Abstract

A universal spatial automaton, called WAVE, for highly parallel processing in arbitrary distributed systems is described. The automaton is based on a virus principle where recursive programs, or waves, self-navigate in networks of data or processes in multiple cooperative parts while controlling and modifying the environment they exist in and move through. The layered general organisation of the automaton as well as its distributed implementation in computer networks have been discussed. As the automaton dynamically creates, modifies, activates and processes any knowledge networks arbitrarily distributed in computer networks, it can easily model any other paradigms for parallel and distributed computing. Comparison of WAVE with some known programming models and languages, and ideas of their possible integration have also been given.

1. Introduction

While traditional distributed programming models are based on stationary programs exchanging data by sending messages, i.e. on data flow or data mobility, WAVE represents a quite opposite paradigm [16, 17, 20]. Originating from a practical implementation of mobile control migrating in heterogeneous computer networks and organising intercomputer dialogue [15], which was much easier to implement than a global management system, WAVE is based on program mobility while data is rather a stationary world which these programs navigate. In WAVE, special recursive programs (or waves) spread themselves through other systems which they consider as data. While moving, waves may be dynamically self-replicated, split into pieces and modified. Waves propagate in a distributed network space operating with variables which access only local data in current nodes or data items which are transferred with the moving wave code.

The WAVE language, a core of this model, describes parallel propagation through a distributed knowledge network where both nodes and links may hold arbitrary information (which can include procedures to be executed). The whole process is described as a sequential-parallel composition of elementary actions, or moves, which can include hops through data links (permitting also broadcasting and multicasting), assignments to variables, and condition checking filters, all these alternatives having the same rank. During the navigation process, waves may create or modify the very network they move through.

WAVE has navigational, not message passing, semantics. Explicit message passing is used only on the implementation level where communicating interpreters execute heads of moving waves while sending their tails and intermediate data, as messages, to other interpreters. This results in a drastic simplification of application programs (usually being one-two orders of magnitude shorter than in traditional programming languages) as the main synchronisation and data communication routines are hidden inside the implementation of the language. From WAVE, communication with other systems resident on the same hosts is possible.

WAVE language has been successfully used for solving complex problems in distributed systems including theoretical graph and network problems, integration of distributed databases, distributed simulation of dynamic systems, modelling of collective behaviour of robots, intelligent management of computer, telecommunication and transport networks, design of intelligent infrastructures for distributed federations, as well as for distributed dynamic 3D virtual reality [1, 2, 6, 7, 8, 9, 12, 13, 19, 21, 23].

In this paper, main features of the WAVE model, as well as the gained experience of its implementation and use in a variety of dynamic and distributed applications, are summarised by representing WAVE as a new type of a universal parallel computational automaton. This automaton is capable of solving any problems in arbitrary computer network topologies in a program flow and pattern-matching modes of operation, without any central resources.

The rest of the paper is organised as follows. Chapter 2 provides examples of programming in
WAVE. In Chapter 3 a brief description of the WAVE language is given. Chapter 4 describes a layered organisation of the spatial WAVE automaton, functions performed by each layer, interactions between layers, as well as a general implementation architecture of the WAVE interpreter. In Chapter 5, a comparison of WAVE with some other models and languages is provided, with giving hints on their possible integration. Chapter 6 concludes the paper.

2. Mobile programming in WAVE

WAVE directly processes knowledge networks (KN) consisting of nodes and links (oriented as well as non oriented) connecting them, with any information associated with the both. Such networks may be arbitrarily distributed between processors, and the computer network topology may correspond to or be quite different from the knowledge network topology.

2.1. Elementary example: following a path

Let us consider an elementary program which, starting from node "a" in the network of Fig. 1, follows a route consisting of links named first "p" and then "q", and prints a name of the node reached. Its usual, verbal, description may be as:

Start in "a"
Hop through "p"
Hop through "q"
Print name of the current node

A corresponding WAVE program is shown in Fig. 1 which, after being applied in "a", moves through the network while incrementally interpreted in nodes with losing its worked parts. In this program "#" is a hop operator whose left operand names links to be passed and the right one identifies nodes these links should lead to (their absence makes any links or nodes allowable).

