SCALABLE COMPUTING
Practice and Experience

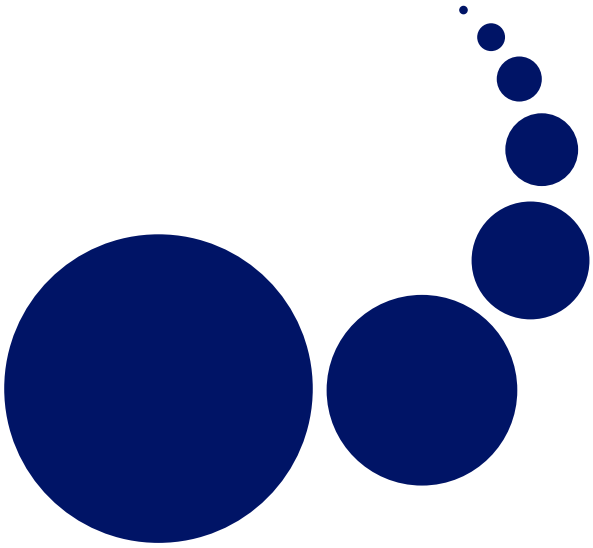
Special Issues:

Selected Papers from workshops ACSys2009 and IDC'2009

Editors: Viorel Negru, Adina Magda Florea, Costin Bădică

Selected paper of Grid&SEA 2008

Editor: Dana Petcu



Volume 11, Number 1, March 2010

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 11, Number 1, March 2010

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 11, Number 1, p. i. http://www.scpe.org
ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE: SELECTED PAPERS FROM WORKSHOPS ACSYS2009 AND IDC'2009

Dear SCPE Reader,

The recent development of applications that exploit multiple information sources, distributed resource processing, user mobility, interaction and information sharing calls for novel models of coordination, data acquisition, transfer and integration, efficient access and processing of information. The articles presented in this issue propose some intelligent distributed models and algorithms that try to cope with the large quantity of information available to users in different application contexts, and draw from different computation paradigms such as multi-agent technology, biologically inspired computation, peer-to-peer systems, mobile ad hoc network, grid computing.

The articles in this issue of SCPE are extended versions of selected papers presented at the workshops ACSys2009 and IDC'2009.

ACSys2009, the 6th Workshop on Agents for Complex Systems was held in Romania, Timişoara, September, 26–29, 2009. The workshop was organized in the frame of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2009).

The aim of the ACSys2009 workshop was to bring together researchers and developers from academia and industry in order to discuss current scientific and technical results on using agent technology for solving complex, distributed real-world problems (http://synasc09.info.uvt.ro/workshops/ACSys/).

The article "Algorithmic Solutions for Several Offline Constrained Resource Processing and Data Transfer Multicriteria Optimization Problems" by Mugurel Ionuţ Andreica and Nicolae Ţăpuş presents novel algorithmic solutions for several resource processing and data transfer multicriteria optimization problems. The results of most of the presented techniques are strategies which solve the considered problems (almost) optimally.

The article "Sensors Data—Stream Processing Middleware Based on Multi-Agent Model" by Ovidiu Aritoni and Viorel Negru proposes a multi-agent architecture for an intelligent sensor data processing middleware that allows acquisition, interpretation and aggregation of sensor data-streams. A healthcare system is used for validating the architecture.

The article "Context-Aware Emergent Behaviour in a MAS for Information Exchange" by Andrei Olaru, Cristian Gratie and Adina Magda Florea presents, based on authors previous work on self-organizing multi-agent systems for information exchange, two aspects of context-awareness—pressure and interest—that make the system's emergent behavior context-sensitive and, therefore, more adaptive to a changing environment.

IDC'2009, the 3rd International Symposium on Intelligent Distributed Computing, was held in Cyprus, Ayia Napa, October 13–14, 2009.

The aim of this symposium was to bring together researchers involved in intelligent distributed computing to allow cross-fertilization and synergy of ideas and to enable advancement of researches in the field (http://www.idc2009.cs.ucy.ac.cy/).

The article "Efficient Broadcasting in MANETs by Selective Forwarding" by Doina Bein, Ajoy K. Datta, and Balaji Ashok Sathyanarayanan presents a new protocol called Efficient Broadcasting by Selective Forwarding that minimizes the number of transmissions or retransmissions needed for broadcasting in mobile ad hoc networks.

The article "Group-Based Interactions for Multiuser Applications" by Carmen Morgado, José C. Cunha, Nuno Correia, and Jorge Custódio presents a group-based model for the development of interactive applications where multiple users dynamically interact with each other, and which facilitates the organization, access and sharing of content by multiple users. It also describes examples to support communities in tourism and school campus scenarios.

The article "Exploring Carrier Agents in Swarm-Array Computing" by Blesson Varghese and Gerard Mckee proposes a swarm-array computing approach to bridge the gap between autonomic and parallel computing. The feasibility of this proposal is validated by two simulation studies: using a multi-agent simulator on a Field-Programmable Gate Array, and using Message Passing Interface on a computer cluster.

Great thanks are first given to SCPE and its Editor-in-Chief Dana Petcu who kindly offered the publication of this special issue. We would also like to thank George A. Papadopoulos, who helped with co-organizing IDC'2009.

Viorel Negru, Adina Magda Florea, Costin Bădică

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 1–17. http://www.scpe.org
ISSN 1895-1767

ALGORITHMIC SOLUTIONS FOR SEVERAL OFFLINE CONSTRAINED RESOURCE PROCESSING AND DATA TRANSFER MULTICRITERIA OPTIMIZATION PROBLEMS

MUGUREL IONUŢ ANDREICA* AND NICOLAE ŢĂPUŞ*

Abstract. In this paper we present novel algorithmic solutions for several resource processing and data transfer multicriteria optimization problems. The results of most of the presented techniques are strategies which solve the considered problems (almost) optimally. Thus, the developed algorithms construct intelligent strategies which can be implemented by agents in specific situations. All the described solutions make use of the properties of the considered problems and, thus, they are not applicable to a very general class of problems. However, by considering the specific details of each problem, we were able to obtain very efficient results.

Key words: data transfer optimization, resource processing, peer-to-peer, data structures, permutations, sorting

1. Introduction. Resource processing and data transfer multicriteria optimization problems occur in many situations, particularly in large distributed systems, like Grids, distributed databases, live and on-demand video streaming applications, peer-to-peer systems, and so on. The problems that occur in practical settings are of a wide diversity and consider the optimization of various parameters, while imposing constraints on others. Handling such problems from a general perspective is useful when trying to understand and classify them, but it is of little use when we want to solve a specific problem. In this paper we consider several such multicriteria optimization problems, for which we provide novel, very specific algorithmic solutions. Instead of presenting a general technique and analyzing its efficiency on various types of optimization problems, we focus on the problems and develop different solutions for each problem. The considered problems are mainly offline, meaning that all the required information is available in advance. From this point of view, the presented results cannot be applied directly in real-time settings (where most of the resource processing and data transfer optimization problems occur). However, from a theoretical point of view, the developed algorithms are of significant interest and they are the first steps towards obtaining efficient online solutions to some of the considered problems.

The rest of this paper is structured as follows. In Section 2 we discuss several offline (multi)point-to-(multi)point data transfer optimization problems, like finding a deadline-constrained packet transfer strategy, computing a minimum cost communication spanning tree when the link providers have special offers, and finding a subset of edges (vertices) of maximum average weight (when we have two weights assigned to every edge or vertex). In Section 3 we propose a new agent-based peer-to-peer content delivery architecture. We present generic methods and guidelines and discuss the issues of point-to-point and multicast data transfers within the topology. In Section 4 we discuss several constrained (multi-)permutation construction and sorting problems. Such problems are partially related to efficiently ordering the packets of a communication flow in the transmission buffer. In Section 5 we present new algorithmic extensions to the Union-Find and Split-Find problems. These are set maintenance problems which require efficient management and querying of sets of elements. They have applications in several resource processing situations (e.g. when reserving resources, see [6]). In Section 6 we discuss another resource processing problem, for which we present an optimal algorithm. The problem consists of activating/deactivating resources in a graph with bounded treewidth by using a minimum cost strategy. This problem is a more general version of the *minimum all-ones problem* [9], which assumes that the final state of each resource must always be *active* and the cost of selecting a resource is always 1. Finally, in Section 7 we discuss related work and in Section 8 we conclude.

2. Offline (Multi)Point-to-(Multi)Point Data Transfer Optimization.

2.1. Optimal Deadline-Constrained Packet Transfer Strategy. We consider a directed graph with n vertices and m edges. A packet is sent from the source node s at time 0 and must reach the destination node d by time T. Every directed edge e from a vertex u to a vertex v has an associated start time $tstart(e)$ and a finish time $tfinish(e)$ ($tfinish(e) > tstart(e)$). The meaning of these parameters is that the packet can only be sent along that edge, from u to v, starting at the moment $tstart(e)$ and will only arrive at vertex v at the moment $tfinish(e)$. Thus, edge e corresponds to a reservation in the underlying network. Moreover, out of the total packet transmission time (equal to $tfinish(e) - tstart(e)$), $twait(e)$ is the total time during which the

*Computer Science and Engineering Department, Faculty of Automatic Control and Computer Science, Bucharest, Romania ({mugurel.andreica, nicolae.tapus}@cs.pub.ro).

packet has to wait in the waiting queues (e.g. it must wait for some data processing task or must wait until other packets before it are sent along the edge). The time between the moment when the packet arrives to a vertex u and the moment when it is sent to another vertex v (or between the moment when it last arrives at vertex d and the moment T) also counts as waiting time. We want to find a packet transfer strategy minimizing the total waiting time. The steps of the algorithm are described below.

1. For every vertex v of the graph:
   (a) We will sort together the incoming and outgoing edges in increasing order, according to a weight assigned to every edge: for an incoming edge e from a vertex u to vertex v, the weight is $w(v,e) = tfinish(e)$; for an outgoing edge e from vertex v to a vertex u, the weight is $w(v,e) = tstart(e)$.
   (b) If two edges (an incoming and an outgoing one) have the same weight, then we will place the incoming edge before the outgoing edge in the sorted order.
   (c) For every edge in the sorted order of a vertex v we will store its type: incoming or outgoing.
   (d) Let $deg(v)$ be the total number of incoming and outgoing edges adjacent to vertex v.
2. We will compute $TWmin(v,i)$=the minimum total waiting time required for the packet to be located at vertex v at the time moment $tm(v,i)$=the weight of the i^{th} edge in the sorted order for vertex v ($1 \leq i \leq deg(v)$).
3. We will consider $TWmin(v,0) = +\infty$ and $tm(v,0) = 0$ for every vertex $v \neq s$ and $tm(s,0) = TWmin(s,0) = 0$.
4. We will sort ascendingly all the time moments $tm(v,i)$ ($i \geq 1$) in increasing order (e.g. by merging the lists of time moments $tm(*,*)$) and we will store for each moment the associated values v and i.
5. We will traverse all the time moments $tm(v,i)$ in increasing order.
   (a) If $tm(v,i)$ corresponds to an incoming edge e (from a vertex u to vertex v), then we will first find the index j of the edge e in the sorted order of the edges adjacent to vertex u.
   (b) We will have $TWmin(v,i) = \min\{TWmin(v,i-1)+tm(v,i)-tm(v,i-1), TWmin(u,j)+twait(e)\}$.
   (c) If $tm(v,i)$ corresponds to an outgoing edge e from vertex v to a vertex u then we set $TWmin(v,i) = TWmin(v,i-1) + tm(v,i) - tm(v,i-1)$.

We can find the index j of an edge e in the sorted order of a vertex u by using a hash table $HT(u)$. After sorting the edges adjacent to vertex u we traverse these edges: let the edge e' be the i^{th} edge in this order—then we insert the pair $(key = e', value = i)$ in $HT(u)$. Thus, in order to find the index j associated to an edge e in the sorted order of a vertex u we just search in $HT(u)$ the value associated to the key e. Such a lookup takes $O(1)$ time and the overall time complexity is $O(m \cdot \log(m))$.

The final answer is $\min\{TWmin(d,i) + T - tm(d,i) | 1 \leq i \leq deg(d), tm(d,i) \leq T\}$.

As we can notice, the problem can also be interpreted as a shortest path problem in the graph of the pairs (v,i), where the starting pair is $(s,0)$. We have an edge from each pair $(v,i-1)$ to the pair (v,i) with cost $tm(v,i) - tm(v,i-1)$ ($1 \leq i \leq deg(v)$). Moreover, we have an edge from each pair (u,j) to a pair (v,i), with cost $twait(e)$ if e is an edge from u to v and is the j^{th} edge in the sorted order of vertex u and the i^{th} edge in the sorted order of vertex v. With this interpretation, we can compute a shortest path from $(s,0)$ to the pairs $(d,*)$, in $O(m \cdot \log(m))$ time (the graph of pairs has $O(m)$ vertices and edges). If we denote by $TWmin(v,i)$=the length of the shortest path from $(s,0)$ to (v,i), then the answer is computed the same as before.

2.2. Maximum (Minimum) Ratio Constrained Subsets of Edges (Vertices). We are given a (directed) graph G with n vertices and m edges. Each (directed) edge (u,v) has two associated values: $p(u,v) \geq 0$ and $q(u,v) > 0$, and each vertex u has two associated values: $p(u) \geq 0$ and $q(u) > 0$ (e.g. these values could be bandwidth, cost, or latency). We want to find a subset of edges E (vertices V) having a specified property $Prop$, such that its ratio is maximum (minimum). The ratio A of a subset of edges E (vertices V) is defined as $\frac{\sum_{(u,v) \in E} p(u,v)}{\sum_{(u,v) \in E} q(u,v)}$ $\left(\frac{\sum_{u \in V} p(u)}{\sum_{u \in V} q(u)}\right)$. We will use a technique which was previously discussed in other research papers [4] for solving other optimization problems. Let's consider the following problem: We are given a (directed) graph G with n vertices and m edges. Each (directed) edge (u,v) (vertex u) has a weight $w(u,v)$ ($w(u)$), which may be zero, positive or negative. We want to find a subset of edges E (vertices V) satisfying property $Prop$, such that the sum of the weights of the edges $(u,v) \in E$ (vertices $u \in V$) is maximum (minimum). We will denote the optimization algorithm by $OPTA(G)$, where G is the graph. $OPTA(G)$ returns the weight of the subset of edges (vertices). The algorithm will compute several other values, based on which we will easily be able to find the subset of edges (vertices) itself (not just its weight). With this algorithm, we will be able to solve the maximum (minimum) ratio problem as follows. We will binary search the maximum (minimum) ratio A_{opt} in

the interval $[0, AMAX]$, where $AMAX$ is a good upper bound (e.g. $AMAX$ is the maximum among the $p(*,*)$ ($p(*)$) values, divided by the minimum among the $q(*,*)$ ($q(*)$) values). For a candidate value A_{cand}, we will construct a graph G' having the same set of edges and vertices. The weight of a (directed) edge (u, v) (vertex u) in G', $w(u, v)$ ($w(u)$), will be $p(u, v) - A_{cand} \cdot q(u, v)$ ($p(u) - A_{cand} \cdot q(u)$). We will now call $OPTA(G')$ and store its value as A_{res}. If $A_{res} < 0$, then A_{cand} is larger than the maximum (minimum) ratio and we need to consider smaller values; otherwise, $A_{cand} \leq A_{opt}$ and we can test larger values. The binary search ends when the length of the search interval becomes smaller than ε (where $\varepsilon > 0$ is a very small constant). We will now provide a few examples of optimization problems and algorithms ($OPTA$) which can be fit into the generic framework introduced earlier.

Given a directed graph G with n vertices (numbered from 1 to n) and m edges, where each directed edge (u, v) has a weight $w(u, v)$ (which may be zero, negative or positive) and a length $l(u, v) > 0$, we want to find a (possibly self-intersecting) path (cycle) of total length at least L and at most U, whose total weight is maximum. We will present a pseudo-polynomial algorithm, for the case when the edge lengths are integers. For the path case, we will compute for each vertex i and each number of edges k, the value $W_{\max}(i, k)$=the maximum total weight of a path with length k, which ends at vertex i. We have $W_{\max}(i, 0) = 0$ and for $1 \leq k \leq U$, we have $W_{\max}(i, k) = \max\{W_{\max}(j, k - l(j, i)) + w(j, i) | (j, i) \in G, l(j, i) \leq k\}$ or $-\infty$, if no edge (j, i) with $l(j, i) \leq k$ exists. The maximum total weight of an optimal path is $\max\{W_{\max}(i, k) | 1 \leq i \leq n, L \leq k \leq U\}$. The time complexity is $O((n + m) \cdot U)$. For the cycle case, we consider every possible starting vertex s and run the algorithm described above, except that $W_{\max}(s, 0) = 0$ and $W_{\max}(i \neq s, 0) = -\infty$. The weight of the optimal cycle starting at vertex s is $\max\{W_{\max}(s, k) | L \leq k \leq U\}$. When there is no constraint on the length of the path (cycle), we first need to decide if the graph contains a cycle whose total weight is positive. If it does, then the weight of the optimal path (cycle) is $+\infty$. We will perform this checking by computing a value $W_{\max}(i)$ for each vertex i, using the Bellman-Ford-Moore algorithm. We initialize $W_{\max}(*) = 0$ and insert all the vertices into a queue Q. Afterwards, we repeatedly extract from Q the first vertex i, consider all the directed edges (i, j) (from i to j) and update $W_{\max}(j)$. If $W_{\max}(i) + w(i, j) > W_{\max}(j)$, then we set $W_{\max}(j) = W_{\max}(i) + w(i, j)$ and insert the vertex j at the end of the queue (if it is not already in the queue). We will also maintain a counter $nin(i)$ for each vertex i, denoting the number of times vertex i was inserted into the queue. If, at some point, we have $nin(i) > n$ for some vertex i, then a cycle with total positive weight exists in the graph. If the algorithm ends without finding a positive cycle, then the maximum total weight of a cycle is 0 and that of a path is $\max\{W_{\max}(i)\}$. The time complexity of the algorithm in this case is $O(n \cdot m)$.

When the graph is a directed path, i. e. we have only the edges $(i, i + 1)$ $(1 \leq i \leq n - 1)$, the optimal path problem can be solved with a better time complexity. The unconstrained version is equivalent to computing a maximum sum segment in the sequence of $n - 1$ weights $w(i, i + 1)$. The length constrained version is equivalent to the length constrained maximum sum segment problem.

2.3. Minimum Cost Spanning Tree with Special Offers. We consider here a minimum cost spanning tree problem, augmented with special offers. We have an undirected graph with n vertices (numbered from 1 to n) and m edges. Each edge e connects two different vertices $a(e)$ and $b(e)$, has an owner $o(e)$ and two prices: $np(e)$ and $sp(e)$. $np(e)$ is the normal price of the edge and $sp(e)$ is the special price of the edge; $sp(e) \leq np(e)$. There are q owners overall, numbered from 1 to q. Each owner has a special offer: it allows us to pay the special prices for any edges we want owned by that owner, with the condition that we do not take the special offer of any of the other owners. We want to establish a minimum cost spanning tree of the graph, by using at most one special offer of one of the edge owners.

At first, we will compute the minimum spanning tree of the graph considering normal prices for all the edges (in $O((m + n) \cdot \log(n))$ or $O(m + n \cdot \log(n))$ time). Let $MSTN$ be the set of $n - 1$ edges composing the "normal" minimum spanning tree and let $CMSTN$ be the cost of the minimum spanning tree considering the normal prices.

We will now consider every owner i (from 1 to q) and compute the minimum spanning tree in the case when we take advantage of owner i's special offer, i. e. when we consider the special prices for all the edges owned by i. We can recompute each such spanning tree in the same time complexity as when we computed the first minimum spanning tree, but the overall time complexity would be $O(q \cdot (m+n) \cdot \log(n))$ or $O(q \cdot (m+n \cdot \log(n)))$. Instead, we will proceed as follows. When computing the minimum spanning tree for the special offer of the owner i, we will consider only the subset of edges $SE(i)$ composed of all the edges owned by i (for which we consider their special prices) and all the edges from $MSTN$ which are not owned by i, for which we consider

their normal price. Then, we will compute the minimum spanning tree using only the edges in $SE(i)$. Note that any edge outside of $SE(i)$ cannot be part of this minimum spanning tree. Thus, the time complexity for one minimum spanning tree computation is $O((|SE(i)| + n) \cdot \log(n))$ or $O(|SE(i)| + n \cdot \log(n))$.

The sum of the values $|SE(i)|(1 \leq i \leq q)$ is at most $m + (n-1) \cdot q$. Thus, the overall time complexity will be $O((m + n \cdot q) \cdot \log(n))$ or $O(m + n \cdot q + n \cdot q \cdot \log(n))$, which is much better than that of the trivial algorithm.

For a more in-depth analysis of this problem (which considers several other constraints and other types of minimum spanning tree algorithms), see [10] (a book based on the first author's Ph.D. thesis).

3. Towards an Agent-Based Peer-to-Peer Content Delivery Framework. In this section we propose a novel architecture of an agent-based peer-to-peer content delivery framework in which data transfer optimization techniques can be used. The problems posed by this framework are mainly online, but we will also discuss a semi-offline problem in the last subsection. Note that we are only presenting the generic principles and guidelines in this section, and not a full implementation.

Each agent (which will be called *peer* from now on) of the peer-to-peer framework has an identifier which is a point in a d-dimensional space (in which the range of the coordinates in each dimension i is $[0, VMAX(i)]$). The identifiers can be self-generated (e.g. by using hash functions which generate unique values with high probability), or may be computed according to some specific rules (e.g. they may be mapped in the latency space) [11]. The peers interconnect in a peer-to-peer topology by using a distributed decision making mechanism. In [12] we presented a generic framework for constructing a peer-to-peer topology based on the peers' geometric coordinates. Several methods for choosing neighbors are also described in [12], out of which we would like to highlight the *Hyperplanes* method. This method makes use of a set of hyperplanes, all of which contain the origin. These hyperplanes form Q disjoint regions, whose union is the entire hyper-space. When a peer X has to choose its neighbors from a set $I(X)$, it will divide the peers Y from $I(X)$ according to which region (among the Q regions) they belong to (when peer X is considered to be the origin) and then it will select as neighbors the closest K peers from each region (K may be different for each region and for each peer). Closeness may be defined using any geometric norm L_h $(1 \leq h \leq +\infty)$. The peers periodically broadcast their existence a number of hops away in the topology and the set $I(X)$ is composed of those peers Y which broadcasted their existence to X recently. The topology changes as the sets $I(*)$ change and the peers select new neighbors, until it reaches an equilibrium (if no new peer joins the system and no old peer leaves it).

In order for the peer-to-peer topology construction mechanism to work properly, we assumed that all the peers are connectable (i. e. they have public IP addresses and are not located behind a firewall). Note that the system may still work if, whenever a connectable peer X selects an unconnectable peer Y as its neighbor, Y also selects X as a neighbor. In order to handle unconnectable peers we consider assigning coordinates at the edge of the multidimensional space to these peers. This way we expect that an unconnectable peer Y is less likely to be selected as a neighbor by a connectable peer X which is not selected back as a neighbor by Y. This does not solve the problem of two unconnectable peers which may select each other as neighbors. We assume that every peer can detect if it is unconnectable (this may be easier when the peer has a private IP address than when it is located behind a firewall).

