highlighting a number of its most important applications at present as well as revealing future plans.

The Appendix contains full WAVE-WP language syntax in both extended and compact forms, as well as main abbreviations of the language keywords allowing us to make programs very compact.

The References list main publications on the WAVE-WP model (Sapaty, 1999b; 2000–2004) and also relate to further practical applications of the previous model WAVE (Gonzalez-Valenzuela and Leung, 2002; Gonzalez-Valenzuela and Vuong, 2002; Gonzalez-Valenzuela et al., 2001) and its implementation (Borst, 2002), which appeared after the first book was published. Other works in the related areas are included too.