"@" means an associative (or "tunnel") link to a node from an outside of the system or between any two nodes, not necessarily neighbouring. C is an "environmental" variable always lifting a content (name) of a node in which wave currently resides. T means terminal (special read–write variable) accessible from a current node (may not be the same in different network nodes). Period delimits operations which should succeed one another while executed in a network.

This program after node "d" splits into two copies, along each "q" link, while bringing replicated operation "T=C" into "e" and "f" which prints node names in parallel. To restrict this program to a single solution, say "e" as a destination, both the link and node operands must be present in the last hop: @#a.p#.q#.e.T=C.

2.2. Collecting names of all network nodes

Let us consider another program which collects names of all network nodes into one list in parallel and prints this list in a starting node, let it again be "a". A usual description of a recursive breadth-first search & collection algorithm for this may look like:

Define MOVING_PROCEDURE as:

Hop to all neighbours
If node is not marked
put its name into NODAL_LIST
otherwise halt this branch
Do sequentially the steps
Step1: Apply MOVING_PROCEDURE
Step2: Copy NODAL_LIST into MOVING_LIST
Hop to predecessor
Append MOVING_LIST into NODAL_LIST
end MOVING_PROCEDURE

Start in "a"
Put node's name into NODAL_LIST
Do sequentially
Apply MOVING_PROCEDURE
Print NODAL_LIST

The corresponding WAVE program and its parallel development in the network are shown in Fig. 2, where the dynamically created breadth-first spanning tree is depicted in bold. This tree is subsequently used for collecting node names and merging partial lists (in a logarithmic total time) until the final list appears in the starting node "a", which is then printed.
wave code), a mere naming of which (without any operation) causes an injection and immediate execution of the corresponding code as part of the wave. "&" is a list append operation (with the result recorded on the left operand, being a variable). P is an environmental variable always giving address of the predecessor node from which the current node has been reached.

N is a stationary, or "nodal", variable copies of which are associated with different network nodes, "==" means "equal to" (its missing right operand means "nothing", not defined). Fr is another moving, frontal variable, and SQ is a "sequence" control rule activating two program branches (separated by comma) sequentially, from the same node (where SQ is interpreted), with the second branch starting only after a complete termination of the (recursive) first branch.

Only movements of the resultant partial lists are shown in Fig. 2, with the copied SQ rule dynamically appearing in network nodes (symbolised by loops). As can be seen from this simple program, waves are creating cooperative mobile program societies which spread operations, control and local data in a distributed space behaving altogether as a fully controlled system.

2.3. Creating network topology

Arbitrary knowledge networks may not only be processed by mobile waves but also created within the same WAVE language syntax. In Fig.3.a the creation of a simple network is shown where wave template is embraced by a CR ("create") rule which allows to create a distributed topology in "one breath", without repeating CR for each element. The CR-rule is inherited when waves replicate and reduce while unwrapping in space. The language interpreter will automatically distribute this network during its creation between different processors, according to some general recommendations given to the system (for example, "put each new node to a new processor").

To make an exact distribution, each new creative hop should be preceded by moving to the corresponding processor by its address, with saving nodal addresses (environmental variable A), if needed, in frontal variables, like in Fig.3.b. Here, the interpreter will be recommended to place as many nodes as possible into the same processor (thus putting "a" and "b" together).

To create and distribute the more complex network from Fig.1, the following wave, based on a depth-first spanning tree template, will be sufficient (starting from "d" and using moving variable F for saving address of "d" for implementing cycles, and not replicating "d"):

\[
\text{CR}@d.F=A.(p#a.(k#b,k#F),(m#c,n#F)),(q#e,r#f,q#F))
\]

The CR rule influences only hops with both link and node names (not addresses) explicitly given in the wave it controls, whereas other operations remain unaffected and are performed as in a usual program.

The simple examples described above explain some peculiarities of the mobile programming in WAVE. Syntactically WAVE is a very simple language but, due to its recursive structure, it allows for the expression of arbitrary complex (parallel and fully distributed) algorithms in a very dense form. Waves, being creative & navigative templates rather than the traditional programs, are spreading like physical waves or viruses in distributed systems.