In order to send data from one peer to another, we can employ geometric routing (i. e. we repeatedly forward the data to a neighbor which is closer in the geometric space to the destination), or we can use multi-path routing mechanisms [12].

Searching for a piece of content is not within the scope of this paper. Nevertheless, when the content units have a set of index properties, the approach from [13] can be used in order to obtain references to every content unit (or at most a number M of them) whose properties are within a specified multidimensional range. Once we have a reference to a content unit, we can initiate a data transfer from its owner.

Except for point-to-point data transfer services, we are interested in supporting multicast communication services within the peer-to-peer architecture. In the following two subsections we will propose two different methods for achieving this. We will first present a generic method for constructing multicast trees on demand. Then, we will discuss the problem of maintaining a multicast tree with improved stability, when we have extra information regarding the peers' life times.

3.1. On Demand Multicast Data Distribution. In order to support multicast communication services, we propose building a multicast tree on demand on top of the peer-to-peer topology. Many spanning tree construction algorithms for arbitrary topologies have been proposed in the literature [14, 15]. However, the

topology proposed in the previous section has a special geometric structure. Thus, we propose the following high-level construction method.

One of the peers A will initiate the multicast data transfer. Let's denote by $Z(X)$ the zone for which a peer X will be responsible. $Z(A)$ will be the entire geometric space. Let's assume that a peer C received a message with the value of $Z(C)$ (peer A is considered to receive such a virtual message in the beginning). Then, C will select a subset of its neighbors $S(C)$ which are located within $Z(C)$ and will construct tree edges from itself towards the peers B in $S(C)$. Then, a disjoint zone $Z(B)$ of the space is computed by peer C for each peer B in $S(C)$, such that: (1) every zone $Z(B)$ is fully contained in $Z(C)$; (2) every peer D whose coordinates are within $Z(C)$ (except for C and the peers from $S(C)$) must be located within one of the zones $Z(B)$ ($B \in S(C)$). When using the Hyperplanes neighbor selection methods, the zones $Z(B)$ can be computed by clipping the zone $Z(C)$ with a subset of hyper-planes (thus, the zones are convex). Then, peer C sends the message with $Z(B)$ to every peer B from $S(C)$. The time complexity of this algorithm is optimal, being proportional to the diameter of the underlying peer-to-peer topology.

In some cases, we may decide that the zones $Z(B)$ of the peers $B \in S(C)$ do not have to be disjoint (but their union may be equal to $Z(C)$). Non-disjointness may imply that some peers will end up receiving the data multiple times. This is acceptable if a peer can detect that the received data is duplicate (i. e. the data packets have unique identifiers) and if the complexity of computing non-disjoint zones $Z(B)$ would be too high (either from a computational point of view, or because the description complexity of $Z(B)$ would be too high).

The multicast tree constructed by this method would stay alive only as long as data is transferred along its edges (each edge would have its own expiration timer). A new tree would be constructed for every multicast flow. Although this generates a tree construction overhead for each flow, it is more robust in the presence of node failures (because the trees are not long-lived, they are less likely to experience node failures or departures).

3.2. Constructing Multicast Trees with Enhanced Stability based on Availability Information. In this subsection we consider the following scenario. Let's assume that every peer X that enters the peer-to-peer system knows the time moment $T(X)$ when it will leave the system (and can guarantee that it will stay within the system until the moment $T(X)$). We would like to use this information in order to construct and maintain a multicast tree with improved stability.

A method of achieving this is the following. The coordinates of a peer X will be of the form $(v(1), \ldots, v(d-1), v(d) = T(X))$ (where $v(1 \leq i \leq d-1)$ can be generated the way we discussed earlier). Then, the peer X will join the peer-to-peer topology. Once its neighbors are stable (i. e. peer X does not select new neighbors for at least $KR(X)$ consecutive gossiping periods), peer X will choose one of its neighbors Y from the peer-to-peer topology as its neighbor in the tree. We will refer to Y as peer X's preferred neighbor ($P(X) = Y$). The set of candidate peers $PC(X)$ from which $P(X)$ can be chosen consists of those peers Y which are neighbors with X in the topology and have $T(Y) \geq T(X)$. If X has no neighbor Y with $T(Y) \geq T(X)$ then X has the highest value of $T(X)$ from the system. In this case, $PC(X)$ consists of all of X's neighbors within the peer-to-peer topology.

The preferred neighbor $P(X)$ may be chosen arbitrarily from $PC(X)$, but we could also use other rules. For instance, for each peer Y, we may impose an upper bound $UB(Y)$ on the number of peers Z for which $P(Z) = Y$. Then, we need to remove from $PC(X)$ those peers Y which have already reached their limit. Among the remaining peers we could choose the one with smallest maximum distance from itself towards some other peer in the tree (the distance is equal to the number of tree edges which need to be traversed from one peer to another), in order to minimize the tree diameter. Such information can be maintained through gossiping in the multicast tree (see, for instance, [16]). If all the peers in $PC(X)$ have reached their upper limit, then we may decide to either break one of these limits (e.g. by choosing the peer $Y \in PC(X)$ with the smallest degree) or we could replace a tree edge $Z - Y$ (where $Y \in PC(X)$). The tree edge would be replaced by the edges $Z - X$ and $X - Y$. In this case, it might be better to choose that tree edge $Z - Y$ for which the maximum distance from Z towards a peer in its subtree ($Dmax(Y, Z) - 1$ if we use the notations from [16]) is minimum (also in order to try to minimize the diameter). Replacing these edges also implies replacing the corresponding preferred neighbors: if $P(Z) = Y$ then we will set $P(Z) = X$ and $P(X) = Y$; otherwise, we will set $P(Y) = X$ and $P(X) = Z$.

With this method of constructing the tree, the peers X with smaller values of $T(X)$ will be located towards the edge of the tree. Thus, when a peer leaves the system, it will be a leaf in the tree and its departure will not disconnect the tree. This saves us from the need of taking special measures in case of unexpected peer departures which may leave the multicast tree disconnected.

A situation in which the peers X may know exactly the values $T(X)$ is in the case of virtual machines (VMs) deployed in Clouds. If a Cloud computing resource is leased for a fixed amount of time in order to run a virtual machine, then the virtual machine may know the amount of time left before it is powered off. Note also that since leasing virtual machines involves a certain Service Level Agreement (SLA), we can count on the fact the the VM will be running for the specified amount of time without interruptions. In a way, we are transferring the reliability of the SLA negotiated with the Cloud service provider in order to construct a reliable (and stable) multicast tree on top of a flexible peer-to-peer topology. For other scenarios, the assumption of knowing the exact time when the peer leaves the system leads to a semi-offline problem.

4. Construction and Sorting of (Multi-)Permutations.

4.1. Permutations with Average Values in Between. We are interested in constructing a permutation with n elements, such that, for every two distinct values $p(i)$ and $p(j)$ $(i < j)$ of the same parity, the position k on which the element $(p(i) + p(j))/2$ is located must satisfy the condition: $k < \min\{i, j\}$ or $k > \max\{i, j\}$, i. e. the average value must not be located in between the two values in the permutation.

An $O(n \cdot \log(n))$ divide-and-conquer algorithm is not difficult to obtain. We notice that we can have all the $n/2$ even numbers located before all the $n - n/2$ odd numbers in the permutation. Then, we need to solve the same problem for $n/2$ and $n - n/2$ elements. We can obtain a valid n-element permutation as follows. We multiply each number in the $(n/2)$-element solution by 2, thus obtaining all the even numbers in the set $\{1, \ldots, n\}$. Afterwards, we turn every element q in the $(n - n/2)$-element solution into $2 \cdot q - 1$ and obtain all the odd numbers in the set $\{1, \ldots, n\}$. By concatenating the two sequences, we obtain the desired permutation. The time complexity is obviously $O(n \cdot \log(n))$, because, at each step, we need to solve two approximately equal problems. The base case is $n = 1$, where the only existing permutations is the solution.

We can improve the time complexity to $O(n)$ as follows:

- If $n = 4 \cdot k$, then we will call the algorithm for $2 \cdot k$ and $2 \cdot k$.
- If $n = 4 \cdot k + 1$, then we will call the algorithm for $2 \cdot k$ and $2 \cdot k + 1$. At the next level, however, for the $2 \cdot k$-element solution, we will need to solve two k-element problems, and for the $2 \cdot k + 1$-element solution, we will need to solve a k-element problem and one $(k + 1)$-element problem.
- For $n = 4 \cdot k + 2$, we will need to solve two $2 \cdot k + 1$-element problems at the next level.
- If $n = 4 \cdot k + 3$, then we will call the algorithm for $2 \cdot k + 1$ and $2 \cdot k + 2$. At the next level, however, for the $2 \cdot k + 1$-element solution, we will need to solve one k-element problem and one $(k + 1)$-element problem, and for the $2 \cdot k + 2$-element solution, we will need to solve two $(k + 1)$-element problems.

What we notice is that at every level of the recursion, there are at most two distinct (sub)problems to be solved. Thus, there are only $O(\log(n))$ distinct subproblems to solve. We can use the memoization technique. After computing a valid k-element permutation, we store it and retrieve it immediately whenever we require it again. Thus, at subsequent calls, the time complexity will be $O(k)$, instead of $O(k \cdot \log(k))$. Because the sizes of the problems at each level decrease exponentially, the total sum of the sizes of the computed permutations is $O(n)$. Thus, the overall time complexity is $O(n)$.

A simpler $O(n)$ solution is to consider the largest value $m = 2^k$ such that $m \geq n$. Then, we solve the problem for a permutation with m elements (which always has only one (sub)problem to solve at each recursion level) and we simply remove all the elements $q > n$ from the obtained permutation.

4.2. Sorting Permutations by Rotations - Version 1. We consider a permutation with n elements. We want to sort (ascendingly) this permutation by performing only operations of the following type: (case 1) we choose a position i $(0 \leq i \leq n)$ and: all the numbers on the positions $1, \ldots, i$ are rotated one position to the left (or right, or we can choose the direction of the rotation), and all the numbers on the positions $i + 1, \ldots, n$ are rotated one position to the right (or left, or we can choose the direction of the rotation); (case 2) we choose a position i $(0 \leq i \leq n + 1)$ and all the numbers on the positions $1, \ldots, i - 1$ are rotated 1 position to the left (or right, or we can choose the direction of the rotation), and all the numbers on the positions $i + 1, \ldots, n$ are rotated 1 position to the right (or left, or we can choose the direction of the rotation); if $1 \leq i \leq n$ then the element on the position i is not moved.

We will first consider case 1. We repeatedly choose position n, until element 1 is on the first position of the permutation. Then, the problem is solved incrementally. At step k $(2 \leq k \leq n)$, we have all the numbers from 1 to $k - 1$ on the positions $1, \ldots, k - 1$. We now want to bring number k on position k (unless it's already there). In order to achieve this we will perform the following steps:

1. We will repeatedly choose position 0, until number k is on the last position of the permutation.

2. Then, we repeatedly choose position $n-1$, until the numbers $1,\ldots,k-1$ are located on the positions $n-k+1,\ldots,n-1$ (at every rotation, number k remains on the last position of the permutation).
3. Then, we repeatedly choose position n, until the numbers $1,\ldots,k$ are located on the positions $1,\ldots,k$.

Case 2 can be solved using a similar strategy. At the first step we rotate the entire permutation, until element 1 is located on position 1 (e.g. by repeatedly choosing $i=n+1$). Then, at each step k ($2\leq k\leq n$), we choose position $i=0$ until element k reaches position n. Afterwards, we repeatedly choose the position $i=n$, until the numbers $1,\ldots,k-1$ are located on the positions $n-k+1,\ldots,n-1$. Then, we repeatedly choose $i=n+1$, rotating the entire permutation until the numbers $1,\ldots,k$ are located on the positions $1,\ldots,k$.

In both cases we have to perform $O(n^2)$ operations. If we actually perform the rotations at every operation, the overall time complexity of the algorithm becomes $O(n^3)$. In order to maintain the $O(n^2)$ time complexity, we notice that we do not actually have to perform every operation. For instance, if, during the algorithm, we choose p times consecutively the same position i, this is equivalent to rotating the two sides of the permutation by p positions. Thus, when we want to move a number k on the last position of the permutation we compute the number p_1 of operations which need to be performed ($p_1=pos(k)$ for rotations to the left, or $p_1=n-pos(k)$ for rotations to the right, where $pos(k)$ is the current position of element k in the permutation). Then, when we need to move all the numbers $1,\ldots,k$ from the positions $q,q+1,\ldots,q+k-1$ of the permutation to the positions $r,r+1,\ldots,r+k-1$ ($1,\ldots,k$) of the permutation and they are located in a part containing the positions $1,\ldots,s$ ($s=n-1$ or $s=n$), we will compute the number p_2 of operations which need to be performed: $p_2=q-r$ (if $q\geq r$) or $p_2=q+s-r$ (if $q<r$) for rotations to the left; $p_2=r-q$ (if $r\geq q$) or $p_2=s-q+r$ (if $r<q$) for rotations to the right. Then, for each such values p_1 and p_2 we just perform a rotation by a multiple number of positions (in $O(n)$) time. This way, we only perform $O(1)$ rotations per step, obtaining an $O(n^2)$ time complexity.

4.3. Sorting Permutations by Rotations - Version 2. We consider a permutation of the numbers $1,\ldots,n$ again. We want to sort this permutation ascendingly, by using the following type of operations: we choose a position p and: all the elements on the positions $1,\ldots,p-1$ are reversed, and all the elements on the positions $p+1,\ldots,n$ are also reversed. For instance, if the permutation is $Pe(1),\ldots,Pe(n)$ and we choose a position p, the new permutation will be: $Pe(p-1),Pe(p-2),\ldots,Pe(1),Pe(p),Pe(n),Pe(n-1),\ldots,Pe(p+1)$. We can also choose the positions $p=0$ and $p=n+1$. We will present a solution which performs $O(n)$ operations. At the first operation we will identify the position p on which element 1 is located. Then, we will choose the position $p+1$ (and we perform the operation). At the beginning of each step k ($2\leq k\leq n$) the numbers $1,\ldots,k-1$ will be located on the positions $1,\ldots,k-1$ of the permutation, but in reverse order. The steps to be performed are the following:

1. Let p be the position on which element k is located.
2. We perform an operation by choosing the position p. Now, the permutation contains the elements $1,\ldots,k$ on the positions $p-k+1,\ldots,p$, in increasing order.
3. After this, we will choose the position $p+1$, which will move the elements $1,\ldots,k$ on the first k positions, in reverse order.
4. In the end, we perform one extra operation: we choose the position 0.

The algorithm performs $O(n)$ operations and the total time complexity is $O(n^2)$ ($O(n)$ per operation).

4.4. Constrained Sorting of a Permutation by Swaps. We consider a permutation of the numbers $1,\ldots,n$. We want to sort this permutation ascendingly by performing swaps. A call $Swap(p_i,p_j)$ swaps the elements on the positions p_i and p_j of the permutation. However, we can only call $Swap(p_i,p_j)$ for certain pairs of positions (p_i,p_j).

A simple solution is the following. We construct a graph G in which every vertex corresponds to a position of the permutation and we have an edge between vertices i and j if the call $Swap(i,j)$ is permitted. The presented algorithm runs in n steps. At step i, all the numbers $1,\ldots,i-1$ are already on the positions $1,\ldots,i-1$ and we will bring element i on position i (without changing the positions of the elements $1,\ldots,i-1$). If element i is not already on position i at step i, then we perform the following actions:

1. We search for a path in G from the vertex corresponding to the current position p of the element i to the vertex corresponding to position i; let this path be $v_1=p,v_2,\ldots,v_k=i$
2. We perform, in this order, the calls: $Swap(v_1,v_2)$, $Swap(v_2,v_3),\ldots,Swap(v_{k-1},v_k)$ (i. e. we call $Swap(v_j,v_{j+1})$ for $1\leq j\leq k-1$ in increasing order of j)
3. We perform the following calls: $Swap(v_j,v_{j-1})$ for $2\leq j\leq k-1$ in decreasing order of j

We notice that after the swaps performed at step 2, element i arrives on position i, but many of the other elements may have been moved away from their positions. After performing step 3 however, all the elements are brought back to their original positions (i. e. those before the swaps at step 2), except for the element q which was previously located on position i, and which will now be located on position p. Step 1 can be implemented in $O(n+m)$ time (where m is the number of edges of G), using any graph traversal algorithm (e.g. DFS or BFS). Steps 2 and 3 are trivially implemented in $k = O(n)$ time. Thus, the total time complexity is $O(n \cdot (n+m))$. If we initially split G into connected components and maintain only a spanning of each connected component, then we can find a path between two positions p and i in $O(n)$ time (if it exists), because we will only consider the $O(n)$ edges of a spanning tree. Thus, the overall time complexity becomes $O(n^2)$.

4.5. Minimum Cost Sorting of a Permutation by Swaps. We consider a permutation $p(1), \ldots, p(n)$ of the numbers $1, \ldots, n$. We want to sort this permutation ascendingly by using swaps. Each number i $(1 \leq i \leq n)$ has a cost $c(i)$. The cost of swapping two elements x and y in the permutation is $c(x) + c(y)$. We want to find a minimum cost strategy which sorts the permutation.

We will construct a graph with n vertices, one for each number in the permutation, and n directed edges: $(i, p(i))$ $(1 \leq i \leq n)$. This graph is a union of disjoint cycles. We will consider every cycle Cy containing at least two vertices. For each vertex i on the cycle, let $next(i)$ be the vertex following i on Cy and $prev(i)$ be the vertex preceeding i on Cy. We have two options for bringing the values of the vertices on Cy on their corresponding positions.

The first choice is the following:

1. Let q be the vertex of Cy with the minimum cost $c(q)$.
2. We will repeatedly swap q with every vertex on Cy, starting from $prev(q)$ and moving in reverse direction of the edges along Cy.
3. Let SC be the sum of the costs of the vertices on Cy and let k be the number of vertices on Cy.
4. The cost of performing these swaps is $SC - c(q) + (k-1) \cdot c(q)$.

The second choice is the following:

1. Let r be the element with the smallest value $c(r)$ in the entire permutation.
2. We swap the elements q and r, and then we perform the same swaps as before, only using the element r instead of q.
3. In the end, we swap q and r again (thus, q also reaches its final position).
4. The cost of this choice is $SC - c(q) + (k-1) \cdot c(r) + 2 \cdot (c(q) + c(r))$ (where SC and k have the same meaning as before).

For each cycle Cy we will select the choice with the smallest cost. The time complexity of the algorithm is $O(n)$.

4.6. Circular Sorting of a Multi-Permutation. We consider a multi-permu-tation $p(1), \ldots, p(n)$, in which every number from 1 to k occurs at least once. We want to *circularly sort* the multi-permutation by performing swaps. The multi-permutation is circularly sorted if it is a circular permutation of the multi-permutation in which all the numbers of p are sorted ascendingly. In order to sort the multi-permutation we can perform swaps. The cost of swapping the numbers on the positions i and j is $|i-j|$. We want to minimize the total number of swaps first and, if there are multiple strategies with the same number of swaps, we want to minimize the total cost of the performed swaps.

We will first use count-sort in order to construct in $O(n+k)$ time the ascending order of the values: $a(1), \ldots, a(n)$. Then, we will consider every circular permutation of $a(1), \ldots, a(n)$. Let $q(1), \ldots, q(n)$ be such a circular permutation. We will compute the best strategy (minimum number of swaps and minimum total cost) in order to sort the multi-permutation p in the order given by q.

For every value i $(1 \leq i \leq k)$ we will compute the list $Lp(i)$ of positions on which it occurs in p and the list $Lq(i)$ of positions on which it occurs in q. The lists $Lp(i)$ and $Lq(i)$ are sorted ascendingly. Both sets of lists ($Lp(*)$ and $Lq(*)$) can be constructed in overall linear time, as follows: we traverse the multi-permutation p (q) from the position 1 to n and we add each position i at the end of $Lp(p(i))$ ($Lq(q(i))$) (all the lists are empty before the traversal). The lists $Lp(i)$ need only be computed once, in the beginning (before considering the first circular permutation q).

Then, we will initialize two variables: $ni = 0$ and $ci = 0$. For each value i $(1 \leq i \leq k)$ we will merge the lists $Lp(i)$ and $Lq(i)$ into a list $Lr(i)$. Then, we will traverse the list $Lr(i)$ from the beginning to the end. Every time we find two consecutive equal elements x in $Lr(i)$, we add x to another list $Lc(i)$ (which is empty before

we start traversing the list $Lr(i)$). Then, we will "merge" the lists $Lp(i)$ ($Lq(i)$) and $Lc(i)$ into a list $Lp'(i)$ ($Lq'(i)$). During the merge, as long as we haven't reached the end of any of the two lists, if the current element of $Lp(i)$ ($Lq(i)$) is equal to the current element of $Lc(i)$ then we do not add any of them to $Lp'(i)$ ($Lq'(i)$) and we simply move to the next elements; if they are different, then we add the current element of $Lp(i)$ ($Lq(i)$) to the end of $Lp'(i)$ ($Lq'(i)$). In the end, if any elements from $Lp(i)$ ($Lq(i)$) were not considered before considering all the elements of $Lc(i)$, we add these elements, in order, to the end of $Lp'(i)$ ($Lq'(i)$).

The lists $Lp'(i)$ and $Lq'(i)$ are sorted ascendingly and contain the positions *pos* on which the element i occurs in p and, respectively, in q, but it does not occur on the same position in the other multi-permutation (q, respectively p). Let $Q(i, j)$ denote the j^{th} element of a list $Q(i)$. We set ni to the number of elements in $Lp'(i)$. Then, we traverse the elements of $Lp'(i)$ and we increase ci by $|Lp'(i, j) - Lq'(i, j)|$ (j is the current position of the element from $Lp(i)$; initially, $ci = 0$). ni is the minimum number of required swaps and ci is the minimum total cost (for performing ni swaps). If we were only interested in the minimum total cost, then we would use $Lp(i)$ ($Lq(i)$) instead of $Lp'(i)$ ($Lq'(i)$) when computing the cost ci.

We will maintain the pair (ni, ci) with the minimum value of ni (and, in case of ties, with the minimum value of ci). The time complexity is linear for each multi-permutation q. Since we consider n multi-permutations q, the overall time complexity is $O(n^2)$.

4.7. Sorting a Multi-Permutation by Swapping Adjacent Elements. We consider two multi-permutations with n elements: $p(1), \ldots, p(n)$ and $q(1), \ldots, q(n)$. Both p and q contain values from 1 to k and, moreover, they contain the same number of values i ($1 \leq i \leq k$), i. e. q is obtained from p by permuting its numbers somehow. We want to transform p into q by performing the following type of move: we can select a position i ($1 \leq i \leq n-1$) and swap the values $p(i)$ and $p(i+1)$. We want to compute the minimum number of swaps required to transform p into q.

We will construct a permutation r of the numbers $1, \ldots, n$, as follows. We first traverse the permutation q from position 1 to n and, for every position i, we add i at the end of a list $L(q(i))$ (we maintain k lists $L(*)$ which are initially empty). Then, we initialize an array $idx(i)$ ($1 \leq i \leq k$) to 0 and we traverse the permutation p from the first to the n^{th} position. For every position i ($1 \leq i \leq n$) we: (1) increment $idx(p(i))$; (2) set $r(i) = L(p(i), idx(p(i)))$ (we denote by $L(u, v)$ the v^{th} element of the list $L(u)$). The minimum number of swaps required to turn p into q is equal to the number of inversions of the permutation r.

We can compute the number of inversions of a permutation r using several algorithms. The first algorithm consists of extending the merge-sort algorithm. We will maintain a global variable *ninv* (which is initially 0). Let's assume now that, during the merge-sort algorithm, we need to merge two sorted sequences, A and B (with x and, respectively, y elements), where A consists of elements located to the left of the elements in B within p. During the merging phase, we maintain two counters i and j (initialized at 1), representing the current position in the sequences A and, respectively, B. If $A(i) < B(j)$ we add $A(i)$ at the end of the sorted sequence and we increment i by 1; otherwise, we add $B(j)$ at the end of the sorted sequence, we increment j by 1 and we increment *ninv* by $(x - i + 1)$. When either i exceeds x or j exceeds y, the remaining elements are added to the end of the sorted sequence. In the end, the number of inversions of r is *ninv*. The time complexity of this approach is $O(n \cdot \log(n))$.

A second algorithm consists of using a segment tree. Each of the n leaves of the segment tree will have the values 1. Inner nodes contain the sum of the values of the leaves in their subtrees. We will traverse the permutation r from the position 1 to position n. Like before, we will maintain the variable *ninv* (initialized to 0). For a position i, we query the segment tree in order to find the sum S of the leaves from the interval $[1, r(i) - 1]$. We will increment *ninv* by S and, after this, we set the value of the leaf $r(i)$ to 0 (also updating the values of the leaf's ancestors). The time complexity of this approach is also $O(n \cdot \log(n))$. If instead of the segment tree we use a block partition (in which every position from 1 to n has an initial value of 1 and, after considering the position i, we set the value on the position $r(i)$ to 0), we can obtain a time complexity of $O(n \cdot sqrt(n))$ (we denote by $sqrt(x)$ the square root of x).

The original problem can be solved with a linear ($O(n)$) time complexity if $k \leq 2$. We construct the lists of positions $Lp(1)$ and $Lq(1)$ on which the value 1 occurs in p and, respectively, q. These lists are sorted ascendingly and can be constructed in $O(n)$ time, as we discussed earlier. Then, the total number of required swaps is equal to the sum of the values $|Lp(1, i) - Lq(1, i)|$ (where $1 \leq i \leq$ the total number of values equal to 1 in p).

Note that in the case of the problem discussed in this section we can find the minimum number of swaps in $O(n \cdot \log(n))$ time, but this number may be of the order $O(n^2)$. The strategy performing the swaps is quite

straightforward. For every position i from 1 to n, if $p(i) \neq q(i)$ we will find the element $p(j) = q(i)$ on the smallest position $j > i$ and we swap it repeatedly with the element to its left, until it arrives on the position i.

4.8. Sorting a Multi-Permutation by Grouping Identical Elements. We consider a multi-permutation $p(1), \ldots, p(n)$, which contains every element i $(1 \leq i \leq k)$ at least once. We want to rearrange the numbers in the multi-permutation such that all equal numbers are located on consecutive positions (i. e. grouped together in a contiguous sequence). In order to achieve this we can swap any pair of adjacent values (i. e. we can choose two positions i and $i+1$ and swap the values $p(i)$ and $p(i+1)$; $1 \leq i \leq n-1$). We want to find a strategy which performs the minimum number of swaps.

A first solution is the following. We will consider every possible permutation of the k distinct values (there are $k!$ such permutations). Then, let this permutation be $q(1), \ldots, q(k)$. We will compute the minimum number of swaps required to bring all the values equal to $q(1)$ first, then all the values equal to $q(2)$, and so on.

Before considering any permutation, we will compute the values $cnt(*)$, where $cnt(j)$=the number of elements equal to j in the multi-permutation p (we have $cnt(1) + \ldots + cnt(k) = n$). Then, for a given permutation $q(1), \ldots, q(k)$, we will compute the values $scnt(q(i)) = cnt(q(1)) + \ldots + cnt(q(i-1))$ ($scnt(q(1)) = 0$ and $scnt(q(2 \leq i \leq k)) = scnt(q(i-1)) + cnt(q(i-1))$ (these values are computed in the order given by the permutation q). Then, we initialize a set of values $idx(1 \leq i \leq k)$ to 0 and we traverse the multi-permutation p (from $i = 1$ to n). During the traversal we will construct a permutation r. For each position i of p, we will first increment $idx(p(i))$ by 1 and then we set $r(i) = scnt(p(i)) + idx(p(i))$. In the end, r is a permutation and the minimum number of swaps required to sort the multi-permutation p according to the constraints of the permutation q is equal to the number of inversions of r. Computing the number of inversions of a permutation with n elements can be performed in $O(n \cdot \log(n))$ time, as was shown in the previous subsection. Thus, we obtained an algorithm with a time complexity of $O(k! \cdot n \cdot \log(n))$.

We can improve our solution as follows. In the beginning we will compute a $k - by - k$ matrix num, where $num(a, b)$=the number of pairs of positions (i, j), such that $p(i) = a$, $p(j) = b$, and $i < j$. This matrix can be easily computed in $O(n^2)$ time. We simply initialize it to all zeroes and then we consider every pair of positions (i, j) $(1 \leq i < j \leq n)$ and increment $num(p(i), p(j))$ by 1. However, we can compute it more efficiently. We initialize the matrix to zero and then we traverse the multi-permutation from the position 1 to n. We will maintain an array cnt, where $cnt(j)$=the number of values equal to j encountered so far. Initially, $cnt(1 \leq j \leq k) = 0$. For every position i $(1 \leq i \leq n)$ we will consider every value j $(1 \leq j \leq k)$ and we will increment $num(j, p(i))$ by $cnt(j)$; after this, we increment $cnt(p(i))$ by 1. Thus, we computed the num matrix in $O(n \cdot k)$ time.

After computing the matrix $num(*, *)$ we will consider again all the possible $k!$ permutations $q(1), \ldots, q(k)$ of the distinct values of p. For a given permutation $q(1), \ldots, q(k)$, the minimum number of swaps required to sort the multi-permutation p according to the constraints imposed by the permutation q is equal to the sum of all the values $num(q(j), q(i))$ with $1 \leq i < j \leq k$. Thus, we obtained a solution with an $O(n \cdot k + k! \cdot k^2)$ time complexity. If we generate the $k!$ permutations in the Steinhaus-Johnson-Trotter order (also called *transposition order*), then two consecutive permutations differ from each other in exactly two consecutive positions i and $i+1$. Thus, for the first generated permutation we use the algorithm described above. Then, when we generate a new permutation $q(1), \ldots, q(k)$ which differs from the previously generated permutation on the positions i and $i+1$, we will compute the number of swaps as follows. Let V be the number of swaps for the previous permutation. The number of swaps V' for the current permutation will be equal to $V' = V + num(q(i+1), q(i)) - num(q(i), q(i+1))$. Thus, the time complexity is now $O((n+k) \cdot k + k!)$ (if we generate every new permutation in $O(1)$ (amortized) time).

Another solution, also based on computing the matrix $num(*, *)$ is to use dynamic programming. We will compute the values $nmin(S)$=the minimum number of swaps required to bring the values belonging to the set S in some order before all the other values of the multi-permutation (and ignoring those values which are not part of S). S is a subset of $\{1, \ldots, k\}$. We will consider the subsets S in increasing order of their number of elements (or in lexicographic order). We have $nmin(\{\}) = 0$. For $|S| > 0$ we proceed as follows. For every element i from S we will consider the case when the values equal to i are placed last (after all the other values in S). We will compute $nmin'(S, i) = nmin(S \setminus \{i\})$ plus the sum of the values $num(i, j)$, where $j \in S$ and $j \neq i$. We have $nmin(S) = \min\{nmin'(S, i) | i \in S\}$. The final result is $nmin(\{1, \ldots, k\})$. The time complexity of this solutions is $O(n \cdot k + 2^k \cdot k^2)$.

This time complexity can be slightly improved as follows. Initially, we will compute the values $Sum(S, i)$ for every subset S and every element $i \in S$, representing the sum of the values $num(i, j)$ for $j \in S$ and

$j \neq i$. We will compute these values in increasing order of the elements of S (or in lexicographic order of the subsets S). We have $Sum(\{i\}, i) = 0$. For $|S| \geq 2$ let $j \neq i$ be an arbitrary element of S. We have $Sum(S, i) = Sum(S \setminus \{j\}, i) + num(i, j)$. This takes $O(k \cdot 2^k)$ time overall. Thus, in the algorithm from the previous paragraph we can compute $nmin'(S, i)$ in $O(1)$ time instead of $O(k)$ by using the value $Sum(S, i)$.

5. Set Maintenance based on the Union-Find and Split-Find Problems. The Union-Find problem consists of supporting efficiently the following two operations on a family of disjoint sets: *Union(A,B)* performs the set union of the sets identified by A and B; *Find(x)* returns the identifier of the set which contains the element x. The Union-Find problem has been studied extensively, due to its wide range of applications. In this section we propose several simple extensions, which, nevertheless, are important from a practical point of view.

We consider the elements of each set arranged in a row. At first, we have n elements, identified with numbers from 1 to n; each element i forms a set on its own and has a weight $w(i)$. We consider a sequence of two types of operations: *union* and *query*. A union specifies the identifiers of two elements x and y and a direction d (*left* or *right*). Let $rx = Find(x)$ be the identifier of the set containing x and $ry = Find(y)$ be the identifier of the set containing y. The union operation combines the two sets into a single set, in the following way. If $d = left$, then the elements of the set rx are placed to the left of those in the set ry; otherwise, the elements of the set rx are placed to the right of those in the set ry. A query specifies an element x and asks for the the aggregate weight of the elements y in the same set as x which are located (strictly) to the left of x (using a pre-specified aggregation function $aggf$).

We will represent the sets as rooted trees. The root of each tree will be the representative element (the identifier) of the set. For each element x we store its parent in the tree ($parent(x)$), a value $wp(x)$ and the aggregate weight of the elements in its subtree ($wagg(x)$). We first present a solution for the case when $aggf$ has an inverse $aggf^{-1}$. We initialize $parent(x)$ to $null$, $wp(x)$ to the neutral element of $aggf$ (e.g. 0, for $aggf = +$, xor; 1 for $aggf = *$) and $wagg(x)$ to $w(x)$ for each element x.

For a query operation with the argument x, the answer will be the aggregate of the $wp(y)$ values, where y is an ancestor of x in its tree (including x).

For a union operation, we compute the representatives of the sets containing x and y, rx and ry, by following the parent pointers all the way up to the tree roots. We will use the *union by rank* heuristic. If the height (or total number of nodes) of the tree rooted at rx is smaller than or equal to that of the tree rooted at ry, we set $parent(rx) = ry$; otherwise, we set $parent(ry) = rx$. After setting the parent pointer of one of the two representatives, we consider the direction d of the union:

- If $d = left$ and $parent(rx) = ry$, we set $wp(ry) = wp(ry)\ aggf\ wagg(rx)$ and, after this, $wp(rx) = wp(rx)\ aggf\ (wp(ry))^{-1}$.
- If $d = left$ and $parent(ry) = rx$, then we set $wp(ry) = (wp(ry)\ aggf\ wagg(rx))\ aggf\ (wp(rx))^{-1}$.
- If $d = right$ and $parent(rx) = ry$, then we set $wp(rx) = (wp(rx)\ aggf\ wagg(ry))\ aggf\ (wp(ry))^{-1}$.
- If $d = right$ and $parent(ry) = rx$, then we first set $wp(rx) = wp(rx)\ aggf\ wagg(ry)$; then, we set $wp(ry) = wp(ry)\ aggf\ (wp(rx))^{-1}$ (considering the updated value of $wp(rx)$).

In the end, if $parent(rx) = ry$ then we set $wagg(ry) = wagg(ry)\ aggf\ wagg(rx)$; otherwise, we set $wagg(rx) = wagg(rx)\ aggf\ wagg(ry)$.

Since we use the union by rank heuristic, the height of every tree is $O(\log(n))$. Thus, performing a query takes $O(\log(n))$ time (because an element has $O(\log(n))$ ancestors). We can improve the query time by also using the *path compression* heuristic. The path compression heuristic works as follows. Every time we need to traverse all the ancestors of an element x (from x towards the root of its tree), we change the tree and make $parent(y) = rx$, where rx is the root of element x's tree and y is on the path between x and rx (including x and excluding rx); when doing this, we need to take care of also changing the values $wp(y)$. Let's assume that the path from x to rx consists of the elements: $v_1 = x$, $v_2 = parent(x)$, $v_3 = parent(parent(x))$, ..., $v_h = rx$ (the root). During a *Find(x)* call or a query with argument x, we compute $wpagg(i) = wp(v_i)\ aggf\ \ldots\ aggf\ wp(v_{h-1})$ $(1 \leq i \leq h-1)$ ($wpagg(1 \leq i \leq h-2) = wp(v_i)\ aggf\ wpagg(i+1)$ and $wpagg(h-1) = wp(v_{h-1})$). We set $parent(v_i)$ to rx and we set $wp(v_i)$ to $wpagg(i)$ $(1 \leq i \leq h-1)$. The overall (amortized) time complexity becomes $O(n \cdot \alpha(m, n))$, where $m(\geq n)$ is the total number of operations and $\alpha(m, n)$ is the inverse of the Ackermann function.

We can also use the path compression technique without the complex union rules (and/or heuristics), for the general case where $aggf$ is only commutative and associative (e.g. $aggf = \max$, $\min$). We will maintain the $wagg(x)$ values as before. Each edge $(u, parent(u))$ of a tree will have a value $wskip(u)$. When we unite two sets

with representatives rx and ry such that the set rx will be located to the left of the set ry, we set $parent(ry) = rx$ and set $wskip(ry) = wagg(rx)$. Afterwards, we update $wagg(rx)$ (we set it to $wagg(rx)\ aggf\ wagg(ry)$). The aggregate weight of the elements located strictly to the left of an element x is the aggregate of the $wskip$ values of the edges on the path from x to its set representative (tree root). At every $Find(x)$ (or query with x as an argument) operation, we compute the set representative rx and then the values $wskipagg(y) = wskip(y)\ aggf\ wskipagg(parent(y))$ (where y is on the path from x to rx, including x; $wskipagg(rx) = undefined$, i. e. it is the neutral element of the aggregation function). Then, we set $parent(y) = rx$ for every node y on the path from x to rx (including x, if $x \neq rx$, and excluding rx), as well as $wskip(y) = wskipagg(y)$.

The solution presented so far for this problem considered the online case (i. e. every query and union operation was handled as soon as it was received). In the offline case we can construct a somewhat simpler solution. We will first process all the union operations (in order), ignoring the queries. Like before, we will maintain disjoint sets with a tree structure. For each set with its representative rx we will maintain $leftmost(rx)$ and $rightmost(rx)$, representing the index of the leftmost and rightmost elements in the set. Initially, we have $leftmost(x) = rightmost(x) = x$ for every element x. When we perform a union and the set identified by rx is placed to the left (right) of the set identified by ry, we will add a directed edge from $rightmost(rx)$ to $leftmost(ry)$ (from $rightmost(ry)$ to $leftmost(rx)$). After this, if we need to set $parent(rx) = ry$ ($parent(ry) = rx$), then the new root r will be ry (rx). If rx was placed to the left of ry then we will have $leftmost(r) = leftmost(rx)$ and $rightmost(r) = rightmost(ry)$; otherwise, we will have $leftmost(r) = leftmost(ry)$ and $rightmost(r) = rightmost(rx)$.

After processing all the unions, we consider the graph composed of the n elements as vertices and the added directed edges. From every vertex there is at most one outgoing edge. Thus, the graph is the union of a set of disjoint chains (directed paths). We will arrange all the elements consecutivey in the order in which they appear on their paths. Then we will concatenate the orderings corresponding to each path, considering an arbitrary order of the paths. Thus, we obtain a permutation $p(1), \ldots, p(n)$, such that: if $p(i)$ is not the rightmost element in its set, then $p(i+1)$ is the element from its set which is immediately to its right. Then, we will construct a 1D data structure DS over the ordering of these elements which will allow us to answer range queries efficiently (a query consists of the aggregate weight of the elements $p(i), \ldots, p(j)$ whose positions are contained in a given range $[i, j]$). If the $aggf$ function is invertible, we can compute prefix "sums" in order to answer a query in $O(1)$ time. If $aggf$ is min or max we can preprocess the elements in order to answer range minimum (maximum) queries in $O(1)$ time. Otherwise, we can construct a segment tree over the n elements which will allow us to answer aggregate queries in $O(\log(n))$ time. Moreover, for each element i ($1 \leq i \leq n$) we will store its position $pos(i)$ in this ordering (i. e. $p(pos(i)) = i$). Then, we will process the entire sequence of operations from the beginning. We reinitialize the disjoint sets and, like before, we will maintain the $leftmost(*)$ and $rightmost(*)$ values. This time we will also process the queries. In the case of a query for an element x, we first compute rx the representative of the set containing x. The answer to the query is obtained by range querying DS with the range $[pos(leftmost(rx)), pos(x) - 1]$.

We will now present an extension of the Split-Find problem, which was brought to our attention by R. Berinde. There are n elements placed consecutively in a row (from 1 to n). Initially, they are all part of the same set (interval). We can perform two types of operations. The operations may split an interval into two intervals or undo a split (unite two intervals back into a larger interval). The only initial interval $[1, n]$ has color C. The $Split(i, k, C_{left}, C_{right})$ operation considers the interval $[i, j]$ starting at i and a position k ($i \leq k < j$). The interval $[i, j]$ is split into the intervals $[i, k]$ and $[k+1, j]$. The interval $[i, k]$ is colored with color C_{left} and the interval $[k+1, j]$ is colored using color C_{right}. The $Undo(k)$ operation considers a position k where an interval $[i, j]$ was previously split and unites the two intervals $[i, k]$ and $[k+1, j]$, thus forming the interval $[i, j]$ back. The interval $[i, j]$ will get the color it had before the split. Obviously, this operation can only be used if the intervals obtained after the corresponding *Split* exist (i. e. they have not been split further or, if they have, they were put back together). A third operation $Query(i)$ asks for the color of the interval starting at position i (if such an interval exists).

We will present here a solution which takes $O(1)$ time per operation, no matter how the operations are mixed into the sequence of operations, and uses $O(n)$ memory. We will maintain several arrays: *start*, where $start(i) = 1$ if an interval starts at position i (and 0, otherwise); *col*, where $col(i)$=the color of an interval starting at position i; *isplit*, *jsplit*, *csplit*, where the interval $[isplit(k), jsplit(k)]$ was the last interval split at the position k and $csplit(k)$=the color of the interval $[isplit(k), jsplit(k)]$ before the last split which had the

position k as a split parameter; jj, where $jj(i)$=the finish endpoint of the interval starting at i (it makes sense only if $start(i) = 1$). Initially, we have $start(1) = 1$, $start(i > 1) = 0$, $jj(1) = n$ and $col(1) = C$.

A $Split(i, k, C_{left}, C_{right})$ operation performs the following actions (assuming $start(i) = 1$):

1. $j = jj(i)$
2. $isplit(k) = i$
3. $jsplit(k) = j$
4. $csplit(k) = col(i)$
5. $start(k + 1) = 1$
6. $col(i) = C_{left}$
7. $col(k + 1) = C_{right}$
8. $jj(i) = k$
9. $jj(k + 1) = j$

A $Query(i)$ operation returns $col(i)$, if $start(i) = 1$, or $undefined$, otherwise. $Undo(k)$ performs three steps:

1. $start(k + 1) = 0$
2. $col(isplit(k)) = csplit(k)$
3. $jj(isplit(k)) = jsplit(k)$

We can also support an operation $Undo(k, C')$, having the same meaning as $Undo(k)$, except that the interval which is formed back will get the color C' instead of the color it had before the split. $Undo(k, C')$ contains the same steps as $Undo(k)$, except that step 2 is: $col(isplit(k)) = C'$. We can also define the operations $Undo'(i)$ and $Undo'(i, C')$, where i is the beginning of an interval which will be united back with the interval after it. $Undo'(i[, C'])$ is equivalent to calling $Undo(jj(i)[, C'])$.

6. Minimum Cost Activation and Deactivation of Resources in a Graph with Bounded Treewidth. We consider an undirected graph G with p vertices and m edges, together with a tree decomposition of G, whose (tree)width is bounded by a small constant tw. The tree decomposition T contains n nodes ($n = O(p)$). Every node X contains a subset $S(X)$ of vertices of G. The subsets of vertices of the nodes of any tree decomposition have the following properties:

- if a vertex u of G belongs to both $S(X)$ and $S(Y)$ then u belongs to the subsets $S(Z)$ of every node Z on the path between X and Y in T
- for every edge (u, v) of G there exists at least one subset $S(X)$ such that both u and v belong to $S(X)$
- the size of every subset $S(X)$ is at most tw

Every vertex u of G has an associated resource which can be in one of the following two states: *active* or *inactive*. The initial state of the resource at a vertex u is $I(u)$ (1 for *active*, or 0 for *inactive*). The final desired state of the resource at a vertex u is $F(u)$. In order to change the states of the resources, we can repeatedly perform the following action: we can select a vertex u of G and change the state of the resource at u, as well as the states of the resources of all the neighbors v of u. Changing the state of a resource means bringing it into the state opposite from the current one (i. e. from active to inactive, or from inactive to active). Selecting a vertex u for performing the action incurs a cost $C(u) \geq 0$. We want to find a strategy which brings every resource into its final state and which incurs a minimum total cost.

Solutions for general graphs (without bounded treewidth), as well as for several particular graphs have been discussed in [17]. However, none of those solutions can match the time complexity of the algorithm we will present in this section.

The first observation is that we never need to select a vertex u more than once. Thus, a vertex u is *selected* if it was selected once, and *not selected* otherwise.

We will start by presenting a solution for the case in which G is a tree. In this case we do not need to use the tree decomposition T. We will choose an arbitrary vertex r as the root of G, thus defining parent-son relationships. For every vertex u from G we will compute the values $Cmin(u, state, sel)$=the minimum total cost for bringing all the resources within vertex u's subtree to their final states (except possibly for the resource at vertex u), such that (the resource at) vertex u is in the state *state* and vertex u has been selected (if $sel = 1$) or not (if $sel = 0$). These values will be computed bottom-up.