3. WAVE language

The WAVE language describes propagation (parallel and asynchronous) through a distributed, network-structured data continuum rather than traditional data processing. It has a concise syntax and rich navigational semantics and is a machine-level language intended to be suitable for physical movement and direct hardware interpretation in computer networks.

3.1. General organisation

The syntactic structure of waves is shown in Fig.4 where braces mean zero or more repetitions (with a given delimiter at the right, if more than one), square brackets denote an optional construct, and vertical bar separates alternatives. Period delimits sequential parts and comma separates independent or parallel parts of a wave, called moves.

\[
\text{wave} \rightarrow \{ \{ \text{move} \}, \} \]
\[
\text{move} \rightarrow \text{unit} \{ \text{act unit} \} | [ \text{rule} ] (\text{wave})
\]

Figure 4. Recursive syntax of WAVE.

Moves may be simple, consisting of one or more elementary operations, or acts (like assignments, hops, condition checking filters, etc.) over information units, or be again waves (in parentheses) optionally prefixed by control rules. The latter impose a variety of constraints over distributed development of waves in KN. Starting from some (a current) node, a move brings the wave into a new set of current KN nodes, or Goal Set (GS) which
may include the initial node, to all of which the tail of the wave is applied, waves thus being interpreted incrementally in KN. In general, many waves within the same program (or the different ones) may spread in KN in an asynchronous wavefront mode.

Many self-evolving wave processes may start independently in the shared multiprocessor space and from different sources, as shown in Fig. 5, and may be independent or interact with each other. The spatial processes may cover in parallel any parts of the space. The shapes of asynchronous wavefronts producing new GSi by moving waves may be arbitrary as the waves may represent any algorithms.

![Distributed Knowledge Network](image)

**Figure 5. Spreading waves in a knowledge space.**

### 3.2. Basic information unit

The basic information unit of the language is vector—a dynamic sequence of arbitrary length values generally defined as strings (syntactically separated by a semicolon), concrete interpretation of which depends on the operations involved. This simple data structure with special operations on it, together with the recursive control syntax of the language, is sufficient for representing arbitrary network creation and processing algorithms in a distributed environment. No explicit type descriptions are used in the language: automatic type conversions are activated depending on the current operations involved.

### 3.3. Spatial variables

Information units in a WAVE program can also be expressed by **spatial variables** dynamically distributed throughout KN and being of the three types: **nodal** (prefixed by N) dynamically attached to KN nodes and shared by different moving waves, **frontal** (prefixed by F) moving with the language strings, and **environmental**, accessing currently available resources relating to KN nodes and links. The latter are named as: C - node content, A - node address, L - incoming link content, S - incoming link sign, P - predecessor address, and T - user terminal (or one of them if they are distributed throughout KN). There are also special environmental variables enabling to control efficiency on the implementation level: D - a list of addresses of direct computer neighbours, and V - a threshold of volume of data allowed to be stored in the current processor (in a number of nodes). Frontal variables accompany the mobile wave and carry out local information exchanges between different nodes of KN.

### 3.4. Acts

Basic acts are selective or broadcasting **hops** in KN, condition checking **filters** (halting if FALSE), **data processing** (arithmetic and string operations), explicit **halts** with a repertoire of echoing conditions, and an **external call** permitting an access and exchange of information with other systems distributed in networks. Hops (the "#" act) identify by the left operand the links to be passed, and by the right operand the nodes these links should lead to (nodes are identified by contents or addresses). Omitting the right operand makes any destination nodes acceptable with the given links. Omitting the left operand leads to a neglect of the link contents and broadcasting to certain (if names are provided) or to all neighbours. The special name "@" used as the link operand triggers direct (tunnel) jumps between any (including non-neighbouring) nodes, and makes broadcasting to all other nodes of KN if the right operand is empty. If more than one link named in the hop is associated with the node, all these may be passed in parallel.

Filters ("==" - equal, "/=" - not equal, "+" - less, "+=" - less or equal”) allow for the further wave propagation if their result is TRUE, and halt if FALSE. Data processing includes arithmetic acts ("+", "-", "/", "/"), splitting string into a vector ("|") and merging vector into a string ("%") with the given delimiters, appending vectors ("&"), finding/recording a content by an index ("."), finding an index by a content or recording by a content ("::"). Act "?" makes an access to other systems on the host (via its operating system) and "/!" is a programmed halt with the operands establishing different halting conditions (right hand operand), or switching off the track mechanism for launching uncontrolled waves with establishing their life time (by the left operand).