For a leaf vertex u we have $Cmin(u, I(u), 0) = 0$, $Cmin(u, 1 - I(u), 1) = C(u)$ and $Cmin(u, I(u), 1) = Cmin(u, 1 - I(u), 0) = +\infty$. For an inner vertex u we will compute the required values as follows. Let $ns(u)$ be the number of sons of the vertex u and let $s(u, j)$ be the j^{th} son of the vertex u, in some arbitrary or-

der ($1 \leq j \leq ns(u)$). We will first consider the cases with $sel = 0$. We will compute the values $Sum(u,k)$ ($k = 1, 2$) as the sum of the values $\min\{Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, 0), Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, 1)\}$ with $1 \leq j \leq ns(u)$. We also compute $NumSel(u,k)$ as the number of sons $s(u,j)$ for which $\min\{Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, q) | q = 0, 1\} = Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, 1)$ ($1 \leq j \leq ns(u)$). $NumSel(u,k)$ is the number of selected sons which contribute to the sum $Sum(u,k)$. Let $DifMin(u,k) = \min\{|Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, 1) - Cmin(s(u,j), (F(s(u,j)) + k) \mod 2, 0)| | 1 \leq j \leq ns(u)\}$.

We have $Cmin(u, (I(u) + NumSel(u,0)) \mod 2, 0) = Sum(u,0)$ and $Cmin(u, (I(u) + NumSel(u,0) + 1) \mod 2, 0) = Sum(u,0) + DifMin(u,0)$.

For the case $sel = 1$ we have $Cmin(u, (I(u) + NumSel(u,1) + 1) \mod 2, 1) = Sum(u,1) + C(u)$ and $Cmin(u, (I(u) + NumSel(u,1)) \mod 2, 1) = Sum(u,1) + DifMin(u,1) + C(u)$.

The minimum total cost for bringing every resource into its final state is $\min\{Cmin(r, F(r), q) | q = 0, 1\}$. The time complexity of the presented algorithm is $O(p)$ (i. e. linear in the number of vertices of G).

We will now return to our original problem. For every node X of T, let $num(X)$ be the number of vertices $u \in S(X)$ and let these vertices be $u(X,1), \ldots, u(X, num(X))$. We will compute the values

$$Cmin(X, (state(1), \ldots, state(num(X))), (sel(1), \ldots, sel(num(X)))),$$

representing the minimum total cost for bringing to their final states the resources of all the vertices of G belonging to subsets $S(Y)$ where Y is in node X's subtree (except possibly for the vertices $u(X,j)$, $1 \leq j \leq num(X)$), such that the resource in every vertex $u(X,j)$ has changed its state an even (if $state(j) = 0$) or odd (if $state(j) = 1$) number of times, and the vertex $u(X,j)$ has been selected (if $sel(j) = 1$) or not (if $sel(j) = 0$) ($1 \leq j \leq num(X)$).

For every node X and every combination $CSel = (CSel(1), \ldots, CSel(num(X)))$ we will compute

$$NumSel(X, i, CSel) \quad (1 \leq i \leq num(X))$$

as the number of vertices $u(X,j) \in S(X)$ ($1 \leq j \leq num(X)$) with $CSel(j) = 1$ and which are neighbors with the vertex $u(X,i)$, and $SumCSel(X, CSel)$ as the sum of the costs $C(u(X,j))$ of the vertices $u(X,j)$ with $CSel(j) = 1$ ($1 \leq j \leq num(X)$).

For each node X and each son Y of X, we define the set $Common(X,Y)$, containing those vertices belonging to the intersection of $S(X)$ and $S(Y)$: $v(Y,1), \ldots, v(Y, NumCommon(Y))$, where $NumCommon(Y)$ is the number of vertices in the set $Common(X,Y)$. If r is the root of T then we define $Common(parent(r), r) = \{\}$.

For every node X and for every combination $CSel' = (CSel'(1), \ldots, CSel'(NumCommon(X)))$ we will compute $NumSelCommon(X, i, CSel')$ as the number of vertices $v(X,j) \in Common(parent(X), X)$ ($1 \leq j \leq NumCommon(X)$) for which $CSel'(j) = 1$ and which are neighbors with $v(X,i)$. We will also compute $SumCSelCommon(X, CSel')$ as the sum of the costs $C(v(X,j))$ of the vertices

$$v(X,j) \in Common(parent(X), X) \quad (1 \leq j \leq NumCommon(X))$$

for which $CSel'(j) = 1$.

We will also maintain two hash tables at every node X of T. The first one will map every vertex $u(X,j)$ to its corresponding index in the set $Common(parent(X), X)$ (i. e. based on this hash table we will be able to find out if $u(X,j) \in Common(parent(X), X)$ and, if so, we will be able to find the index p associated to $u(X,j)$ such that $u(X,j) = v(X,p)$). The second hash table will map every vertex $v(X,j)$ to its corresponding index in the set $S(X)$ (i. e. based on this hash table we will be able to find the index q associated to $v(X,j)$ such that $v(X,j) = u(X,q)$). Every operation on each of the hash tables takes constant time.

Let's assume first that we computed all the values $Cmin(X, (*, \ldots, *), (*, \ldots, *))$ for a node X of T. After having these values computed, we will compute the values

$$CminCommon(X, CState', CSel') = \min\{Cmin(X, CState, CSel) | CSel'(j) = CSel(q)$$

and $CState'(j) = CState(q)$ (such that $v(X,j) = u(X,q)$) for every $1 \leq j \leq NumCommon(X)$, and $CSel(j') = 0$ or 1 for every vertex $u(X,j') \notin Common(parent(X), X)$ ($1 \leq j' \leq num(X)$), and $CState(j'') = ((F(u(X,j'')) + I(u(X,j''))) \mod 2)$ for every vertex $u(X,j'')$ such that $u(X,j'') \in S(X)$ and $u(X,j'') \notin Common(parent(X), X)$ ($1 \leq j'' \leq num(X)$) }. The easiest way to perform these computations is to first

initialize $CminCommon(X,(*,\ldots,*),(*,\ldots,*)) = +\infty$. Then, we will consider every possible pair of combinations $(CState, CSel)$. We extract $CState'$ from $CState$ and $CSel'$ from $CSel$ (by maintaining only those indices j for which $u(X,j) \in Common(parent(X),X)$ and reordering the values corresponding to those indices in the order corresponding to the indices of the vertices from the set $Common(parent(X),X)$ (the q^{th} component of $CState'$ and $CSel'$ corresponds to $v(X,q)$ $(1 \leq q \leq NumCommon(X))$). Then, we set $CminCommon(X,CState',CSel') = min\{CminCommon(X,CState',CSel'), Cmin(X,CState,CSel)\}$.

We will now show how to compute the values $Cmin(X,(*,\ldots,*),(*,\ldots,*))$ of every node X, when traversing the tree T bottom-up (from the leaves towards the root). For every node X we will first initialize $Cmin(X,(*,\ldots,*),(*,\ldots,*)) = +\infty$. Then, we will consider every possible combination $CSel$ and we will compute the values $Cmin(X,(*,\ldots,*),CSel)$.

If X is a leaf in T, then the state of the resource from a vertex $u(X,j)$ is $(I(u(X,j))+NumSel(X,j,CSel)+CSel(j)) \mod 2$. Thus, we will set $Cmin(X,CState = (CState(i) = (NumSel(X,i,CSel)+CSel(i)) \mod 2 \mid 1 \leq i \leq num(X)), CSel) = SumCSel(X,CSel)$.

If the node X is not a leaf in T, then let $Y(1),\ldots,Y(ns(X))$ be the $ns(X)$ nodes which are the sons of X (in an arbitrary order). We will compute the values $Cmin'(X,j,CState,CSel)$=the minimum total cost of bringing into their final states the resources of all the vertices of G located in the nodes Z, where either $Z = X$ or Z is a descendant of one of the nodes $Y(q)$ (including $Y(q)$) $(1 \leq q \leq j)$, except possibly for the resources of the vertices of $S(X)$, whose states were changed a number of times whose parity is defined by $CState$. $CSel$ is the combination we fixed earlier.

For every combination $CState = (CState(i) = (NumSel(X,i,CSel) + CSel(i)) \mod 2 | 1 \leq i \leq num(X))$, we will set $Cmin'(X,0,CState,CSel) = SumCSel(X,CSel)$; for the other possible combinations $CState$ we will set $Cmin'(X,0,CState,CSel) = +\infty$. After this, we will consider the sons $Y(j)$ of X, in increasing order of j. We will first initialize all the values $Cmin'(X,j,(*,\ldots,*),CSel) = +\infty$. Then, we will construct a new combination $CSel'$ obtained by removing from $CSel$ the components q' corresponding to those vertices $u(X,q')$ which do not belong to $Common(X,Y(j))$. Then, the remaining components are ordered such that the q^{th} component of $CSel'$ refers to $v(Y,q)$ $(1 \leq q \leq NumCommon(Y(j)))$.

After this, we will consider all the $2^{NumCommon(Y(j))}$ possible combinations (tuples)

$$CState' = (CState'(1),\ldots,CState'(NumCommon(Y(j)))).$$

For each combination $CState'$ we will consider every possible combination

$$CState = (CState(1),\ldots,CState(num(X))).$$

For each such combination $CState$ we will construct a new combination $CState''$, where:

$$CState''(i) = CState(i),$$

if $u(X,i) \notin Common(X,Y(j))$, and

$$CState''(i) = (CState(i) + CState'(q) + NumSelCommon(Y(j),q,CSel')) \mod 2,$$

if $u(X,i) \in Common(X,Y(j))$ and $u(X,i) = v(Y(j),q)$ $(1 \leq i \leq num(X))$.

Then, we will set

$$\begin{aligned} Cmin'(X,j,CState'',CSel) = \min\{&Cmin'(X,j,CState'',CSel), Cmin'(X,j-1,CState,CSel) \\ &+ CminCommon(Y(j),CState',CSel') - SumCSelCommon(Y(j),CSel')\}. \end{aligned}$$

In the end, we will have $Cmin(X,CState,CSel) = Cmin'(X,ns(X),CState,CSel)$ (for every possible combination $CState$).

The minimum cost we are looking for is $\min\{Cmin(r,CState = (CState(i) = (I(u(r,i)) + F(u(r,i))) \mod 2 \mid 1 \leq i \leq num(r)),*)\}$, where r is the root node of T. With a careful implementation we can obtain an $O(n \cdot tw \cdot 2^{3 \cdot tw})$. If we precompute all the transformations for each (generic) pair $t_1(CSel,$ *subset of indices*$) \rightarrow CSel'$ and $t_2(CState,$ *subset of indices*$) \rightarrow CState'$, and we establish a consistent ordering for the vertices in each subset $S(X)$ and $Common(U,V)$ (e.g. the vertices are ordered increasingly according to their identfiers), then we can obtain a time complexity of $O(n \cdot 2^{3 \cdot tw})$.

7. Related Work. Data transfer optimization problems have been considered in many papers, because of their highly important practical applications. References [1] and [2] present offline and online algorithms for several multicriteria data transfer optimization problems (e.g. deadline-constrained data transfer scheduling). [8] considers the optimal scheduling of two communication flows on multiple disjoint paths in order to minimize the makespan. Applications of several data structures to resource reservations were discussed in [6]. The technique for computing optimal average subsets of edges (or vertices) has been mentioned (in a similar form) in [4].

String and permutation sorting problems have also been considered from multiple perspectives and considering various constraints: [3] considers the sorting of sequences by interchange operations, while [7] considers sorting permutations by reversal operations.

A permutation sorting problem related to the ones mentioned in this paper was proposed as a task at the Baltic Olympiad in Informatics 2007: We have a permutation $p(1), \ldots, p(n)$ of the numbers $1, \ldots, n$. We want to sort this permutation in ascending order by using a sequence of the operations $Move(i, j)$ (which removes the element from position i in the permutation and inserts it at position j). The cost of an operation $Move(i, j)$ is $i+j$ and we are interested in sorting the permutation with a minimum total cost. The solution to this problem is based on two important observations: 1) every element is moved at most once; 2) the moved elements are moved in decreasing order of their values. These observations lead to a dynamic programming solution. We compute $Cmin(i, j)$=the minimum total cost of moving the elements $i, \ldots, n$ such that they are in their correct relative order and element i is located at position j. We have $Cmin(n+1, n+1) = 0$ and $Cmin(n+1, 1 \leq j \leq n) = +\infty$. We will compute the values $Cmin(i, *)$ in decreasing order of i. We will always have the possibility of moving the element i or of not moving it. However, when not moving element i, we will consider a compensation cost, such that elements smaller than i may consider that i was moved. We will start by computing the array $pos(k)$=the position on which element k is located in the original permutation ($1 \leq k \leq n$). Now, before computing the values $Cmin(i, *)$, we will first compute the value $pos'(i)$, which is equal to 1 plus the number of values k such that $1 \leq k \leq i-1$ and $pos(k) < pos(i)$ (because these are the only elements which are still to the left of element i in the permutation). We will consider that element i is moved right before the element $i+1$. Thus, we will consider all the values j from 1 to $n+1$, in increasing order. While traversing these values, we will maintain a variable $pdest$, which is initially equal to 1. When we reach a value j, we will first update $pdest$: if $p(j) < i$ then $pdest = pdest + 1$ (we consider $p(n+1) = n+1$). Then, we set $Cmin(i, j) = Cmin(i+1, j) + pos'(i) + pdest$. We will now consider the case when i is not moved. We will consider all the positions j from $pos(i)+1$ up to $n+1$ in increasing order; during this time, we will maintain a variable $extra\_cost$ (which is initially 0). When we reach such a position j, we set $Cmin(i, pos(i)) = min\{Cmin(i, pos(i)), Cmin(i+1, j) + extra\_cost\}$. Then, if $p(j) < i$ we will update the variable $extra\_cost$: $extra\_cost = extra\_cost + (i - p(j))$ (this is because there are $i - p(j)$ elements which will be moved before element i, and their contribution would not be considered otherwise). The minimum total cost is $\min\{Cmin(1, j) | 1 \leq j \leq n+1\}$.

The problem presented in Section 6 is a more general version of the minimum all ones problem [9]. Our dynamic programming state definition is similar to the one from [9], but our algorithm is substantially different.

8. Conclusions. The novel contributions of this paper can be classified into three categories:

- offline algorithms for (multi)point-to-(multi)point data transfer optimization
- offline algorithms for resource processing optimization
- the architecture of an agent-based peer-to-peer content delivery framework

The considered problems are either new, or extended (e.g. more general) versions of other existing problems. All the presented solutions make use of the specific properties of the considered problems, thus obtaining optimal or near optimal results.

The agent-based content delivery framework is focused on efficient multicast data distribution. Some of the presented offline algorithms could be used for computing a multicast data delivery tree, if information regarding the entire system is available. The framework is based on the peer-to-peer architectural model presented in [12], but the multicast extensions and the proposal for handling unconnectable peers are novel contributions of this paper.

As future work, we intend to consider online versions of some of the considered problems and adapt some of the offline solutions that we developed in order to obtain online techniques. These techniques could then be implemented by specialized agents for solving the problems in real-time.

REFERENCES

[1] B. B. Chen and P. V.-B. Primet, *Scheduling Deadline-Constrained Bulk Data Transfers to Minimize Network Congestion*, Proc. of the 7^{th} IEEE International Symposium on Cluster Computing and the Grid, pp. 410-417, 2007.
[2] M. S. Eltayeb, A. Dogan, and F. Ozguner, *Concurrent Scheduling: Efficient Heuristics for Online Large-Scale Data Transfers in Distributed Real-Time Environments*, Lecture Notes in Computer Science, vol. 3149, pp. 468-475, 2004.
[3] A. Amir, T. Hartman, O. Kapah, A. Levy, and E. Porat, *On the Cost of Interchange Rearrangement in Strings*, Lecture Notes in Computer Science 4698, pp. 99-110, 2007.
[4] D. Eppstein, *Choosing Subsets with Maximum Weighted Average*, Journal of Algorithms 24 (1), pp. 177-193, 1997.
[5] T.-H. Fan, S. Lee, H.-I. Lu, T.-S. Tsou, T.-C. Wang, and A. Yao, *An Optimal Algorithm for Maximum-Sum Segment and Its Applications in Bioinformatics*, Lecture Notes in Computer Science 2759, pp. 46-66, 2003.
[6] M. I. Andreica and N. Ţăpuş, *Efficient Data Structures for Online QoS-Constrained Data Transfer Scheduling*, Proc. of the 7^{th} IEEE Intl. Symp. on Par. and Distrib. Computing, pp. 285-292, 2008.
[7] H. Kaplam and E. Verbin, *Sorting signed permutations by reversals, revisited*, Journal of Computer and System Sciences, vol. 70, no. 3, pp. 321-341, 2005.
[8] M. I. Andreica and N. Ţăpuş, *Optimal Scheduling of Two Communication Flows on Multiple Disjoint Packet-Type Aware Paths*, Proc. of the 10^{th} IEEE Intl. Symp. on Symbolic and Numeric Algo. for Sci. Comput., pp. 137-144, 2008.
[9] Y. Lu and Y. Li, *The Minimum All-Ones Problem for Graphs with Small Treewidth*, Lecture Notes in Computer Science, vol. 4616, pp. 335-342, 2007.
[10] M. I. Andreica, *Techniques for the Optimization of Communication Flows in Distributed Systems*, CIBERNETICA MC Publishing House, Bucharest, 2010.
[11] T. S. E. Ng and H. Zhang, *Predicting Internet Network Distance*, Proceedings of the IEEE INFOCOM, pp. 170-179, 2002.
[12] M. I. Andreica, E.-D. Tîrşa, and N. Ţăpuş, *A Peer-to-Peer Architecture for Multi-Path Data Transfer Optimization using Local Decisions*, Proc. of the 3^{rd} Workshop on Dependable Distributed Data Management, pp. 2-5, 2009.
[13] M. I. Andreica, E.-D. Tîrşa, and N. Ţăpuş, *A Fault-Tolerant Peer-to-Peer Object Storage Architecture with Multidimensional Range Search Capabilities and Adaptive Topology*, Proc. of the 5^{th} IEEE Intl. Conf. on Intelligent Computer Communication and Processing, pp. 221-228, 2009.
[14] A. J. Mooij, N. Goga, and W. Wesselink, *A Distributed Spanning Tree Algorithm for Topology-aware Networks*, Proc. of the 2^{nd} Conf. on Design, Analysis, and Simulation of Distrib. Syst., pp. 169-178, 2004.
[15] M. Elkin, *A Faster Distributed Protocol for Constructing a Minimum Spanning Tree*, Proc. of the ACM-SIAM Symposium on Discrete Algorithms, pp. 359-368, 2004.
[16] M. I. Andreica, E.-D. Tîrşa, and N. Ţăpuş, *Data Distribution Optimization using Offline Algorithms and a Peer-to-Peer Small Diameter Tree Architecture with Bounded Node Degrees*, Proc. of the 17^{th} International Conference on Control Systems and Computer Science, vol. 2, pp. 445-452, 2009.
[17] M. I. Andreica, *Efficient Gaussian Elimination on a 2D SIMD Array of Processors without Column Broadcasts*, Politehnica University of Bucharest (UPB) Scientific Bulletin, Series C—Electrical Engineering and Computer Science, vol. 71, issue 4, pp. 83-98, 2009.

Edited by: Viorel Negru, Adina Magda Florea
Received: February 27, 2010
Accepted: March 31, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 19–32. http://www.scpe.org
ISSN 1895-1767

SENSORS DATA-STREAM PROCESSING MIDDLEWARE BASED ON MULTI-AGENT MODEL*

OVIDIU ARITONI† AND VIOREL NEGRU‡

Abstract. The goal of this study is to propose an architecture for an intelligent sensor data processing middleware. In order to fulfill the ambient assisted living data processing requirements we design a flexible and scalable architecture based on multi-agent model. This architecture allows acquisition, interpretation and aggregation of sensor data-streams. Our system is able to process different sensor data-streams, to adapt to different levels of abstraction, to define different data-processing workflows. An extended operators over data stream language is used to define workflows. Different types of agents (simple sensor agent, logic sensor agent, virtual sensor agent etc) are defined. The designed system is a domain independent multi-agent system which can be instantiated for particular AmI problems. The middleware architecture will use a healthcare system for validation.

Key words: ambient intelligence, ambient assisted living, context-aware sensitivity, healthcare systems, multi-agent model.

1. Introduction. The goal of an Ambient Intelligence (AmI) system is to response in an intelligent way to changes of context, providing intelligent services and adaptation to changing after the delivery [5]. These services include the control of the actuators integrated in the application, but also the acquisition of contextual information. AmI integrates a set of concepts from mobile computing, intelligent sensor, artificial intelligence, service oriented architecture and reflexive and adaptive systems.

1.1. Ambient Assisted Living Systems. Ambient Assisted Living (AAL) systems provides services support for daily life based on context and the situation of the assisted person.

The healthcare context awareness systems are a distinct category of AAL systems. These systems are used in hospitals or houses to improve the patient's life and to provide some useful services for some specialists, such as the nurses and doctors, for a rapid intervention in the patient life.

In [6] is described a healthcare system which provides an intelligent bed that eases the patient life. This system knows the patient's or nurse's identity and displays some relevant information which depends on the person which accesses them. This system is also used to store the electronic records of the patients.

The system described in [28] uses another perspective in healthcare systems: helps the emergency team on their intervention in different situations. The application is used for the localization of the emergency team members, via some active badges, and the communication between them. The potential emergency team members are notified by the application to join the team and if one member is not reachable the system searches another doctor or nurse. Also if an emergency team member is reachable but is not able to join the team it can initiate an audio-video conference.

The MobileWARD project [21] describes an architecture to support the morning procedure from a hospital, being able to display a patient list and all the patient information. De facto MobileWARD project develops a prototype used as an electronic patient records database with some context-aware facilities.

An ubiquitous system used for home medication monitoring and assistance is presented in [12]. A prototype is proposed that uses the RFID technology for detecting the pill bottle position and weight. Another similar project [13] uses the mobile communication to send reminder messages to the patient and information about the patient medication to his doctor.

VirtualECare [11] is an agent oriented assisted living project that uses web services to provide resources management or monitoring facilities. A similar project that use a service-oriented approach is AMIGO [38].

1.2. Article Overview. The main goal of this article is to propose a middleware architecture that allows acquisition, interpretation and aggregation of sensor data-streams. These operations over data must be deployed at the sensors middleware in order to solve the rapid response challenge.

The sensor data and the operations over them are used for:

- directly and immediately alerting doctors and paramedics to attend the patient. This includes in some situations reasoning and interpretation over data-streams.
- storage and building some historical data-streams for diagnosis and future research;

*This work is partially supported by the FP7 project DEHEMS and PNII national project ASISTSYS.
†IeAT - E-Austria Institute Timisoara, România, (oaritoni@info.uvt.ro).
‡West University of Timisoara, Department of Computer Science, Timisoara, România, (vnegru@info.uvt.ro).

- building new data-stream using the received data and techniques such as completing or decompleting data-stream, event detection, inflexion point detection etc. (For details see section 3.6.1)
- detect anomalies or irregularities in the received data.

In order to fulfill the AAL data processing requirements we design a flexible and scalable architecture based on multi-agent model. Our system is able to process different sensor data-streams, to adapt to different levels of abstraction, to define different data-processing workflows.

We define different types of agents (simple sensor agent, logic sensor agent, virtual sensor agent etc). The designed system is a domain independent multi-agent system which can be instantiated for particular AmI problems.

The rest of this paper is organized as follows: the *section 2* presents the general requirements for sensor middleware oriented to AAL systems; the *section 3* describes the proposed architecture; the *section 4* describes the related work and the advantages of the proposed architecture; the *section 5* will conclude the results presented in this paper, and offer an overview of future research.

2. Requirements. The section presents the main problems occurred in the design and development of a healthcare context-aware sensor middleware. The design process of such systems must take in consideration many aspects: the design of wireless sensor network (WSN), the data models, the communication protocol between simple sensor node and the sink node, the middleware runtime support, the middleware services, the quality services mechanisms, the data management (including data acquisition, data processing and data storage), the information discovery and the resource management, the middleware support for interoperability with another systems, the capabilities for software auto-reconfiguration.

In order to respect the interoperability requirement all the received data must respect imposed standardization. There are ISO standards that describes biomedical data such as ISO / IEEE 11073-30300:2004 (that defines an IrDA-based transport profile for medical device communication) [2], ISO 13606-3:2009 (for the communication of part or all of the electronic health record (EHR) of a single identified subject of care between EHR systems, or between EHR systems and a centralized EHR data repository) [4], ISO 12967-1:2009 (provides guidance for the description, planning and development of new systems, as well as for the integration of existing information systems) [3] . The sensor middleware must validate the received data using these standards. Also, to improve the application flexibility, the middleware allows the specification of new standards.

WSNs are constrained by resources such as: bandwidth, computation, communication capabilities, energy, etc. The middleware is used to provide an efficient management of all wireless sensor node.

There are three approaches to implement data-storage [32, 37]: external-storage, local-storage and data-centric storage. The data centric storage is the most popular approach. The external approach is used to store the data in a database station outside the sensor network and the local approach for storing the data where they are generated. The data-centric storage is a compromise between these two approaches.

3. Proposed Architecture. This section describes the proposed architecture for the context-aware sensor data-stream middleware. The solution regards the improvement of flexibility and scalability of such middleware. The proposed design tries to fulfil the ambient intelligent and the healthcare context-aware systems requirements.

3.1. Wireless Sensor Network for Data Acquisition. The integration of ubiquitous computing in healthcare was first developed by a team of University of Karlsruhe where smart sensors are used together with personalized mobile computing systems [23]. They introduced the concepts of Body Area Network, Personal Area Network and Wide Area Network. The differences between them are provided by the area of monitoring and the communication between devices:

- Body Area Network (BAN) - the monitoring area consist in sensors and devices near the patient body.
- Personal Area Network (PAN) - the monitoring area is the patient's environment.
- Wide Area Network (WAN) - PAN are connected to the central server.

The assisted living systems use various data such as: human body temperature and humidity, EKG signal variations, blood pressure, breathing frequency, resident's activity of the daily living, pulse-oxymeter data, motion data etc. Our system uses a Wireless Body Area Network (WBAN) [20] which integrates intelligent monitoring devices. A WBAN is a set of physiological and / or environmental sensors an their number depends on the end-user application. Using a WBAN we propose a middleware that it is characterized by:

- support for sensor data acquisition and processing;
- support for reasoning, interpretation and decision over the sensor data;

- support for alert system;
- scalability;
- the flexibility and easy configuration in order to support a wide range of protocols and sensor nodes;
- integration in a wide range of software systems (house intelligent systems, vehicle monitoring systems, etc).

All the received data needs to be stored in a database, or to be use in different inference / aggregation processes or to be interpreted. The sensor data interpretation is an annotation, due to a Jess rule that has as left-side the sensor data value and as right-side the corresponding label. The sensor data interpretation corresponds to the first level of sensor data processing described in Joint Directors of Laboratory [25]. The inference means that we obtain new information using interpreted sensor data and applying over them inference rules. For example the human body temperature must to be stored in the database, only if there are some variations related to a temperature higher than 37 Celsius degree. The nurse or the doctor use the received data to make some diagnoses or predictions about the future patient's life, about his medical treatment, or about anything else that is important for him. To middleware gives the sensor data to doctors and / or nurses in a readable format, and to assure this it must to do some interpretation or inference.

The wireless sensor network is composed of sensor nodes, a sink node and the gateway. The WSN takes into consideration three layers (Figure 3.1).

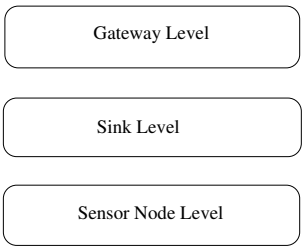


FIG. 3.1. *WSN Middleware Layers*

The *sensor node layer* is used to describe the communication protocol between the simple node and the *sink*. The node layer uses nesC [1] capsule programs which describe this protocol [15, 14]. The simple nodes only send regularly data gathered from the environment. There is no other responsibility for the sensor node that to send data.

The *sink layer* is used to receive data from sensor nodes, to build data-streams for each sensor node and to send them to the gateway.

The *gateway level* is used to collect the received data from all the sensor nodes and to provide some useful function such as filtering and storing data. Our middleware is developed for the gateway level. Our approach uses a simple perspective for the simple sensor node comparing with another WSN context-aware system which gives a lot of responsibilities to the node layer. The proposed system does not load the node layer or sink layer with a lot of functionalities, it uses intensively only the gateway level to process all the gathered data. There are other approaches that use the simple node layer for data-stream aggregation, interpretation, normalization, etc. In our approach all these data-stream operations take place at the gateway level. The context-aware healthcare system must respond in real-time and with a good accuracy. It is important to obtain quickly the vital signs of one patient in order to help him immediately. For this reason many operations that can take place at node level are deployed at the gateway level, and the only responsibility of node layer is to send data with the established frequency, or to receive query messages from the sink node. We adopt this approach because the node layer has many limitations about processing speed, memory, energy etc. Because there are these limitations, it is obvious to move a lot of responsibilities to the gateway level.

The gateway level receives streams and has the responsibilities to provide data which describe the context that can be used in our system. If it is necessary, the gateway level makes some data interpretation or applies some inference rules. For example, if at one instant the received temperature is 37.5, this temperature is translated by the middleware in "sub-feverish". Also if the moving detection sensor sends the value 0.05, this

[1] nesC is an extension of C programming language used to build applications for sensor nodes operating systems, such as TinyOS.

means that the patient in the emergency room is "sleeping" or is "watching TV". The middleware has the responsibility to detect some situations that can be translated in "alert".

Data interpretation means a translation of received streams in a relevant way for the application, or for the human user who does not understand the value of sensor received data. For better sensor data interpretation it is useful to use an ontology that provides support for semantic data annotation [36]. These ontologies are used also in the inference process over the interpreted data-streams. It is a necessary to interpret and inference the sensor data at the middleware level and not to demand special engines built for this purpose, because the system must quickly respond to the environment changes.

3.2. Multi-Agent Architecture. The architectural view of the gateway level is described in Figure 3.2.

The middleware builds agents for each sensor node. Each agent has as main goal the data-stream processing of the corresponding sensor. The use of a multi-agent architecture is motivated by the middleware complexity, adaptability and continuity requirements.

The data-stream is processing to obtain new and useful data, and in many cases is mandatory to do some abstraction. For each level of abstraction we use dedicated agents. We can have on this way an abstraction hierarchy.

A set of sensor data can describe a situation. Seng W. Loke in [26], formalizes and explains how to do reasoning about situations in order to obtain new data or to take correct decision. We can identify new situations continuously acquiring data about a person vital signs such as temperature, blood pressure, pulse frequency. The acquisition process is about the patient short-term, medium-term or long-term behavior. The agent "memory" store data to reasoning on short-term, medium-term or long-term. All these data will be interpreted, evaluated and compressed. The logical predicates of Seng W. Loke will be replaced in our architecture by a set of agents. These agents are used to identify states or state transitions of the patient from which the WBAN collects data. Examples of situations are:

```
(S1)Patient X is sleeping
(S2)Patient X has fallen down
(S3)Patient X is eating
(S4)Patient X has not respond to a call
```

To detect all of these situation there are agents that use some inference rule to obtain new data. The sensor data translation in understanding data is the first abstraction level. Detection of a simple situation, like in the above examples, is the second abstraction level. The simple situation describes a patient state, such as: is sleeping, has falling down. The simple situation can be described using only one inference over the sensor interpreted data. We can continue to do some inference over the last data, as example:

(Patient X has fallen down) AND (Patient X is sleeping) => (Emergency Situation about Patient X)

In the above situation we obtain a new data using another inference. In this way we use interpretations and inferences over the received sensor data in order to provide for the final user understandable and useful data. Identifying situations, the system does its job without human intervention.

The number of agents depends on the received sensor data and to the required abstraction level. The data-flow is changing and depends on the data itself. It is impossible to take in to consideration all the scenarios that can be happen in order to define data-processing workflows. The connections between these agents are not direct or predetermined. So, to respond on this indirect and dynamic environment a *multi-agent architecture* based on blackboard model [10, 30] is the appropriate solution.

The *blackboard* on our architecture is a shared memory that can be used to give an overview for all the agent. The blackboard contains temporary all the received data-streams. It's special goal is to store the incremental view about the patient context. Additionally to sensor received data, this repository contains also the result of data interpretation and aggregation produced by some agent behavior. We have a three-level blackboard. At the first level there are sensor received data. On the second level we find sensor received data interpreted. On the third level we find the inference data result such as: "Patient X is sleeping", "Patient X has not respond to a call", etc. The inference results over the interpreted data are stored on the third level. The blackboard contents is changing dynamically. The blackboard is a temporary buffer with a limited storage capability. The blackboard data that is never used for a long period of time is deleted. For each data-type we have define a time-limit. We use an algorithm to delete data, based on data relevance, insertion time, and the time-limit to be store on the blackboard. The sensor received data are stored on the blackboard for a short time (only a few seconds), and the interpreted or aggregated data are deleted only when there are not used for a long time period.

Our blackboard metrics are: the insertion-time and the relevance. For each data we store the insertion time on the blackboard contents, as the time when we received it from the sensors or from middleware agents. We define four relevance degrees:

- 0 – relevance for the sensor received data;
- 1 – relevance for the interpreted sensor data;
- 2 – relevance for the data obtain by using inference rules over the interpreted sensor data;
- 3 – relevance for the alert signals;

The less relevant data are the sensor received data-stream. The more abstract and useful is the most relevant data. The highest relevance is allocated to alert data. Also the blackboard keeps an index for all the data. When we insert a new data on the blackboard, we calculate an index value for it. This index is a unique number, and it is assigned for each new data entry. The blackboard uses a queue where the most relevant data are stored for a long time, and the non-relevant data are deleted from the queue.

We have defined rules that allow to build new agents. We have **Simple Sensor Receiver Builders** used to build agents only for sensor stream processing. Also, we have **Logic Sensor Receiver Builders** used to build agents that allows data aggregation / inference or interpretation. De facto, these builders are the knowledge sources for the blackboard. The result of each builder activations is a new agent. The builders are a set of rules that are activated when on the blackboard we find some special data. A special data corresponds to the builder rule left-side patterns. As example we have two builder: AmbientTemperatureBuilderAgent and HumanBodyTemperatureBuilderAgent. For the first builder a special data, is a number between -5 and 35, without decimals, and for the second is a value between 35 and 42, with one or two decimals. In this way the blackboard can be viewed as a reactive database. For example when we find on the blackboard temperature sensor data in the range 36-40 Celsius degree, the sensor receiver builder will define a Simple Sensor Agent for Biomedical Temperature. Agent's definitions are available on the middleware. An agent definition contains the situation when it must be build and it's core functionalities, scenarios and a list of ontologies that can be load. Using the matching mechanism over the blackboard data, the JESS rule engine is using to activate the building process. These builders or knowledge sources are developed using the direct communication with the final user or by learning. These builders contains some expertise about some types of data. These expertise contain patterns that allow the Jess rule left-side matching process. For the temperature data we can have a builder / knowledge source, for the patient state we can have another builder / knowledge source and in this way we can be sure that the middleware will be populated with a lot of agents that will process entirely all the data-stream from the backboard. Between the builder and the supervisor there is a publish-subscribe mechanism that allows the communication between them. The builders publish the models for the future agents on the blackboard and send a message to the supervisor to announce that.

The supervisor makes runtime decisions about the agents on the middleware. The supervisor has two main functionalities: first it must create agents using the builder proposals, and also it must to manage the correspondence between blackboard data and the middleware agents. Let's consider the situation when for the same data-stream we have in the middleware two or maybe three agents. The supervisor will solve also the conflicts that can appear (related to the blackboard access, agents with similar competencies etc). The supervisor will take decisions about the agent that will survive and about the agents that is redundant and must be killed. Also when a data-stream is broken the supervisor will kill the corresponding agent. The supervisor decides also what agent to create when the builder makes multiple proposals. The supervisor is shared to all the builders, and in this way we assure the communication between the builders and supervisor. The supervisor subscribes to all the builders, and the builders send messages to them. Using these messages the supervisor is informed that the builders have doing some proposal about the future middleware agents. Each builder proposal it is stored on the blackboard, and the supervisor read all these proposals and decides which agent will be built. This happens when the builder has time for storing its proposal on the blackboard. There are situations when is no time for proposals storing and supervisor decision. For the current cost sensor data is useful to issue proposals and to choose from these proposals, but for the blood pressure sensor data this is not a good practice.

3.3. Simple Sensor Agent, Logic Sensor Agent, Virtual Sensor Agent. At the gateway level, we have data aggregation or the inference process over the interpreted data. The main data processing workflow is:

sensor data acquisition → interpreting data → aggregates information.

For example let's consider the situation where the moving detection sensor sends the value 0.05, the time is 23:45, the heart rate is 63 $beats \cdot min^{-1}$. These information are interpreted as it follows: No Movement, Night, LowHeartActivity. The aggregation result about the interpreted data is: the patient is sleeping [39]. The proposed architecture gives to the node level the responsibility to send data, to the sink level the capability to collect data, and to the gateway level the processing data operations, data-stream interpretation and aggregation.

The middleware uses:

- A Dispatcher Sensor Stream Agent (DSSA) that has as main responsibility to receive the signal from the physical sensors and to create a temporary stream warehouse for the SSA.
- Builder Agents that do proposals for the future middleware agent. They write their proposals on the blackboard.
- A Knowledge Discovery Agent that explore another external systems or databases in order to obtain new and useful data. All the data are stored on the blackboard.
- Simple Sensor Agent that are used for the sensor data-stream processing. The processing results are stored on the blackboard.
- Virtual Sensor that indicates some useful time-series data that can be produced by the middleware.
- Logic Sensor Agent that are used to obtain new knowledge over the data from the blackboard: sensor-data, data from another external sources, data gathered from the SSA.

All the interactions between the middleware agents are mediated by the blackboard. Each agent stores its output on to the blackboard, and another agent is activated to process this output if the situation requires this. There are no direct interactions between two SSA or two LSA. There is a data-flow that use a lot of agents, but it is mediated by the blackboard.

Using a lot of software agents that process each sensor data-stream we solve the complexity issue. The WSN uses a lot of sensors. We can have for each sensor category one or more agent. There are situations where a sensor family measure the same fact: in a room there are a lot of sensor temperature used to detect the average temperature. In another situation we have the same kind of sensor but all of these sensors are used to measure different facts. Let's consider in this situation current cost sensor used for each house's appliances. For each sensor data-stream our middleware use **Simple Sensor Agent** (SSA) . The SSA has as responsibility the sensor data-stream processing. The main SSA responsibility is the signal acquisition from the corresponding sensor or sensor family. The SSA is able to understand the received data-stream and to detect all the events in this time-series. The agent's behavior can be changed if special events have been detected in the data-stream. As example let's consider the situation when in the data-stream a strange value was discovered ; in this situation the SSA must to detect if this strange value is a noise or not, and in this way it changes his classical behavior. The main goal of each SSA is to process the received data-stream in order to obtain useful new data. So, the agent's goal does not require a strong proactivity. It is important for the SSA to be more reactive than to be proactive.

There are situations where for each sensor the middleware build a SSA and situations where we build one SSA for a sensor family. When we have a lot of temperature sensors (a sensor family) that measure the same human body, like in the above example, the middleware must build one SSA. When we use sensors for independent phenomena the middleware built different SSA.

The data-stream interpretation or aggregation is due to **Logic Sensors Agent**(LSA). The LSA has the same functionality as SSA, the difference between them is that the LSA is working with the output of SSA or the output of another LSA. In section 3.1 we explained how we can create some abstractions hierarchy, that are translated in one or more LSA hierarchies.

Our architecture also uses **Virtual Sensor Agents** (VSA). Let's consider a first example of a VSA for that encapsulate the time. When we detect that a patient is sleeping we use a detection sensor, a heart-rate sensor and a VSA for the time.

3.4. Dispatcher Sensor Stream Agent (DSSA). An important agent of this architecture is the *Dispatcher Sensor Stream Agent*. This agent connects with the WSN gateway. Our gateway receives all the sensor signals and send them to this agent. The "Dispatcher Sensor Stream Agent" scans all the detected streams, selects only the representative data-stream, and store these data for a very short period of time - 30 seconds as example. The representative sensors are the sensor which will be use by the AAL system, for which the middleware has an acquisition goal. The received data-streams from the representative sensors, are called representative data-streams.

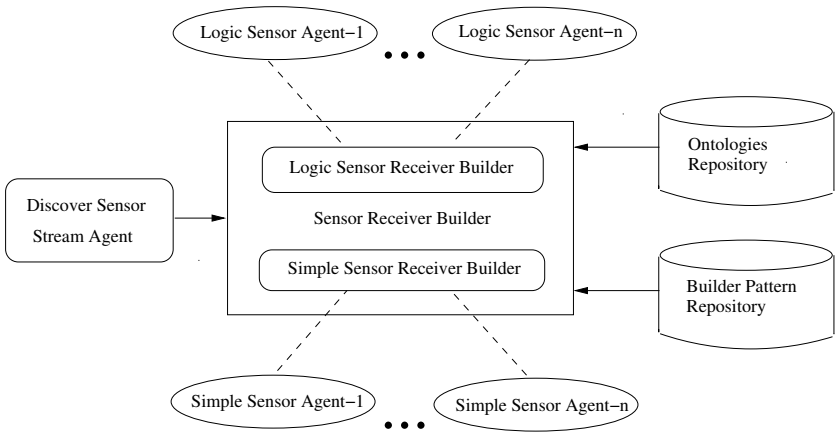


FIG. 3.2. *Middleware Components*

The assisted living systems use a limited sensor data-stream. When in a room we have a lot sensors that send data packets, the middleware receives only the sensor data-stream for which is an acquisition goal. Suppose that in a room we have sensor for ambient temperature, moving detection, luminosity, optical meter for gas consume, electricity etc and also a body sensor network - accelerometer, human body temperature, humidity, etc. The gateway will receive all the signals from all the sensors, and all these data-stream will be stored on the DSSA for a short time period. The AAL system considers as representative sensor, on this example, only the ambient temperature, moving detection, accelerometer, human body temperature and humidity sensor. The DSSA is a temporary buffer that it is used by all the agents. From this buffer the simple agents "extract" their data-stream. The DSSA represents the first blackboard level, because inside it, is a stream warehouse for all the SSA.

There is a ZigBee communication between the sensor nodes. The ZigBee communication is developed using XBee modules for each sensor node. In this way each sensor node has a unique identifier, the Personal Area Network - ID (PAN-ID). By default, for each device the PAN-ID is 3332 and the sensor network administrator must to change this ID below to integrate the sensor nodes in the network and to start the middleware. The WSN main feature is the heterogeneity, because there are sensor nodes for temperature, heart rate, moving detection etc. The WSN heterogeneity is due to sensor nodes physical features and communication protocol. We use a SocketProxyAgent for the communication with sensor nodes and also JESS to initiate this communication. The SocketProxyAgent can be used for maximum 50 parallel connections and also has limitations about the communication protocol. For solving this problem we can also use JESS, to create a socket connection with the sensor network, and to read the received packet. Also for the middleware scalability we can use many SocketProxyAgents. This solution is more complicated than the Jess solution, because it implies the management of all the agents and also we can have the risk of a broken connection and this can change the agent behavior.

Our middleware will process only the representative data-stream: in our case only the data from the accelerometer, human body temperature and humidity, ambient temperature and moving detection sensor. The XML file SensorDataModel describes the data template for the representative sensor. The data template for the representative sensor describes:

- how many fields has a received sensor-data packet;
- each field of this packet: his position on the packet, beginning from left to right, his name, his minimal / maximal value and the unit used to measure this value.

The middleware is able to be configured to select and to operate with some data stream. For this it is necessary to rigorously define the representative sensor data-streams. The XML file SensorDataModel describes the acceptable middleware package data formats. If a data package is not corrupted and this package data-type is not in the SensorDataModel.xml the middleware does not accept this package. There is the possibility to add new package data-type, but this operation is allowed only with human user assistance. For example, the temperature sensor data for a human patient is described as follow:

```
<temperatureSensorModel1>
  <fieldsNumber>3</fieldNumber>
  <field>
     <position>1</position>
     <name>Time</name>
     <minValue>0</minValue>
     <maxValue>89400</maxValue>
     <unit>S</unit>
  </field>
  <field>
     <position>2</position>
     <name>Temperature</name>
     <minValue>0</minValue>
     <maxValue>43</maxValue>
     <unit>C</unit>
  </field>
  <field>
      <position>3</position>
      <name>ID</name>
      <minValue>0</minValue>
      <maxValue>100<maxValue>
      <unit/>
   </field>
</temperatureSensorModel1>
```

For each sensor data-stream a sensor receiver agent is built with the responsibility to manipulate this stream.

3.5. Knowledge Discovery Agent. This agent is used in our architecture only to extract data from different external databases or healthcare systems and to process this data. There are a lot of situations where there is a strong requirement to have a general view about the problem and some information is missing from the local system. In these situations the Knowledge Discovery Agent (KDA) is used to "travel" along different external data-storage repositories and to extract some useful data. On the context of AAL systems, data is distributed over multiple organizations such as hospitals, departmental healthcare-insurance office, etc.

The knowledge discovery process is composed of several phases such as: business understanding, data understanding, data preparation, modeling, evaluation and deployment [9]. For each phase of the discovery process we define an operator that is used to achieve the step goal.

The first task of this agent is to define correctly data about which it is interested. The KDA will analyze data from external sources and will compare this data with its descriptions. When we have a patient that changes his hospital in a period of one week, it is required to obtain some data stream about his cardiac frequency from the last three days. The KDA will inspect a lot of external databases that contains data stream about this patient cardiac frequency, but will select only the data from the last three days. In order to complete his task the KDA will collect data about this patient, and our system, locally, will select the relevant information. The agent query will be more general, to obtain in this way a lot of data about this patient. It is more efficient to collect all data in one agent "travel" than to do more "travels" for each query. So, we can use one or more KDA agents to obtain relevant data about the patient: one agent will travel to collect data and others will process data locally. The file SensorDataModel describes data received from a local sensor, and in the same way another file ExternalDataModel describes data that must be extracted from another external databases or systems. This file describes the query over another external databases and data quality. The extracted data can satisfy all the requirements from the ExternalDataModel files, or partially. A metric will be used to say exactly to how many requirements the extracted data corresponds. Of course, the KDA will use a fuzzy approach to select the relevant data. So the second KDA agent's task is to define the extracted data quality, and another task is to select the relevant data.

The last task KDA agent's is data preprocessing. The agent must prepare data to be used by another middleware agents. This means to translate the extracted data in a form that allows the processing at the middleware level.