Act "/=" means a mere assignment of the result obtained on the right to the variable on the left. In its absence, the result of the data processing operations is assigned to the leftmost unit (ought to be a variable). For example, N=N+F-1 is equivalent to : N+F-1, thus making expressions more compact.

### 3.5. Rules

Main rules and their abbreviations are: SeQuence (SQ), Or Sequential (OS), Or Parallel (OP), And Sequential (AS), And Parallel (AP), RePetition (RP), WaiTing (WT), InDivisible (ID), and CReate (CR). The
rules split waves into branches and coordinate their cooperative (parallel or sequential) development in the KN (SQ, OS, OP, AS, AP), provide distributed logical synchronisation (WT) and indivisible access to shared resources (ID), repeated application of the wave (RP), enable the wave to create or modify the KN it moves through (CR). The control points triggered by rules dynamically appear in different KN nodes and make distributed coordination of the propagating waves to a proper depth, using the tracks mechanism. These points cease to exist after termination of the controlled waves.

3.6. Dynamic code injection

It is possible to inject new strings into the moving wave as procedures (kept and processed as string contents of variables) which accompany the waves (in frontal variables) or are picked up in nodes of KN during navigation of the latter (nodal and environmental variables). This provides flexibility in creative and navigative network processes where the evolving spatial program may be additionally fed from the distributed environment it moves through. Syntactically this is expressed by a move consisting of a single unit (a variable), without any act, which causes injection of its content into the wave with immediate execution. Such dynamic code injection may be recursive.

4. The WAVE automaton

In this chapter we will give an informal presentation of the WAVE model as a spatial automaton processing arbitrary network topologies in a parallel mode. A layered structure of this automaton will be outlined with the main interactions between different layers discussed.

4.1. Layered organisation of the automaton

The WAVE automaton has a four-layer organisation depicted in Fig. 6 with the layers having the following meanings.

- Mobile waves layer
- Dynamic tracks layer
- Knowledge network layer
- Computer network layer

Figure 6. Layered organisation of WAVE.

1. Computer network layer. The lowest, or Computer Network (CN), layer may have any number of computers and arbitrary interconnection topology. Each computer in the network must have a unique address. These addresses may be used for sending messages between any two computers by using standard communication facilities (say Internet) supposed to be in a regular service on the CN layer and which are not specified in the automaton.

2. Abstract knowledge network layer. Knowledge network (KN) layer reflects structuring of information within an application area and may have any topology. KN consists of nodes and links and may be arbitrarily distributed between computers of CN, where each computer may have zero or more nodes allocated to it. Links of KN may therefore connect nodes within the same or between different computers. Both nodes and links of KN, being arbitrary strings of characters, may associate with any information, as was already mentioned above. KN nodes have absolute addresses in space consisting of the two parts: a physical address of the computer they reside in and an address within the memory of the computer.

3. Dynamic tracks layer. Tracks accompany and support the spreading and coordination of distributed processes in KN. Starting in different nodes of KN they grow as trees. Track nodes dynamically match KN nodes (zero or more track nodes may be associated with the same KN node), while track links match KN links through which the processes evolve. Track links may also reflect direct, or “tunnel”, hops between non-neighbouring KN nodes. Different track trees, starting independently in the same or different KN nodes, may overlap in its structure. The track layer serves as a special dynamic control infrastructure while executing mobile algorithms in KN. Tracks are used for generalising states (like success or failure) of multiple distributed processes by using different echoes which are backwarded via track links and merged in track nodes. Tracks also serve as channels for further spreading of suspended mobile processes (wave tails), till proper conditions are met, and support life time of temporary variables dynamically associated with KN nodes.

4. Mobile waves layer. Recursively organised mobile programs, or waves, are on the top organisational level. They have navigational, rather than the traditional reductional (for functional models) or message passing (for communicating processes) semantics, and solve all problems in a KN by an incremental matching of its topology, while propagating through it. During propagation, waves may self-replicate, split into parallel branches and modify, losing the worked parts. They can transfer with them local data while leaving other data in KN nodes to be shared with other waves, as has already been specified in the WAVE language description.

4.2. Interactions between layers

There are regular communications between any two
neighbouring layers, as well as a possibility of direct interactions between any non-adjacent layers. The main cases of this activity are as follows.

Waves layer — tracks layer. All activity of the WAVE automaton is initiated from the waves layer by injecting waves into one or a number of KN nodes, the waves subsequently self-propagate in KN. Moving waves access and change only local information when staying in KN nodes. Any hop between KN nodes and any split of a wave into branches is accompanied by extending of tracks. Rules coordinating sequential or parallel invocation of branches become temporarily associated with certain track nodes. The rules use echoes coming back via tracks to assess success or failure of the whole wave branch at the root. The rules forward the suspended wave tails to the fringe track nodes (associated with the certain KN nodes) from which they develop further. All tracks are automatically removed when the wave program echoes complete termination via its track tree. Track branches may also be deleted at runtime when the corresponding waves fail or execute special halts, while other branches may continue.

Waves layer — KN layer. Initially, waves directly access KN layer, which is accompanied by creating tracks as a service layer between suspended waves and the KN topology. These tracks subsequently serve as bridges for further spreading "waves" of waves. Waves can make "surface" hops in a KN via its links, or "tunnel" hops (directly between any, including non-adjacent, nodes) by their addresses or contents. Waves can also stay in KN nodes arbitrary long time while performing any sequences of operations over local data.

Waves layer — CN layer. Waves may not only process existing KN topologies but may also create and modify them while navigating the computer network directly. Creation/modification of KN and computations on it may be done simultaneously by the same waves.

Track layer — KN layer. Tracks, evolving in a KN topology, are matching the latter, and their existence is generally connected with the existence of the corresponding elements of KN. Tracks, however, have a certain degree of autonomy and may remain alive while the corresponding KN nodes and links are removed by other waves, thus preserving the continuity of a distributed control in KN.

KN layer — CN layer. Distribution of KN in a computer network may be explicit by waves, by first hopping to particular computers and then creating KN nodes in them and links to the predecessor nodes, or implicit, where KN nodes and links are created and spread between computers automatically. The latter uses a special threshold parameter (V) accessible from the waves layer and establishing a maximum number of KN nodes allowed to be allocated in different computers.

As can be seen, the waves layer, tracks layer and knowledge network layer have a flexible cooperative organisation which dynamically creates and supports distributed knowledge structures and the processes evolving on them within arbitrary computer networks. The latter may also be dynamic, with the number of computers and their interconnection topologies changing at runtime.

4.3. The WAVE interpreter architecture

The WAVE automaton has been implemented as a direct interpreter of waves operating in arbitrary computer networks [5, 18]. It dynamically supports all the four layers and their interactions described. A copy of this interpreter must be installed in each computer, the interpreters may communicate with the neighbouring interpreters, thus forming a distributed machine driven by mobile waves.

The interpreter (see Fig. 7) consists of incoming and outgoing queues for exchanging waves and echoes with other interpreters, and the three main specialised functional units: parser, data processor, and control engine. The KN layer (part of KN allocated to the current computer) is kept within the data processor which also holds nodal variables dynamically attached to KN nodes. The tracks layer is supported by the control engine which implements all control rules of waves and suspends wave tails until the rules terminate, the tails being subsequently sent further via the created tracks.

![Figure 7. WAVE interpreter architecture.](image-url)

Parser decomposes waves into their heads (first period-separated part on the top level) and tails (the rest of waves) and sends the parsed heads accompanied by the wave tail to the data processor if the head identifies elementary operations (acts) on top level. If the head splits on top level into parts separated by comma (called sectors), the original wave is substituted in the waves queue by a set of waves formed by the sectors (as new heads) with appending to each a common tail. This decomposition process in the parser recursively
continues (with removing parentheses which become redundant) until the elementary acts or rules are found in the heads of waves, after which either the data processor or the control engine become engaged.