This agent can use some transducer agents that allow the communication between the KDA and the external database [16]. The requests must be translated from the ExternalDataModel to ACL (Agent Communication Language) or to native requests over the external data sources.

3.6. Sensor Agent. The Figure 3.4 describes the sensor agent internal architecture. This design integrates three layers: *behavioral level, the core level and the semantic layer*. The core layer is mandatory and the behavioral and semantic layer are optional. The sensor agent behavior consists of a lot of operators that can be used to manipulate the data stream. The sensor operators implement a generic interface. In this way it is possible to define new operators, using these interfaces. For example we can have filter operators, matched-filter operators, persistence operators, transformation operators, aggregation operators etc. The semantic layer loads an ontology in order to "understand" the received sensor data-stream. The understanding process of sensor data is "de facto" due to the interpretation or inference over the received information.

The agent core layer contains all the mechanisms that are used to load the ontology, to define the agent's behavior and to be context-aware. The agents use a pull mechanism to obtain the corresponding sensor data-stream from the blackboard. This mechanism is developed on the core layer. The agent communication with the blackboard is defined on the core layer. This layer contains also the primitives for the initialization and destruction of each agent. In order to ensure the context-aware agent feature is important to store a knowledge base that allows to be reactive at the environment changes. A set of rules are stored on the core layer. The core layer has two main responsibilities: scenario recommendation and its execution. The agent behavior is defined by a scenario, that is an operators workflow. There are a lot of workflows definitions available in a special repository. The workflow is selected using an evaluation formula [24, 34] build based on the ISO / IEC 9126 standard [1].

Another important responsibility is the communication between the agent and the physical sensor or sensors. We don't use the Plug-and-Play mechanism, we read all the received sensor data-stream and we select only the representative data-stream in our middleware. For this reason the agent must communicate with the physical sensor. The agent calculate and allocates the PAN-ID for the sensor / sensors, and it can initiate some special query over the sensor / sensors.

3.6.1. Operators over data-streams. The sensor agent behavior is implemented as a directed graph of operators. Thus, we choose a directed graph as container for the operators because the sensor's behavior must respond to different situations. The problem of sensor data-stream persistence is solved using the persistence operators, which use a reflexive mechanism. To support these operators structure it is also necessary to define some decisional operators.

Our architecture defines three categories of sensor data stream operators: data-stream segment operators, time-set data operators, singular record operators. The proposed operators architecture respects the requirements presented in [35]. The data-stream segment operators operate over a set of received data in a short or long period of time: five minutes, one hour, one day, etc. A temperature-sensor data stream segment can be the following:

```
23:49:40. 37,1
23:49:50. 37,1
23:50:00. 37,1
23:50:10. 37,1
23:50:20. 37,1
23:50:30. 37,1
23:50:40. 37,0
```

In this category we have the following operators:

- data-stream completing operator. This operator replaces the missed value from the data-stream. As example, if in the last data-stream segment the record (23:50:30. 37,1) will be missing, the middleware must complete the data stream. The inserted value will be the average between the record before an the after-record.
- data-stream de-completing operator. This operator deletes all the irrelevant data from the data-stream. In the above example we have the same temperature value from 23:49:40 to 23:50:30. The de-completing operator will delete all the records from this interval or will be reduce the number of records with a specified percent.

- event detecting operator. This operator is used to detect a relevant event from the statistical or semantic point of view. The definition of these operators implies in most cases a pattern recognition engine embedded in the agent.
- inflexion point detection on data-stream operators. This operator has the same behavior like "data-stream de-completing" operator with the difference that it is used to detect if there is a dependency between the time and the sensor value. If there is a dependency, this operator deletes the value that can be calculated and store only the inflexion points (the points where these dependencies are changed).
- statistical operators such as: min, max, average, etc.
- redundant data deleting operators. Redundant data is defined by the user. In some situations redundant data like in the example of "data-stream de-completing operator" or "inflexion-point detection on data-stream operators".
- noise elimination operators. This operator delete some abnormal values. In the above data-stream segment an abnormal value is -37.1 at the moment 23:50:20.
- filtering-operator. This operator identifies some special values in the sensor data-stream, or some pattern in this data-stream. The filter is specified by the user.
- data-stream fusion operators. This operator is used when the segment contains a lot of records and we want to reduce this number. Some other possibilities are to use the de-completing or inflexion point detection operator. This operator will use some statistical method in order to reduce the records number: it deletes a lot of stream-values and introduces new values calculated using some operations like general average over stream-values.
- historical data stream interpretation operators. These operators are used to do some interpretation for the data-stream using data from the "ancient" time.
- data-stream normalization operators.
- data validation. These operators validates the data-stream segment from the semantic point of view.
- stream-to-cube operator, used to produce a cube using the received facts. It is triggered by time or tuple-based events.
- distinct element detection operator over the data-stream segment.
- distinct elements counter operator over the data-stream segment.
- distinct elements frequency operator over the data-stream segment.

Time-set data operators operate over a set of data received at the sensor and gateway level at the same time, that can be manipulated by the same middleware agent. An example for this is the situation when for a patient we consider at the same time more EKG sensors. For this category we can consider the following operators:

- data-interpretation operators.
- aggregation operators.
- fusion operators.
- filtering operators.
- database - storage operators.
- statistical operators such as: min, max, average, etc.

The singular record operators operates over a single record from the data-stream. In this category there are operators used for data-interpretation, database-storage, event-detection and the most important is the alarming operators that detect some special situation and send an alarm message.

Our architecture is open to define new operators for sensor-data stream. In this way, we have define an interface for each category of the above operators. Each new operator must implement an interface. Also we have define inverse operators such as:

- inverse storage operators which defines a data-stream using the stored information;
- inverse interpretation operators that builds the data-stream doing an inverse interpretation.

The sensor behavior is "de facto" an operators workflow. Due to the agent reactivity, the behavior is implemented with a context-sensitive operators workflow.

3.6.2. Ontologies. The interpretation of sensor data-stream is done using some special operators designed for this goal. For a good interpretation or inference, the semantic layer will load an adequate ontology.

The details about the context can be manipulated using different model such as: key-value model, the logic model, the object oriented model, etc. The paper [36] presents a comparative study about the models that can be used for context representation. We chose to use the ontological paradigm to define the context because

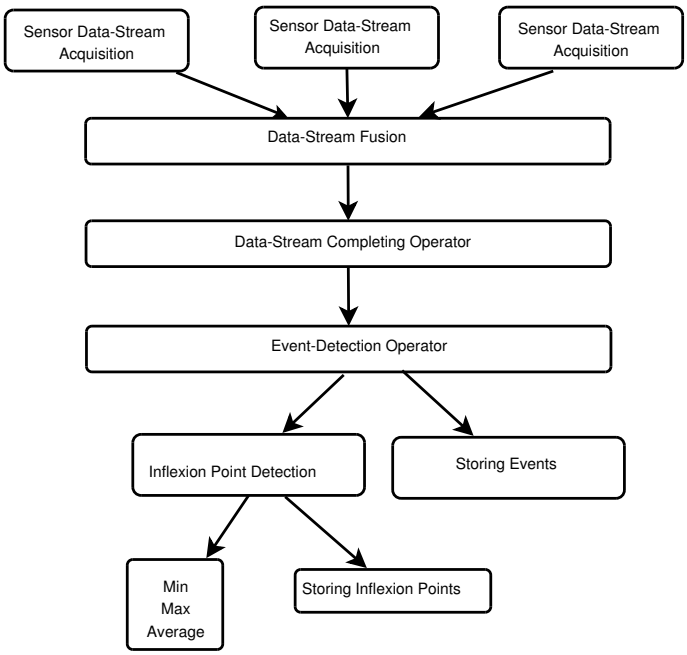


FIG. 3.3. *Sensor Data Stream Operators Workflow*

this model gives support for reasoning, validation and distributed composition. Our system uses different data repositories from different location. On each location we can have different interpretation for the relevant data. So, when we import data from a repository we must to import the ontology and / or the meta-model that describes this data. This means that the ontology model should respond to the distributed composition requirement. Our data must to be validated in a specified context. There is a difference between how we validate the cardiac frequency for a patient with several cardiac attacks or for a normal patient. For each patient we have a different context, and this context is implemented in our middleware using ontologies.

The agent's semantic layer will load the adequate ontology in order to support the context-aware approach. This application will be use in different contextual situation, as example, in a hospital emergency rooms and also in a house for the surveillance of patients. The interpretation and data aggregation must take in consideration the patient's context, his medical background etc. To support these different contextual environments the definition of context based on ontologies is the better choice. When the assistive system "run" for an hospital emergency room will load for all the agents the ontology that describes an emergency room [18]. In another situation when the situation is about elderly people the semantic layer will load the ontology that describes elderly people. The patient's sex also will customize the used ontology. The interpretation of some vital signs depends by patient age or sex. The core of the middleware offers facilities to transfer the contextual information to another sensor agent using the suitable ontology. In this way the middleware can be easy adaptable to different environments.

The context changes over time, so we need to define the past context, using adequate ontologies [22]. Data interpretation depends by the acquisition time. If a person has falling down and at this moment he is in a fix point, the moving detection sensor data is for example 0.05. In a normal situation at the same time of the day the moving detection sensor data has higher value like 0.95. The middleware will load the ontology that describes people that are falling down, an in this way we can obtain a good context overview.

Intelligent ambient systems manipulates large amounts of data, with a high degree of heterogeneity. The ontologies give support to the data heterogeneity and also allow the interoperability. The context information must be commonly understood by all the agent [31]. Another important requirement is related to a more general character of data. Ontologies give support to define concepts hierarchies. The middleware architecture use intensively the Dependency Inverse Principle [27], so the agent are working with more abstract "objects" that are defined using ontologies.

The middleware uses an ontology repository, that is used to describe the context in different situations. The middleware selects the adequate ontology from the above repository, that depends on the following features:

- patient demographic characteristics like: sex, age;
- patient medical background: past diagnostics, medical events in his life.
- patient monitoring context: emergency room in a hospital, patient house, etc.

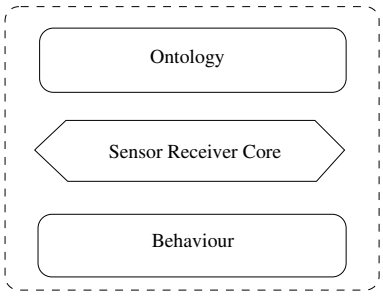


FIG. 3.4. *Inside the Sensor Receiver*

Our ontologies are build around the concept of event. Our middleware has as main goal the event detection on the sensor received data-stream. The main concepts of our ontologies are: patient, event, activity. There are three main event categories: about the subject of our system - the patient- and about the data-stream and about the environment description. The ontologies about the patient contains concepts and predicates that describes exactly the patient state, his actions, his medical background, etc. The data-stream ontology describes the events that can be appear in the received data from the sensor: sensor stops, sensor interruption, sensor changeovers, etc. The environment ontology describes all the events related to environment: the patient environment (EKG, pulse, oxymeter, moving data etc), the room environment (opening a windows / door, cleaning the room, etc), the sensor environment (sensor start event, sensor interruption event, etc).

More exactly our ontology is focused on the event that allow transitions between different states. We chose the event approach to describes the ontologies because our middleware is a real-time software that allows rapid intervention of some specialists (doctors, nurse etc). Our ontologies are more dedicated to the development of predicates that allow the event recognition, than to the concepts definition. Previous ontologies for assisted living system have been proposed in [22, 8, 33]. Comparing with SOPRANO project ontology we focus on the event, not to the state, as central ontology concept. Our approach uses the ontology to a better context understanding, because the agents load the adequate ontology, and we are not working over the ontology like in SOPRANO. On the above project, the ontology is used as a contract between the system components in order to give semantic coherence. Our middleware loads the adequate ontology and shares to all the agents. Our ontology system models the sensor events that allow the acquisition, on the context of an AmI system.

4. Related Work. The gap between the assisted living system and the wireless sensor network represents the necessity to develop sensor middleware. The middleware concept is frequently used in distributed systems. The idea that a sensor network is a distributed system gives reason that a middleware is a solution for this gap. Middleware is an approach to satisfy all the requirements presented in the second section in order to provides services to assisted living system. This section examine the existing middleware for WSNs.

MiLAN [19] uses as an input the following information about: variables of interest for the software application, the QoS for each variable and the QoS level. According to these inputs and the sensor collected data, the middleware provides as output a stream that will be used by the application. To discover and to obtain information about the sensors MiLAN uses a service discovery protocol. MiLAN runs different types of application and need some adaptation to provide services for them.

A related project, IrisNet (Internet - Scale Resource - Intensive Sensor Network Services) middleware use an external storage approach [17, 29]. It uses XML to represent sensor - produced data. These middlewares use software agents to collect and to organize data. There are two kind of agents used in IRISNET: sensing agent and organizing agent. The sensing agents are used to access sensors, and the organizing agent implements a distributed database of services for the collected data. IrisNet uses an adaptive data placement algorithm provided by organizing agent.

AMF (Adaptive Middleware Framework) [40] is a sensor middleware that enables the reduction of the energy consumed in the process of sensor information collection. It uses the "predictability of sensor readings" to improve the efficiency. It provides precision and prediction-based adaptation .

5. Conclusions and future works. This paper introduces a new multi-agent approach for a context-aware sensor data middleware used in healthcare and assisted living systems. Our approach uses the ontology definition of context and introduces some special components to manipulate the sensor data-streams. A original aspect is related to the use of logical sensor agent in order to obtain new knowledge. Comparing with similar approaches [22] our middleware uses ontologies for a contextual understanding. This architecture based on the multi-agent model is scalable, extensible and provides openness to define new operation and adequate workflows for stream manipulation or to define a new way to understand the context.

Our middleware provides support for reasoning at different levels of abstraction over the received data. The data repository store the sensor received data and also high-level information provided by the logic sensor receivers. The main function of the related middleware is to collect the sensor data, but our solution gives support to develop intelligent assistive systems. An ambient intelligent system can be described with three keyword: sensing, thinking, reacting. The described middleware provides services for sensing and support to thinking about the context. Our approach introduces the idea of reasoning over the sensor data, and in this way the assistive system core has as responsibility to take the correct decision and to execute these decisions. This feature is provided by the use of logic sensor receivers and ontologies that describe different situation.

We will take in consideration to describe different context using adequate ontologies. First, this means building ontologies and also giving a mechanism to select the better ontology that describes an environment or a situation. A lot of interest will be focused to the development of a framework that allows data-stream operators, ambient assisted living ontologies, builder pattern and workflow definition. A prototype will be developed on JADE [7].

We will develop our middleware in order to be used not only for assisted living systems. We try to develop a generic middleware for data-stream that can be used in other ambient intelligent systems: home-control systems, traffic control systems, etc.

REFERENCES

[1] ***, *ISO/IEC 9126-1:2001 - Software Engineering. Product Quality*, ISO, Geneva, Switzerland, Jan. 2001.

[2] ———, *ISO/IEEE 11073-30300:2004 - Infrared wireless*, Health informatics – Point-of-care medical device communication – Part 30300: Transport profile, ISO, Geneva, Switzerland, Jan. 2004.

[3] ———, *ISO 12967-1:2009 - Enterprise viewpoint*, Health informatics – Service architecture, ISO, Geneva, Switzerland, Jan. 2009.

[4] ———, *ISO 13606-3:2009 - Part 3: Reference archetypes and term lists*, Health informatics – Electronic health record communication, ISO, Geneva, Switzerland, Jan. 2009.

[5] J. C. Augusto, H. Nakashima, and H. Aghajan, *Ambient intelligence and smart environments: A state of the art*, in Handbook of Ambient Intelligence and Smart Environments, H. Nakashima, H. Aghajan, and J. C. Augusto, eds., Springer, New York, 2010, pp. 3–31.

[6] J. E. Bardram, *Applications of context-aware computing in hospital work: examples and design principles*, in SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, New York, NY, USA, 2004, ACM, pp. 1574–1579.

[7] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, Wiley, 2007.

[8] Y. Cao, L. Tao, and G. Xu, *An event-driven context model in elderly health monitoring*, Ubiquitous, Autonomic and Trusted Computing, Symposia and Workshops on, 0 (2009), pp. 120–124.

[9] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, *Crisp-dm 1.0 step-by-step data mining guide*, tech. rep., The CRISP-DM consortium, August 2000.

[10] D. D. Corkill, *Collaborating Software: Blackboard and Multi-Agent Systems & the Future*, in Proceedings of the International Lisp Conference, New York, New York, October 2003.

[11] R. Costa, P. Novais, L. Lima, D. Carneiro, D. Samico, J. Oliveira, J. Machado, and J. Neves, *Virtualecare: Intelligent assisted living*, in Electronic Healthcare, vol. 0001 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, 2009, pp. 138–144.

[12] K. Fishkin and M. Wang, *A flexible, low-overhead ubiquitous system for medication monitoring*, Tech. Rep. IRS-TR-03-011, In: Intel Research, 2003.

[13] C. Floerkemeier and F. Siegemund, *Improving the effectiveness of medical treatment with pervasive computing technologies*, 2003.

[14] D. Gay, P. Levis, D. Culler, and E. Brewer, *nesc 1.1 language reference manual*, 2003.

[15] D. Gay, M. Welsh, P. Levis, E. Brewer, R. von Behren, and D. Culler, *The nesc language: A holistic approach to networked embedded systems*, in Proceedings of Programming Language Design and Implementation (PLDI), year = 2003, pages = 1–11.

[16] M. R. Genesereth and S. P. Ketchpel, *Software agents*, Commun. ACM, 37 (1994), pp. 48–53; 147.

[17] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, *Irisnet: An architecture for a world-wide sensor web*, IEEE Pervasive Computing, 2 (2003).

[18] T. Gu, H. K. Pung, and D. Q. Zhang, *A service oriented middleware for building context-aware services*, Journal of Network and Computer Applications, 28 (2005), pp. 1 – 18.

[19] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, *Middleware to support sensor network applications*, Network, IEEE, 18 (2004), pp. 6–14.
[20] E. Jovanov, A. Milenkovic, C. Otto, and P. de Groen, *A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation*, Journal of NeuroEngineering and Rehabilitation, 2 (2005), p. 6.
[21] J. Kjeldskov and M. B. Skov, *Supporting work activities in healthcare by mobile electronic patient records*, in Computer-Human Interaction, Springler, 2005, pp. 191–200.
[22] M. Klein, A. Schmidt, and R. Lauer, *Ontology-centred design of an ambient middleware for assisted living: The case of soprano*, in Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIMCU), 30th Annual German Conference on Artificial Intelligence (KI 2007), Osnabrck, September 10, 2007, T. Kirste, B. Knig-Ries, and R. Salomon, eds., 2007.
[23] C. Kunze, U. Grossmann, W. Stork, and K. Muller-Glasser, *Application of ubiquitous computing in personal health monitoring systems*, Biomed Tech(Berlin), 47 (2002).
[24] Y. Lei and M. P. Singh, *A comparison of workflow metamodels*, 1997.
[25] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White, *Revisiting the jdl data fusion model ii*, in In P. Svensson and J. Schubert (Eds.), Proceedings of the Seventh International Conference on Information Fusion (FUSION 2004, 2004, pp. 1218–1230.
[26] S. W. Loke, *Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective*, Knowl. Eng. Rev., 19 (2005), pp. 213–233.
[27] R. C. Martin, *The dependency inversion principle*, C++ Report, 8 (1996).
[28] S. Mitchell, M. D. Spiteri, J. Bates, and G. Coulouris, *Context-aware multimedia computing in the intelligent hospital*, in 9th ACM SIGOPS European Workshop, ACM Press, 2000, pp. 13–18.
[29] S. Nath, P. B. Gibbons, and S. Seshan, *Adaptive data placement for wide-area sensing services*, in FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, 2005, pp. 4–4.
[30] P. Nii, *The blackboard model of problem solving*, AI Mag., 7 (1986), pp. 38–53.
[31] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. D. Bosschere, *Towards an extensible context ontology for ambient intelligence.*, in EUSAI, P. Markopoulos, B. Eggen, E. H. L. Aarts, and J. L. Crowley, eds., vol. 3295 of Lecture Notes in Computer Science, Springer, 2004, pp. 148–159.
[32] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, *Data-centric storage in sensornets*, SIGCOMM Comput. Commun. Rev., 33 (2003), pp. 137–142.
[33] A. Sixsmith, S. Meuller, F. Lull, M. Klein, I. Bierhoff, S. Delaney, and R. Savage, *Soprano — an ambient assisted living system for supporting older people at home*, in ICOST '09: Proceedings of the 7th International Conference on Smart Homes and Health Telematics, Berlin, Heidelberg, 2009, Springer-Verlag, pp. 233–236.
[34] K. Stoilova and T. Stoilov, *Comparison of workflow software products*, in International Conference on Computer Systems and Technologies - CompSysTech, 2006.
[35] M. Stonebraker, U. Çetintemel, and S. Zdonik, *The 8 requirements of real-time stream processing*, SIGMOD Rec., 34 (2005), pp. 42–47.
[36] T. Strang and C. Linnhoff-Popien, *A context modeling survey*, in Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004.
[37] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, *Collaborative storage management in sensor networks*, International Journal of Ad Hoc and Ubiquitos Computing, 1 (2005), pp. 47–58.
[38] M. Vallee, F. Ramparany, and L. Vercouter, *A multi-agent system for dynamic service composition in ambient intelligence environments*, in The 3rd International Conference on Pervasive Computing (PERVASIVE), 2005.
[39] M. R. Waldeck and M. I. Lambert, *Heart rate during sleep : Implications for monitoring training status*, Journal of Sport Science and Medicine, 2 (2003), pp. 133–138.
[40] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian, *Adaptive middleware for distributed sensor environments*, IEEE Distributed Systems Online, 4 (2003).

Edited by: Adina Magda Florea, Costin Bădică
Received: February 29, 210
Accepted: March 31, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 33–42. http://www.scpe.org
ISSN 1895-1767

CONTEXT-AWARE EMERGENT BEHAVIOUR IN A MAS FOR INFORMATION EXCHANGE*

ANDREI OLARU, CRISTIAN GRATIE, AND ADINA MAGDA FLOREA†

Abstract. The plethora of interconnected devices that surrounds modern people has yet to work together as a whole. An intelligent environment must sense and react to the actions of people, but to that end a large quantity of information must be exchanged throughout the system. Under realistic conditions, it is impossible to control and coordinate the exchange of information in a centralized way. Solving this problem involves key concepts like self-organization, emergent behaviour and context-awareness. Continuing previous work on self-organizing cognitive multi-agent systems for the exchange and management of information, this paper introduces two aspects of context-awareness – pressure and interest – that make the system's emergent behaviour more context-sensitive and, therefore, more adaptive to a changing environment.

Key words: multi-agent system, self-organization, cognitive agent, context-awareness

1. Introduction. People in the modern world are surrounded by a huge number of electronic devices that have different capabilities, different sizes and different performance, all of them interconnected by wireless or wired networks, but not quite working together and cooperating towards the resolution of tasks.

Ambient Intelligence – or AmI – is the field that deals with electronic environments that are sensitive and responsive to the presence and actions of people [1]. It refers to a ubiquitous electronic environment that supports people in their daily tasks, in a proactive, but "invisible" and non-intrusive manner [14, 20]. It implies, on the one hand, embedded, natural and personalized interfaces and, on the other hand, an underlying ubiquitous network that links the devices together into one single system that acts as a whole.