After the data processor, the wave tails are sent back to the parser or to the control engine. The latter action takes place when hops to other nodes should be executed on the operands prepared in the data processor, as hops must be accompanied by new tracks. Control engine executes control rules of waves and establishes links between tracks and the wave branches into which the rule-controlled waves are split, these branches being processed from the waves queue in the parser. Echoes are merged in the track nodes by the control engine with the final results used to assess the success or failure of the whole branches and influence invocation of other branches by the rules. The control engine also triggers the process of garbage deletion in the data processor when waves terminate (which is associated with the deletion of tracks). An external call act allows for communication with other systems on the same computers by exchanging data with them via the data processor.

Parts of the KN and track forests located in different interpreters form together a seamless distributed and dynamic information & processing space where waves (accompanied by moving data variables) and echoes are propagating either within the memory of the same machines, or being automatically passed via incoming–outgoing queues to other interpreters on other machines.

A publicly available WAVE interpreter is currently written in C (with graphical interface in TCL) and operates via Internet.

5. WAVE and other paradigms

In the same way as graphs and networks are widely used for description and analysis of systems of a different nature (say, biological, social, or technical), the WAVE paradigm, creating and processing arbitrary knowledge networks in a distributed environment, can be readily used for modelling any other programming techniques. The latter may especially include different models and languages for parallel and distributed processing.

5.1. Petri nets / dataflow

Petri nets, originally proposed by Petri [14], still remain widely used as a model for description of event-driven asynchronous systems of any nature. In Fig.8, a WAVE program is shown which first creates a certain petri net topology with setting up its initial marking in place p5 (places represented by circles), and then puts into all transitions (drawn as bars) active functions which continuously check presence of tokens in all their incoming places, and if "yes", remove tokens from these places with adding them to the all outgoing places of the fired transition. The created net operates arbitrary long time, with transitions t2 and t3 firing in parallel. The WAVE program puts active functions only into the transition nodes, recognising them by checking if their names contain letter "t". ("a" is used as all link names.)

\[
\text{SQ(CR}@\text{t1,F=}\text{A}.+\text{a#p2}.+\text{a#t3}.+\text{a#p4}.+\text{a#t4}. \\
(\text{a#p3}.-\text{a#t2}.-\text{a#t1}.-\text{a#F}),(+\text{a#p5}.N=1,+\text{a#F})), \\
(@\#.C|::t=.RP(AS( (AP(-#.N/=).N=1), (+#.N+1),3))),))
\]

**Figure 8. Petri net creation & activation.**

Similar to the example above, any other graph-based models with interpreted nodes and moving tokens (the latter as arbitrary data structures), like dataflow, actors, neural networks, etc., may be efficiently represented in WAVE. Moreover, such WAVE-based networks may evolve in space and change their topology at runtime which many other existing models cannot do.

5.2. Mobile agents

Mobile agents [3, 24] have become a hot topic for discussion during the last years. They allow to launch autonomous programs into a network which may freely travel between computers and do some jobs on behalf of a particular user, like business negotiation, booking tickets, etc., letting programs move to data, and not the usual way round. Mobile agents allow, in many cases, to reduce traffic in networks and organise solutions of many problems in a much more flexible way. Mobile agent techniques are usually based on object technologies, with self-contained and rather large programs. In WAVE, as was shown above, the moving and cooperating parts of a spatial program may be of arbitrary size – from large programs to elementary operations. WAVE also contains powerful spatial control rules for coordination between different moving parts which mobile agents are usually lacking.

Any existing mobile agent systems can be easily modelled in WAVE. Let us consider (Fig. 9) a creation and activation of two agents: a1 and a2, where the first one starts from node "b" and travel sequentially through links "k" and "q", the latter leading to node "e", and the second one should start from "c" while passing "n" and "q" links, the later ought to lead to "f". Let a2 be allowed to start only after a1 terminates, and let such control be located in node "a". Starting in "a", the wave

```
```

```
```
in Fig. 9 creates the needed agents, whereby a2 is activated only after full completion of a1 which is echoed to the SQ rule via dynamically created tracks. Other spatial control rules of WAVE may be used too for direct subordination and coordination between agents.

![Spatial control diagram]

**Figure 9. Creation & coordination of mobile agents.**

Agents may also cooperate indirectly, by sharing with each other any information in nodes they pass, like in the case for agents a3 and a4 below, where a4 will be busy-waiting in node "d" until a3 passes and marks it.

\[
\begin{align*}
a3: & \quad (@#e.\#.N=1.k#) \\
a4: & \quad (@#f.\#.RP(N==).n#)
\end{align*}
\]

Both agents are sharing nodal variable N in "d". Integration of these two agents within a wave is simple:

\[
(@#e.\#.N=1.k#),(@#f.\#.RP(N==).n#).
\]

5.3. VRML

Languages for representing 3D virtual reality in computer networks are of growing popularity nowadays. The current VRML 1.0 specification details a text language for describing three-dimensional scenes [4] and has provision for hyperlinks. In an effort to add dynamics to VRML scenes, a number of its extensions have been developed. All these, however, base the description of a scene on a rigid structure, usually a tree known as a "scene graph" which cannot be effectively changed from within the VRML programs as it reflects the program text. WAVE may provide fully distributed and highly parallel multi-user processing of the VRML scenes, parts of which (or the whole) could be easily modified (by dynamic restructuring of scene graphs in WAVE) and could migrate between computers. Efficiently working with the knowledge (semantic) networks, WAVE may provide parallel processing and inference on the deepest levels of distributed knowledge representations. The languages like VRML supporting visual representations of the modelled worlds and direct communications with the users may be on the surface of this semantic knowledge processing.

In Fig.10, a representation in WAVE of a scene graph is shown where numbers at links set an order in which the corresponding program parts (subtrees) should be located in the VRML program text. A wave which adds to this graph a new transform node with parameters, allowing the whole set of n objects rotate as a group, is also shown. As this new part should be placed before the other parts in the text (i.e. to be reflected by link named "1" in the graph), all other names of links emanating from the top separator node must be incremented by 1.

![Scene graph modification diagram]

**Figure 10. Scene graph modification in WAVE.**

This WAVE-based scene graph easily converts into a VRML text by the following highly parallel wave program which applies in the "start" node in Fig.10:

\[
\begin{align*}
F_0 & = Fc+1.AS(Fc#.,N&x.Fc#.,F_0). \\
F & = SQ(OS((Fc=.F_0.F),(Fr=C&'{'&N&'}'%' '.#P.N:Fr))),(F_0=F'&'\%') \} .#P.N:Fr \}'). \\
@#start.RP(SQ((N=x.F.1),T=N).5?sleep)
\end{align*}
\]

This program recursively navigates the (distributed) graph while regularly synthesising (here every 5 sec.) the VRML text from its parts when echoing up the tree, whereas the graph may itself be changed at the same time by another wave(s). The resultant VRML text may be rendered using standard techniques, and will look like:

```
Separator
  {  Transform { rotation 0 1 0 0 }  \\
    Separator
      {  Material { diffuseColor 1 0 0 }  \\
        Transform { translation -4 4 }  \\
        Transform { rotation 0 1 0 0 }  \\
        Cube {}  }  ...
  }
```

Instead of synthesising the VRML source code, much more efficient would be to provide direct rendering of the scene graphs produced and coded in WAVE.

Any other visualisation techniques may be easily used on top of the dynamic distributed semantic worlds expressed in WAVE. For example, a possible representation of a terrain may be a grid, each node containing data such as height, surface type, etc. Such a grid may be created in WAVE as a KN and dynamically distributed between any number of computers. Mobile wave societies may produce on these grids actively...
changing shapes spread among computers (e.g. growing craters, flooding, or moving mountains). This process is fully open, i.e. any (multiple) agent activities in these worlds may be started in parallel at any time, by different users, and from different machines.