An Ambient Intelligence environment may be regarded as comprising several layers [17]: the hardware – comprised of sensors, actuators, mobile devices, computers, intelligent appliances and material; the network – an ubiquitous infrastructure that provides connectivity by means of various wired and wireless, possibly ad-hoc, media and protocols; the interoperability layer – that will allow uniform communication among heterogeneous devices using different hardware and different types of connections to the network and will enable easy information transfer and code migration between devices; the application layer – that will be more oriented towards knowledge and semantic information and will offer context-aware services to the layer above; and the intelligent user interface – that will offer to the user a natural, intuitive, non-intrusive and multi-modal means of interaction with the system.

From our point of view, one essential issue is what happens *between* the upper layer of the human-machine interface and the layers of interoperability, network, and hardware – i. e. at the application layer. The exact manner in which information is transferred between devices is far from having a trivial solution. First, all aspects of one's life taken into account, the quantity of information that transits the system is very large. Second, most of the devices that use the information (mobile phones, mp3 players and even simple sensors and actuators) have reduced storage and processing capacity. Considering the required flexibility of the system and the number of devices involved, it is obvious that centralized control is not viable, therefore the system must self-organize. Moreover, considering the very context-aware nature of Ambient Intelligence, the vast majority of the information that exists in the system is bound to a certain type of context: temporal, social, but most times spatial.

An appropriate paradigm for the implementation of AmI, addressed especially to the application layer, is the agent-oriented approach. Agents have features like reactivity, proactivity, autonomy and reasoning. Cognitive agents are by definition fit to working with information in a more semantic manner. Adding self-organization [18] to a multi-agent system allows for decentralization and resiliency, as well as for a greater degree of flexibility and adaptivity.

We have addressed the issue of a self-organizing system for the management of information in our previous work [11, 12]. However, the notion of context representation and awareness had not yet been integrated.

This paper takes a step forward towards the implementation of context-awareness in a system for the distributed management of information, using agents that are cognitive but have limited storage capacity. In this case, a simple and generic enough representation of context is necessary, one that is easy to process but also

*This research was supported by Grant CNCSIS ID_1315, 2009-2011, and Grant POSDRU 5159.
†Department of Computer Science, University Politehnica of Bucharest, 313 Splaiul Independentei, Bucharest, 060042 Romania (cs@andreiolaru.ro, cgratie@yahoo.com, adina@cs.pub.ro)

useful enough for the functioning of the system. The paper proposes two elements of context representation, namely *pressure* and *interest*. The two elements relate to two aspects of context awareness: the relevance that the source associates to a piece of information (this is the pressure) and the relevance that the receiver associates with the piece of information (this is the interest).

A system for the emergent management of information, using the two aspects of context awareness, has been designed and implemented. The system has a generic design, but mainly there is an agent in the system that is assigned to each user of the system, and that manages the information that is available to, or made available by, the assigned user. The system has been successfully tested on several scenarios, that, though simple, involve a great number of agents and multiple pieces of information.

Section 2 is dedicated to related work in the fields of context-awareness, emergence and self-organization. Section 3 presents the requirements and the general description of the implemented system. Section 4 introduces some aspects of context awareness in the described system and two elements for the representation of context are proposed: pressure and interest. Section 5 describes the design of the system and the results of the experiments are discussed in section 6. The last section draws some conclusions.

2. Related Work. In the domain of Ambient Intelligence there are many directions of research, ranging from intelligent user interfaces to sensor networks to mobile agents offering services. Architectures for the management of the layer offering context-aware services generally address particular scenarios: conference rooms [8], intelligent buildings [15] or other limited smart spaces [9]. Even generical architectures are oriented towards scenarios implying a small number of individuals, which allows the use of man centralized components [16, 3, 17]. The architecture presented in this paper aims to provide a platform that is scalable and that relies on no centralized components.

Self-organization and emergent behaviour in multi-agent systems are usually inspired from biological systems and have been studied mostly by using reactive agents [18, 10]. Most of the obtained emergents consist of the organization of agents in a spatial or space-related structure that groups agents with certain states: placement of agents in circular shapes or shapes of more complex form [21], coverage of certain areas [2], space related dynamical behaviour [13]. The limitation of the capabilities of these systems comes from the use of reactive agents, that, as opposed to cognitive agents, are very simple and therefore cannot lead to very complex functions at the level of the system. Self-organization has also been studied for systems based on agents having a more elaborate model, but such approaches are less frequent [6, 7] and less adaptive. Moreover, they have not so far been used for the management of information or for the exchange of agent's beliefs.

Research in context-awareness offers complex solutions even in the field of mobile devices [4], but the representation of context lacks generality and is difficult to implement using really small amounts of memory. There are many manners in which complex information may be tackled, for example by using complex ontologies [19]. However, that means that either the device must be able to work with ontologies, or that there has to be a centralized service that offers the possibility of responding to context queries [9], which may not be scalable. This paper aims to offer a solution that requires very low storage and processing capacity. Moreover, the agent's behaviour is tuned in accordance to the context, enabling it to act with increased promptness whenever the situation requires it.

Our previous work [12] described a system for the exchange and management of information. Unlike most studies of self-organizing systems, our approach used cognitive agents instead of reactive ones, because of their increased flexibility and possibilities. The interaction between agents lead to emergent properties related to the uniform distribution of data. However, the context was not taken into account. In an AmI environment, context is essential as most information is related to, and may only have sense in, a certain context, be it spatial, temporal, social or computational [4]. Most times the spatial aspect is considered [9, 8], as it is directly related to directly available resources and services.

3. System Requirements. The purpose of the system we designed is to manage information in such a way that its users will have interesting or needed information available without the need to know where this information comes from or how it was made available to them. From the user's perspective, two operations are possible: either insert information into the system or request certain information he/she might be interested in. Also, the system might provide the user with information that is potentially interesting to him/her. We will discuss each of these key concepts in the following paragraphs.

At this point, there is no specific, real-life application of the system, as we have tried to keep it as generic as possible. However, many applications may be imagined, as most Ambient Intelligence scenarios imply heavy

exchange of information between users or between users and other entities (e.g. smart spaces, AmI services), trying to deliver useful information to the users that need it [5].

The system is implemented as a cognitive multi-agent system. Agents hold information in the form of beliefs, and have goals and plans on what to do with that information. The system has been designed as to manifest self-organization, therefore the agents have been given a behaviour that is local, but that reflects the global, desired, objectives. To that end, agents have been given some human-inspired features, like curiosity, states of being under pressure, and the ability to forget. What each agent should do is to (1) realise what information may be of interest to the associated user and get it and (2) provide other agents with information that it believes that may be interesting to them.

The different pieces of information that exist in the system will be referred to generically as *data*. Data is characterized by its content, by the size of its content and by context-specific information. The actual content of data should not be mistaken for beliefs about that piece of data. For example, a piece of data may be the recipe of how to prepare a certain kind of food. The fact that an agent A knows that B has the recipe (and knows how to prepare the food) does not mean that A can prepare the food, but this knowledge can help A get hold of the useful information.

4. Aspects of Context Awareness. In previous experiments with the information management system that we have developed [12], two issues were encountered. First, the behaviour of the agents was only regulated by the quantity of plans and messages that an agent had and was not related to the informational content of the agent's knowledge or to the received data. The agent acted the same way in the case of receiving data that needed a quick spreading and in the case of data for which there is no such requirement. Second, the data was distributed by the same rules, independent of its content or relatedness with any domain, and independent of the interest of the agents. The two issues are both related to the lack of context awareness.

The solution for the first issue is the integration of a measure of urgency associated with the data, representing how quickly the data should be handled and how important it is for it to be passed on. This measure was called *pressure*. Higher pressure means more urgency and the data should spread more and faster throughout the system. In our example, high pressure may be associated with very important results, or with announcements of great urgency. Lower pressure is associated with information that is only meant to spread slower, in a more limited area.

Pressure is also associated with requests for data. The higher the pressure, the more urgent the request for data is and the system should react more quickly and provide the necessary data to the requesting user. Lower pressure means that the user does not expect the data to be available immediately and the system is allowed to react less promptly.

The global pressure of all facts in the knowledge base represents the pressure on the agent. As discussed later on, the agent will act differently when it is under pressure and when it is relaxed.

The second issue relates to the definition of *interest*, measured according to some *domains* of interest. In this paper, three generic domains of interest are considered, named A, B and C.

Each piece of data is associated with a measure of interest, comprising the amount of relatedness between the data and each domain of interest. For example, data that is in domain A but also has a degree of relationship of 0.2 with domain C will have a measure of interest represented as $\langle A : 1.0,\ C : 0.2\rangle$. We will call this *data-interest*.

Each agent has a measure of its interest towards the three domains, calculated and adjusted according to the data that the agent has and the data that was produced and/or requested by its associated user. We will call this *agent-interest*. Also, each belief that an agent has about some data or some other agent is associated with an unidimensional measure of interest that represents how interesting that fact is to the agent. We will call this *fact-interest*. The representation of facts is discussed later on.

Considering the multi-dimensional (in our generic case, three-dimensional) measures of interest as vectors, the interestingness of a particular belief is calculated as a function of the norm of the difference between the vectors of interest associated with the data and with the agent. The fact-interest is normalized to be in the interval [0, 1].

In more detail, the fact-interest of a belief F about some data D is calculated in function of the data-interest of data D, that is, say, $\langle di_1 \dots di_n\rangle$ and the agent-interest of agent A having that fact in its knowledge base, say $\langle ai_1 \dots ai_n\rangle$. The fact-interest – let it be noted fi – represents the similarity between the two, normalized to be between 0 and 1. We desired that the similarity be greater when the two vectors are closer, but also we wanted

it to not evolve linear with the difference between the vectors, but make large differences between components count more, i. e. reduce the similarity. Therefore, the root mean square of the individual component differences was used, i. e. $sqrt(\sum(ai_j - di_j)^2/n)$, where n is the number of domains of interest. This value was 0 for full similarity and 1 for no similarity, therefore the similarity is calculated as the value above subtracted from 1.

For example, if an agent interested in domains A (more) and C (less) – agent-interest is $\langle A : 0.9,\ C : 0.3\rangle$ – learns (from another agent) a fact about data D – data-interest $\langle A : 0.1,\ B : 0.2,\ C : 0.9\rangle$ – the resulting fact-interest associated to the fact in the agent's knowledge base will be the equal to $1 - ||\langle 0.9, 0.0, 0.3\rangle - \langle 0.1, 0.2, 0.9\rangle||/\sqrt{3} = 0.41$ (the calculus, although not using the exact formula above, leads to the same result). The resulting interest will be moderate, because the agent is only moderately interested in domain C, that the data is mostly related to.

Context is most times related to space. In most cases, the farther the user from the space providing an information, the less the information is relevant to the user. Therefore in the model presented in this paper, pressure and interest are related to distance. Pressure decreases as information is shared farther, and interest will be lower for facts that refer to agents at a longer distance.

5. System Design. The system is conceived as a two-dimensional space in which cognitive agents are placed and each agent has a number of acquaintances with which it can communicate directly. Experiments were carried out using agents placed in a rectangular grid, each agent communicating directly with its 8 neighbours. In the future, the system will be improved to support more flexible and dynamical topologies. Right now, we have focused on how to design agents' behaviour so that the desired global result was obtained.

The multi-agent system has been implemented as a Java application, in which agents run in a pseudo-parallel manner: at each global time step, agent perform one internal step, each at a time, in order.

5.1. Agent Beliefs. Agents are cognitive and implement the Belief - Desire - Intention model. The beliefs of an agent are held in its knowledge base and are represented using three structures: *Data*, *Goal* and *Fact*. A *Data* structure represents information about one piece of data: the content, the size and the interest relative to the domains. The content of data must not be mistaken for the beliefs about the data, as we have said earlier. A *Goal* structure contains information about an objective, or goal, of an agent. One example of an agent's goal (and the most used goal in the system) is "need to get data D", represented as $\langle Get, D\rangle$, where D is a *Data* structure. However, what agents hold in their knowledge bases and what they send to each other are *Fact* structures. A *Fact* is a tuple that can have one of the following forms:

$$\langle Agent, Data, pressure, interest\rangle$$
$$\langle Agent, Goal, pressure, interest\rangle$$
$$\langle Agent, Fact, pressure, interest\rangle$$

These three types of facts represent associations between the specified elements. The first type of fact means that "*Agent* has *Data*"; the second – "*Agent* has *Goal* objective"; the third type means that "*Agent* knows *Fact*". All three types contain an indication of *pressure* on the fact. The pressure shows how important this fact is for the *Agent*. In the first and third cases, it shows how quickly the fact should be processed and / or sent to the neighbours. In the second case, it shows how important it is for the agent to fulfill the goal. The interest is the *fact-interest* associated with the fact, as specified in section 4.

It is very important to note the use of the third type of fact, i. e. agents may know facts about what other agents know. This kind of knowledge may also be passed to other agents, if it is of interest. In order for the agent not to be overwhelmed by irrelevant facts about distant agents, the knowledge base is updated by removing the facts that the agent is least interested in and that have lower pressure. This is also useful if agents reside on devices with lower capabilities.

Another remark about the third type of fact must be made. As this type is recursive, it is obvious that the last nested fact must be of one of the other two types. The knowledge base of an agent must carefully check the contained facts so that no circularity appears.

5.2. Agent Goals. The choice of individual agent goals is particularly important for our approach, as the system is meant to exhibit global, emergent properties as a result of local goals and local interaction between agents. As stated before, the fact that the agents only have local goals and a local image of their environment means that they need less computing power and may be deployed on smaller devices.

The global goal of the system is to integrate new information coming from the users and to make it available to other users that need it or might find it interesting. Therefore, at the local level, agents should interact with their neighbours in order to exchange data and collaborate. With this goal in mind, the agents were designed

with human-inspired features that would help them share relevant information as well as prevent them from being overwhelmed by facts or by the data itself.

The agents have been given the following goals (no particular order was used; in parenthesis – the name of the type in the internal representation):

- share data (according to context) (*INFORM*);
- keep some storage available in case data is injected from the exterior (by the human user) (*FREE*);
- get interesting data from other agents (*GET*);
- fulfill external (human user) requests (*EXT*);
- help other agents fulfill their goals (*SHARED*).

Goals are represented as pairs of the form $\langle Goal\_type, Object \rangle$, where *Object* may be the description of a piece of data (its identifier), in the case of *GET*, or a fact about a piece of data, in the case of *INFORM*.

Goals are chosen according to their importance, which is represented by means of *pressure*. In most cases, the pressure is taken from the fact associated with the goal: for external requests and for helping other agents, it is a fact of the form $\langle Agent, Goal \rangle$ that is received from a neighbour or from the exterior; for getting interesting data, it is (possibly nested into other facts) a fact of the form $\langle Agent, Data \rangle$. The goal of keeping storage available has a pressure which varies exponentially with the amount of used capacity over 75%, as it is essential to have some free storage available at any time so that the user can insert new data.

5.3. Agent Plans. The plans contain the actions that an agent must perform in order to fulfill its goals. In the system that we have designed, an agent may have several ongoing plans at the same time, and may also have a set of *waiting* plans, i. e. their completion depends on an external event, like knowledge or data expected to come from another agent, as a result of a request.

The plans that an agent may build are formed of several basic actions, like:

- send data to a neighbour, following a request;
- request data from a neighbour, if the agent knows that the neighbour has that data;
- inform a neighbour of a fact that the agent believes relevant to the neighbour. This will be done only if the agent believes that the neighbour does not already know that fact. The fact may express that an agent has certain data, or that an agent has a certain goal;
- discard some data, according to its relevance to the agent's domain of interest and according to the recent frequency of requests (external or not) that have been made for that data.

5.4. Agent Behaviour. The behaviour of the agents is fairly normal for a BDI architecture, but it has some particularities that make them suitable for the required application.

The stages that an agent goes through during a step of the system's evolution are:

- **Receive data from and send data to** neighbour agents. Data is transmitted only as a response to previous requests. These operations are simple and need no reasoning, and that is why they are handled separately and before any other decision is made.
- **Revise beliefs**. This is done based on information received from the other agents. Duplicate or circular facts are found and removed. The interest and pressure of new facts are computed. An important thing to note here is that the pressure of received facts is decreased, so that the pressure of a fact diminishes with each step it takes farther from its source.
- **Check ongoing or waiting plans** for completion (was the goal achieved?) and test whether they have become impossible. In the case of completion the plan is discarded and the pressure of the corresponding facts is cancelled. The interest towards the facts, on the other hand, remains constant.
- **Make plans**. Take the goal with the highest current importance (pressure). If there is no plan for it, make a plan composed of the actions needed to be taken and put it in the list of ongoing plans.
- **Execute plans**. Take the plan associated with the most important goal and execute its next action. Each executed action reduces the pressure of the associated goal with some amount. If there are no more actions to be performed, move the plan to the list of waiting plans. It will be checked for completion in the next cycle.
- **Fade memory** of all facts in the knowledge base. Pressure of all facts is faded. Facts in which the agent has no interest or that have low pressure may be discarded ("forgotten"). This step is necessary in order to avoid overwhelming the agent with useless facts and too old concerns.
- **Revise pressure and interest**. The pressure on the agent is recalculated according to the facts in its knowledge base, as a weighted mean of the pressures of the individual facts, giving more importance

to high-pressure facts. The interest is updated as well, according to the pieces of data that the agent holds and according to its recent activity (the resulting interest is also calculated as a mean).

As it is also the case for natural systems, the behaviour of the agent is greatly influenced by the pressure that presses upon it. Besides the instantaneous pressure, the agent's state is also described by a lower and a higher limit for the pressure. If the pressure is between the two limits, activity is considered normal. If pressure is lower than the lower limit, the agent is considered "relaxed". If the pressure is above the higher limit, the agent is considered "stressed".

The more "stressed" the agent is, the more it will focus on completing its most pressing plans. New knowledge and data acquisition will be reduced to a minimum.

The lower and higher pressure limits are not fixed, and they are adjusted in time. If the pressure on the agent continues to be high for a long time, the two limits will rise and the agent will start to consider its condition as a "normal" one.

6. Experiment. There were many experiments carried out in the study of context-aware emergent behaviour. Experiments were performed using a system having 400 agents. The scenario we will present is generic, but fits a great number of real-life situations. It is focused on observing how the facts about data (inserted in single instances, into individual agents, therefore not in multiple copies) spread through the multi-agent system. In this spreading, two things were observed: the speed in which facts spread, the coverage they reach and the particular area (or areas) into which they spread.

The system was implemented as a Java application. The execution of the agents is synchronous – at any moment of time all agents are executing the n^{th} step of their evolution. Besides the objectives stated above, one other objective was to make the system run as fast as possible, making the behaviour of individual agents more efficient. This also lead to lower simulation times and the possibility to perform more experiments. A typical experiment lasted 200-250 steps in the multi-agent system's time and about 15 seconds in real time on a machine with an Intel Core 2 Duo 3.33GHz CPU, with 2GB of RAM memory.

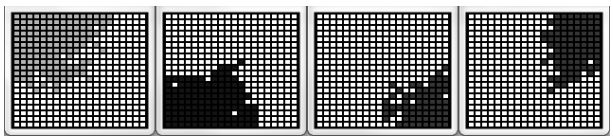
6.1. Monitoring Tools. The behaviour of complex systems is extremely difficult to observe in simple graphical representations and the study of the system's behaviour means, most times, the minute observation of the activity log of each agent. This is why a simple scenario was used, one that makes the evolution of agent states clearer.

This is also the reason why some visualization tools have been developed. The graphs that were used are of three types:

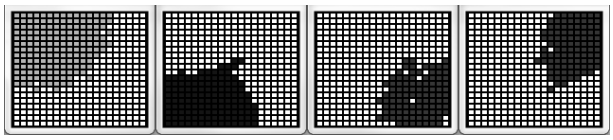
- data-fact distribution – associated with a certain piece of data; the graphic is a two-dimensional grid representation of the system where the color of a cell shows the existence of facts regarding the data in the knowledge base of the agent in that cell (e.g. any graphic from Figure 6.1 (a) and (b)).
- pressure distribution 3D – a three-dimensional surface showing the amount of pressure (z-axis) on each agent in the system (e.g. Figure 6.5 (a) - (e)).
- pressure distribution 2D – a two-dimensional grid representation showing the amount of pressure, as intensity of colour, on each agent in the system (e.g. Figure 6.4 (a) - (e)).
- interest distribution – a two-dimensional grid representation of the system where the colour of a cell shows the amount of interest that an agent has for a certain domain (e.g. Figure 6.2 (a)). The colour of each cell is generated by direct conversion of the agent-interest (which has three components in the interval [0,1]) into an RGB code.
- fact number graph – associated with a certain piece of data; a graph that shows the evolution of the total number of facts regarding the data (calculated as a sum over all the agents) in function of the time step (e.g. Figure 6.1 (c) and Figure 6.3 (b)). The evolution is shown for the whole interval between the start of the system and the current time. The graphs are always scaled to fit the window, but the first number that accompanies the graph show the maximum value of the graph (the minimum is always 0). If there is also a second number, that shows the current value, in case it is lower than the maximum.

6.2. Scenario. As said, the scenario was designed so that it would emphasize the context-ware behaviour, i. e. the influence of pressure and interest on the spreading of facts through the system.

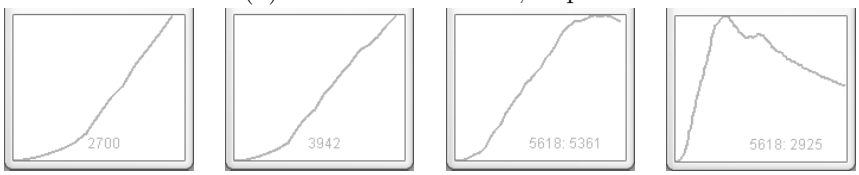
At system start (time 0), the agents in the system have no defined interest and are under no pressure. In the first phase, 4 pieces of data are inserted into the agents in each corner of the grid. They are named $D1 - D4$. The first three relate each to 1 domain of interest ($A - C$) and $D4$ is strongly and equally related to both domains B and C. They all have moderate pressure, except for $D1$ which has higher pressure than the



(a) – facts for $D1 - D4$, step 15



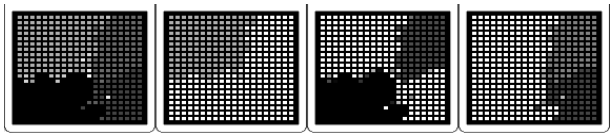
(b) – facts for $D1 - D4$, step 20



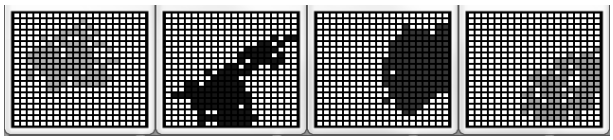
step 15 step 20 step 35 step 100

(c) – fact number for $D1$

FIG. 6.1. *Distributions for facts regarding $D1$ to $D4$, at steps 15 (a) and 20 (b); number of facts regarding $D1$ at several moments of time. Graph scaling is explained in Section 6.1.*



(a) – step 20, interest



(b) – step 35, facts for $D5 - D8$

FIG. 6.2. *(a) Agent interest for domains A, B and C. In the first image interest in all domain is represented. Next three images represent the interest for one domain of interest each, respectively.*

others. The system is given time to stabilize (20 steps) and then 4 more pieces of data are inserted into the system, into agents in the center of the grid. These are named $D5 - D8$ and are related to domains A, B, C and both A and C, respectively. After some time (after 20 more steps, at time step 40), a new piece of data – $D9$ – that is not related to any domain in particular, but has a very high pressure, is inserted into one side of the grid.