5.4. Java

The WAVE ideology of solving complex graph and network problems in distributed and dynamic spaces is being developed for a rather long time, having materialised now into an extremely dense its definition shown in Fig. 4, with only five syntactic categories. As the WAVE model was radically new during all its development history, the author often had to give the answers like follows:

a) WAVE is not petri net, because it is dynamic, operations-moving, and self-evolving; b) it is not dataflow but a quite opposite - program flow; c) it is not neural net as it creates networks through which both operations and data may move, not only analog signals influencing thresholds; d) it is not actors [10] representing only evolution of processes, regardless of data structures to be processed; e) it is not Telescript [24] which appeared much later, after the main WAVE definition was published [16], and actually represents only a small subset of space navigation and coordination features of WAVE. And, to continue this:

**WAVE is not Java.** Java [11] allows for code movement, it is distributed, interpreted and platform independent, like WAVE. But this is only an external similarity. Java is based on a conventional programming philosophy where a program processes stored data and communicates with other programs in a client-server mode. WAVE philosophy is based on a self-navigation and pattern-matching, with mobility and activity of all constructs and any control initiative totally embedded in a mobile code. It also enables to create dynamically any data networks and recursive control in a distributed space, which Java cannot do directly. WAVE is not a programming language in a usual sense, it is rather a computational model, automaton, integrating a universal set of novel space-control features within a program flow mode of computing not present in other paradigms.

An attempt to compare WAVE with Java has been done recently in [22], in favour of WAVE, being however rather artificial as WAVE and Java belong to quite different classes like, say, Lisp and Fortran. Nevertheless an integration of WAVE with Java may be quite useful and is currently being analysed, including re-implementation of WAVE in Java (instead of C), with possibility of accessing any Java routines from WAVE. Moreover, the “flying” WAVE engine may also be accessible from Java supplying it with a universal spatial control Java does not have. Java's multithreading and code mobility may be useful to support WAVE too.

Of course, everything can in principle be programmed in any language (or even in a machine code) in computer networks, if there is a technical possibility of sending messages between machines. However many network processing tasks for which WAVE suits well, when written in other languages, will inevitably have to include explicitly a variety of special functions which are hidden in the wave interpreter and shared by many waves. Direct code in C, as well as in Java, as experiments show, will be more than 50-100 times longer and much more complex.

6. Conclusions

We have described a universal WAVE automaton capable of solving arbitrary problems in distributed and open computer networks without any central resources. The automaton is based on a program flow mode of operation, rather than on traditional dataflow, where cooperative mobile programs navigate in distributed networks while self-replicating, splitting and modified.

The main difference between WAVE and the other mobile programming paradigms is in presence of a powerful recursive spatial control in WAVE efficiently coordinating societies of mobile agents, and the two dynamic service layers operating between mobile application programs (waves) and the computer networks they propagate through. The latter being the knowledge network layer allowing for the creation of abstract distributed worlds reflecting different domains with subsequent processing of these worlds directly, and the tracks layer effectively supporting distributed control and communication in these worlds. The use of these layers allows us to free the application programming from most of the routines on synchronisation, message passing, routing, hierarchical control and garbage collection, which have to be explicitly managed within the traditional distributed systems.

The WAVE automaton ideology efficiently supports distributed algorithms in open networks which:
—may not be known for the network in advance;
—may start from arbitrary node, on the node’s initiative;
—may work with part of the network, which may not be known in advance and may be outlined only at runtime;
—may use any (and all) network resources in parallel;
—behaving like highly organised viruses, may efficiently recover from complex failures, including failures and damages in the underlying system's software and hardware.

Different users may start their spatial mobile programs simultaneously from the same or different
nodes, these programs may overlap in the network and be completely independent from each other or cooperate in the network space while solving complex interactive problems.

The WAVE model, based on a dynamic activation of massively parallel processes in a data space, exhibits a general parallelism proportional to a number of nodes in a network it processes, thus being easily scaled to an arbitrary extent while acting without any central control or other centralised resources.

In spite of a successful direct use of WAVE in a variety of application projects, which have shown its potentially unlimited power for high-performance computing in open network topologies, it needs (and can provide an insight for) the design of higher level, user-friendly languages for mobile programming, as well as an efficient conversion into it from other paradigms.

WAVE, in its pure form, may also serve as a convenient model for studying fundamental features of other models and systems, especially those dealing with dynamics, openness and self-recoverability. It may also provide a basis for the design of radically new distributed algorithms for advanced communications, distributed simulation and control, self-organisation, evolution, and emergent functionality.

Acknowledgements

Special thanks are due to the anonymous referees whose frank comments and robust criticism helped much in improving and enriching this paper.

References