The expected results are the following:

- facts with higher pressure spread more, and faster
- facts spread more in areas where agents are interested in them

6.3. Results and Discussion. Relevant results that have been obtained after experimentation are displayed graphically in the Figure of this section. We will comment on some interesting phenomena that can be observed, with regard to the two measures of context-awareness that were introduced.

Pressure indicates the importance and urgency of a piece of data or fact, and, in the case of agents, the need for quick reaction. This makes facts with higher fact-pressure spread quicker and over larger areas. This can be observed in several cases in the figures.

First, in Figure 6.1 (a) and (b) one can notice that facts for $D1$ spread more, and faster, as they have higher pressure. For comparison, at step 20 there are almost 4000 facts about $D1$ in the system, which is 36% of all the

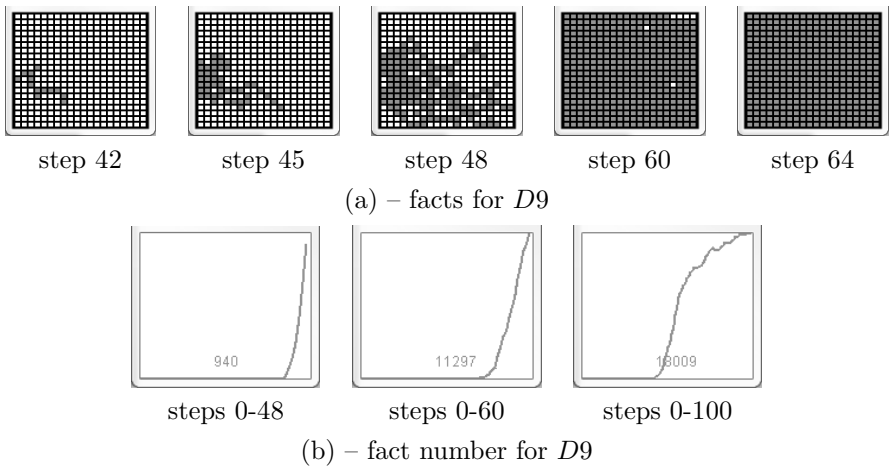


FIG. 6.3. *(a) Evolution of the distribution of facts regarding D9; (b) graphs of the number of facts regarding D9. Graph scaling is explained in Section 6.1*

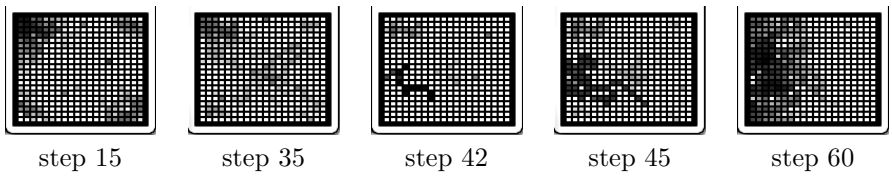


FIG. 6.4. *Pressure on agents in the system, represented as shades of gray, at different moments of time.*

facts. As the agents have no initial interest the interest is formed after the spreading of the 4 initial facts. Since $D1$ has the highest pressure, there will be more agents with interest for domain A, as seen in Figure 6.2 (a).

Second, when $D9$ is inserted into the system, having maximum pressure, it spreads visibly fast. It takes only 14 time steps for the information to be known by every single agent in the system (see Figure 6.3 (a)). This shows that if urgent information is to be inserted, it will indeed spread very fast. This very quick reaction is also due to the agents adjusting behaviour under high pressure. The number of facts regarding $D9$ rises exponentially in the beginning (see Figure 6.3 (b)) and the facts reach a long distance from source very quickly, as compared to the evolution of other facts inserted into the system.

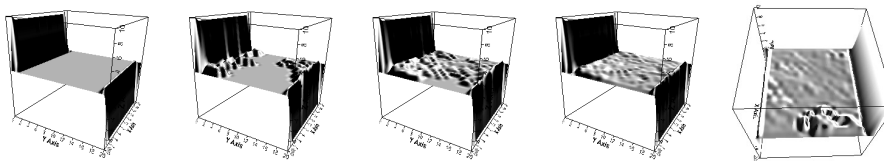
Also regarding pressure, Figure 6.4 shows the pressure on agents at different moments of time. Notice, for example, that in the beginning there is higher pressure in the corners, where new data has been inserted. At steps 42 and 45 the pressed agents are the one holding facts about $D9$. And, due to the effect of $D9$, the whole left side of the system is under high pressure, as it is closer to the origin of $D9$.

A more intuitive perspective upon pressure is presented in Figure 6.5, where one can observe how pressure rises in not necessarily contiguous spots and then becomes more uniform as facts spread in the system. When new data is inserted, pressure rises again (Figure 6.5 (e)). Observe the "wave" of pressure. Behind the "wave", pressure is high, but more uniform, and decreasing. On the edge of the "wave", pressure is non uniform, as there are still some agents that haven't received the facts yet.

As pressure of the facts fades, they may be forgotten, especially if interest in them is lower. Therefore, after the initial surge, the number of facts starts decreasing towards stabilization (see Figure 6.1 (c)). In our experiment, the number of facts for each data stabilized at a mean of 3 to 5 facts per agent.

Interest controls the direction of information exchange and the area of spread, not in terms of size, but in terms of the previous experience of agents. That is, agents that have exchanged more facts related to a certain domain of interest will become more interested in that domain. Positive feedback will lead to the formation of groups of agents interested in a certain domain.

In support of this statement, one can observe Figure 6.2. The most eloquent distribution is that of facts regarding $D6$ – which is related to domain B, that follows the interest of agents in domain B, starting from the center and moving towards the two corners of the grid. On the other hand, as $D9$ is not related to any particular domain of interest, it will spread uniformly through the system.



(a) (b) (c) (d) (e)

FIG. 6.5. *Pressure distributions: (a) initial state (step 0); (b) immediately after inserting new pieces of data into the system; (c), (d) stabilization of pressure across the whole system; (e) a "wave" of pressure at the insertion of a new piece of data.*

Regarding the number of facts, although it may seem as fairly high, one should remember that the representation of facts is very simple and individual facts take very little space. In the future, we will work further towards designing better forgetting strategies that will reduce the number of facts even further.

The experiments have shown that exchanging information based on context leads to good results, even if the representation of context is primitive.

7. Conclusion. One of the essential elements required for the implementation of ambient intelligence is a decentralized, self-organizing exchange of information between devices of reduced storage and processing capacity.

This paper describes a system for information exchange that includes the key concept of context-awareness, by using a simple but effective representation of context, comprised of two elements: pressure and interest. The system has been tested on a scenario involving a large number of agents and experiments have shown that the system, as a whole, is indeed considering context in its behaviour.

The paper does not refer to a specific application. The system and its behaviour have been kept as generic as possible, trying to deal with the general problem of context-aware information exchange.

REFERENCES

[1] J. AUGUSTO AND P. MCCULLAGH, *Ambient intelligence: Concepts and applications*, Computer Science and Information Systems/ComSIS, 4 (2007), pp. 1–26.

[2] C. BOURJOT, V. CHEVRIER, AND V. THOMAS, *A new swarm mechanism based on social spiders colonies: From web weaving to region detection*, Web Intelligence and Agent Systems, 1 (2003), pp. 47–64.

[3] G. CABRI, L. FERRARI, L. LEONARDI, AND F. ZAMBONELLI, *The LAICA project: Supporting ambient intelligence via agents and ad-hoc middleware*, Proceedings of WETICE 2005, 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 5 (2005), pp. 39–46.

[4] G. CHEN AND D. KOTZ, *A survey of context-aware mobile computing research*, tech. report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[5] K. DUCATEL, M. BOGDANOWICZ, F. SCAPOLO, J. LEIJTEN, AND J. BURGELMAN, *Istag scenarios for ambient intelligence in 2010*, tech. report, Office for Official Publications of the European Communities, 2001.

[6] M. GLEIZES, V. CAMPS, AND P. GLIZE, *A theory of emergent computation based on cooperative self-organization for adaptive artificial systems*, in Fourth European Congress of Systems Science, 1999.

[7] D. HALES AND B. EDMONDS, *Evolving social rationality for MAS using "tags"*, Proceedings of the second international joint conference on Autonomous agents and multiagent systems, (2003), pp. 497–503.

[8] B. JOHANSON, A. FOX, AND T. WINOGRAD, *The interactive workspaces project: Experiences with ubiquitous computing rooms*, IEEE pervasive computing, (2002), pp. 67–74.

[9] T. LECH AND L. WIENHOFEN, *AmbieAgents: a scalable infrastructure for mobile and context-aware information services*, Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, (2005), p. 631.

[10] J.-P. MANO, C. BOURJOT, G. LEOPARDO, AND P. GLIZE, *Bio-inspired mechanisms for artificial self-organised systems*, Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini, 30 (2006), pp. 55–62.

[11] A. OLARU AND A. M. FLOREA, *Emergence in cognitive multi-agent systems*, Proc. of CSCS17, 17th International Conference on Control Systems and Computer Science, (2009), pp. 515–522.

[12] A. OLARU, C. GRATIE, AND A. M. FLOREA, *Emergent properties for data distribution in a cognitive mas*, Proc. of IDC 2009, 3rd International Symposium on Intelligent Distributed Computing, (2009), pp. 151–159.

[13] G. PICARD, *Cooperative agent model instantiation to collective robotics*, in Engineering Societies in the Agents World V: 5th International Workshop, ESAW 2004, Toulouse, France, October 20-22, 2004: Revised Selected and Invited Papers, Springer, 2005.

[14] C. RAMOS, J. AUGUSTO, AND D. SHAPIRO, *Ambient intelligence - the next step for artificial intelligence*, IEEE Intelligent Systems, (2008), pp. 15–18.

[15] N. SADEH, F. GANDON, AND O. KWON, *Ambient intelligence: The MyCampus experience*, 2005.

[16] I. SATOH, *Mobile agents for ambient intelligence*, Proceedings of MMAS, (2004), pp. 187–201.

[17] A. E. F. Seghrouchni, *Intelligence ambiante, les defis scientifiques.* presentation, Colloque Intelligence Ambiante, Forum Atena, december 2008.
[18] G. D. M. Serugendo, M.-P. Gleizes, and A. Karageorgos., *Self-organization and emergence in MAS: An overview*, Informatica, 30 (2006), pp. 45–54.
[19] J. Viterbo, L. Mazuel, Y. Charif, M. Endler, N. Sabouret, K. Breitman, A. El Fallah Seghrouchni, and J. Briot, *Ambient intelligence: Management of distributed and heterogeneous context knowledge*, CRC Studies in Informatics Series. Chapman & Hall, (2008), pp. 1–44.
[20] M. Weiser, *Some computer science issues in ubiquitous computing*, Communications - ACM, (1993), pp. 74–87.
[21] F. Zambonelli, M. Gleizes, M. Mamei, and R. Tolksdorf, *Spray computers: Frontiers of self-organization for pervasive computing*, Proceedings of the 13th IEEE Int'l Workshops on Enabling Technologies, WETICE, (2004), pp. 403–408.

Edited by: Viorel Negru, Costin Bădică
Received: March 1, 2010
Accepted: March 31, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 43–52. http://www.scpe.org
ISSN 1895-1767

EFFICIENT BROADCASTING IN MANETS BY SELECTIVE FORWARDING

DOINA BEIN*, AJOY K. DATTA†, AND BALAJI ASHOK SATHYANARAYANAN†

Abstract. A major challenge faced in mobile ad hoc networks (MANETs) is locating devices for communication, especially in the case of high node mobility and sparse node density. Present solutions provided by the ad hoc routing protocols range from flooding the entire network with route requests, to deploying a separate location management scheme to maintain a device location database. Many applications as well as various unicast routing protocols such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it. Flooding is expensive in terms of overhead and wastes valuable resources such as bandwidth and power. We propose to develop a strategy to reduce the redundant transmission of packets in normal flooding used in broadcasting and we describe strategies for choosing only a minimal set of nodes to re-broadcast in grid networks. Our strategies reduce the redundant transmission of packets, thus packets are forwarded with a minimal number of transmissions. To determine the minimal set of nodes we propose a new algorithm called Efficient Broadcasting by Selective Forwarding (EBSF) that uses a distance-based approach in selecting the nodes among all the nodes in a grid network. The distance-based approach is implemented for broadcast and rebroadcast to a set of nodes with the help of a threshold value that reduces the number of redundant transmission. This threshold value can be tuned to show the performance enhancement.

Key words: broadcast, distance-based approach, grid network, mobile network, routing

1. Introduction. A mobile ad hoc network (MANET) is an autonomous system of mobile routers, connected by wireless links, the union of which forming an arbitrary graph (see Figure 1.1). The routers are free to move and organize themselves arbitrarily. Thus, the topology may change rapidly and unpredictably. Such a network may operate in a stand-alone fashion or may be connected to the larger network. In mobile ad hoc networks, it is often necessary to broadcast control information to all constituent nodes in the network. Blind flooding [14] is often deployed to achieve the above objective. Its advantages, the simplicity and the guarantee that every destination in the network is reached, are downsized by the fact that it is expensive in terms of overhead and wastes valuable resources such as bandwidth and power:

- Some routers receive a packet multiple times.
- It leads to transmission of redundant packets.
- Packets can go in a loop forever.
- For dense networks, it causes significant contention and collisions—the so-called *broadcast storm problem.*

Broadcasting, the process by which one node needs to communicate a packet to all other nodes in the network, is a critical mechanism for information diffusion and maintaining consistent global network information. Additionally, it is an energy intensive function. Broadcasting or a derivation of it is used by routing protocols such as Dynamic Source Routing (DSR) [11], Ad Hoc On Demand Distance Vector (AODV) [21], Location Aided Routing (LAR) [13, 15], and Zone Routing Protocol (ZRP) [9], for establishing routing paths. Currently all these protocols rely on a simple form of broadcasting called *blind flooding.* In blind flooding, each node re-broadcast a packet whenever it receives it for the first time: Every incoming packet is sent out on every outgoing line except the one it arrived on. Blind flooding generates many redundant transmissions, which may cause a more serious broadcast storm problem [17]. Given the expensive and limited nature of wireless resources such as bandwidth and battery power, minimizing the control message overhead for route discovery is a high priority in protocol design. Recently, a number of research groups have proposed more efficient broadcasting techniques. Centralized broadcasting schemes are presented in [1, 4, 8]. The algorithms in [16, 25, 19, 20, 22, 23] utilize neighborhood information to reduce the number of redundant messages.

In this paper we present a new protocol called *Efficient Broadcasting by Selective Forwarding* (EBSF) that minimizes the number of transmissions or retransmissions needed for broadcasting using a distance-based approach [2]. In our proposed protocol, only a set of selected nodes are allowed to do the broadcast; these nodes are selected using a so called *threshold distance* [18, 3]. We set the threshold distance to be

$threshold = n \times transmission\_radius$, where n is a real number. This threshold value can be tuned to show performance enhancement, thus minimizing the number of transmissions or retransmissions needed. If

*Applied Research Laboratory, The Pennsylvania State University, University Park, PA 16802, USA (siona@psu.edu).
†School of Computer Science, University of Nevada, Las Vegas, NV 89154, USA (datta@cs.unlv.edu, ashoksathyan@yahoo.com).

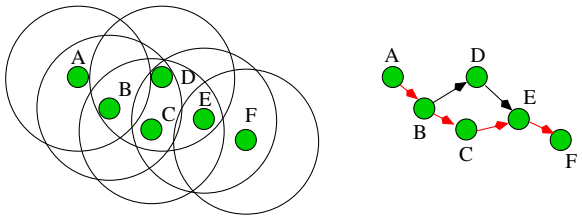


FIG. 1.1. *Example of MANET*

the threshold distance is set to 0, selective broadcasting will be exactly blind flooding. To check for message duplication, thus to reduce the number of redundant messages to be delivered, we implement a new data structure called *message cache*.

We have simulated the communication network using a network simulator called GLOMOSIM [5, 6, 7]. GLOMOSIM (GLObal MObile System SIMulator) is a library-based sequential and parallel simulator for wireless networks. The library of GLOMOSIM is based on the parallel discrete-event simulation capability provided by Parsec, the compiler used in GLOMOSIM. Using GLOMOSIM we show the performance variation of our algorithm using different threshold factors.

In Section 2 we present an overview of various broadcasting schemes followed by the motivation of why blind flooding is not efficient for broadcasting in MANETs. In Section 3 we define efficient broadcasting, followed by a description of our protocol and its advantages. A discussion regarding the simulation results and the performance analysis is given in Section 4. We finish with concluding remarks and present concepts for future research in Section 5.

2. Preliminaries. Broadcasting is a communication paradigm that allows sending data packets from source to multiple receivers. Broadcasting in wireless ad-hoc networks is a critical mechanism for applications such as information diffusion, wireless networks and also for maintaining consistent global network information. Broadcasting or a derivation of it is often necessary in MANET routing protocols such as Dynamic source routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Location Aided Routing (LAR), Zone Routing Protocol (ZRP) for establishing routes. Currently all these protocols rely on a simplistic form of broadcasting called *flooding*. Flooding is a process in which each node (or all nodes in a localized area) retransmits each received unique packet exactly one time. In flooding, every incoming packet is sent out on every outgoing line except the one it arrived on. Whenever a device connected to the LAN (Local Area Network) switch sends a packet to an address that is not in the LAN switchs table or whenever the device sends a broadcast or multicast packet, the switch sends the packet out to all ports. This is referred to as *blind flooding*. The routers forward packets to all ports except the ingress port. There are two advantages to blind flooding: simplicity and the fact that every node in the network is eventually reached. But there are many disadvantages: receiving the same packet multiple times that leads to redundant transmissions, packets can go in a loop forever, inefficient use of bandwidth, high power consumption.

An improvement to blind flooding is to choose only a subset of nodes to re-broadcast and in this manner to reduce the number of data transmissions. Several alternatives are presented next. The probabilistic scheme [23, 24, 10, 12] is similar to blind flooding, except that nodes only re-broadcast with a predefined probability. Since some nodes do not re-broadcast, node and network resources are saved without having delivery effectiveness. In sparse networks, nodes will not receive all broadcast packets unless the probability parameter is high. When the probability is 100%, this scheme is identical to blind flooding.

There is an inverse relationship between the number of times a packet is received at a node and the probability of that node being able to reach some additional area on a broadcast. This result is the basis of the counter-based scheme [17, 23, 24]. Upon reception of a packet never received before, the node initiates a counter with value 1 and sets a random assessment delay (RAD) timeout. During the RAD, the counter is incremented for each redundant packet received. If the counter is less than a threshold value when the RAD expires, then the packet is re-broadcast, otherwise the packet is simply dropped.

The probabilistic and counter-based schemes are simple and inherently adaptive to the local topologies. Their disadvantage is that the delivery is not guaranteed to all nodes even if the ideal Media Access Control (MAC) layer is provided. (The MAC layer at a node provides addressing and access control mechanisms of the

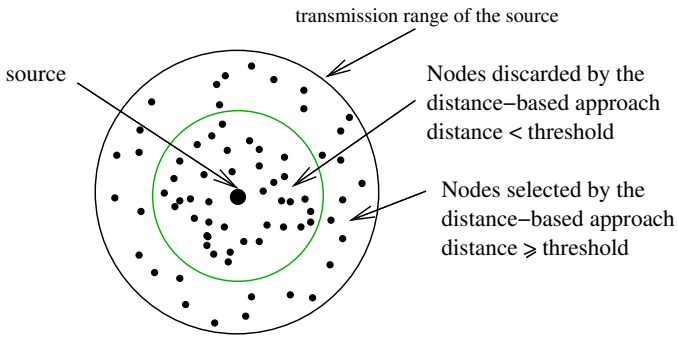


FIG. 3.1. *Efficient Broadcasting by Selective Forwarding*

communication channels of the node that allows the node to send and receive packets in a network.) In other words, both schemes are unreliable.

Area-based methods [24] only consider the coverage area of transmission and do not consider whether there are nodes within that area. A node decides whether to re-broadcast purely based on its own information. There are two coverage area-based methods [23]: distance-based and location-based schemes. In this paper we have developed a new approach based on the physical distance [23] by which the data transmitted is reduced considerably to a greater extent, while ensuring that all data is received by all nodes in the network. In our approach, broadcasting is done by some selected nodes that are privileged to broadcast based on so-called *threshold distance*. If the threshold distance is set to 0 then the network broadcast becomes blind flooding.

A uniform distribution of nodes is an effective way to organize a large network. Flooding provides important control and route establishment functionality for a number of unicast and multicast protocols in ad hoc networks. Considering its wide use as a building layer for other network layer protocols, the flooding method should deliver a packet from one node to all other network nodes using as few messages as possible, including control information. This makes the network wide broadcasting an energy intensive function. An improvement to flooding is to choose only a subset of nodes to rebroadcast and thus reduce data transmissions. To this end, we have devised a mechanism where an optimal node selection is done to determine the minimal set of nodes for broadcast in mobile ad hoc networks.

3. Efficient Broadcasting by Selective Forwarding. EBSF reduces the redundant transmission of packets used in broadcasting. An optimal selection is done to determine the minimal set of nodes to re-broadcast. To determine this minimal set of nodes we employ the distance-based approach [23]. We assume that the nodes are placed on a grid and all nodes have the same transmission range. We also assume that the transmission range is constant. The distance-based approach is implemented with the help of a threshold value that is taken to be $threshold = N \times transmission\_radius$, where N has values $0 < N < 1$.

In blind flooding, location discovery is achieved by allowing all nodes that receive some message to retransmit it. In our case we prevent any node within a distance factor which is the threshold to re-transmit. Thus a few selected nodes will be selected for retransmission, and subsequently we reduce the number of redundant transmissions (see Figure 3.1). This threshold value can be tuned as per requirements. It can be increased and decreased to show the performance variation. Higher the threshold, lesser is the number of packets transmitted by each node in the network and vice versa. If threshold is set to 0, then all packets are transmitted like in blind flooding (no enhancement). The advantages of EBSF are: the selection of optimal number of nodes to retransmit, an effective utilization of bandwidth, minimizing the number of unnecessary transmission and therefore reducing the redundant packets, and minimizing the power consumption. We made the following assumptions:

1. The nodes have uniform transmission power (they have the same transmission radius).
2. The nodes are placed in a grid-like topology.
3. The nodes are almost uniformly distributed in the network.

We show next where our proposed EBSF protocol is executed within the OSI layers.

In the OSI model, upper layers deal with the application's issues and generally are implemented only in software. The highest layer, the application layer, is closest to the end user. The lower layers of the OSI model handle data transport issues, namely communications between network devices. When a packet is to be

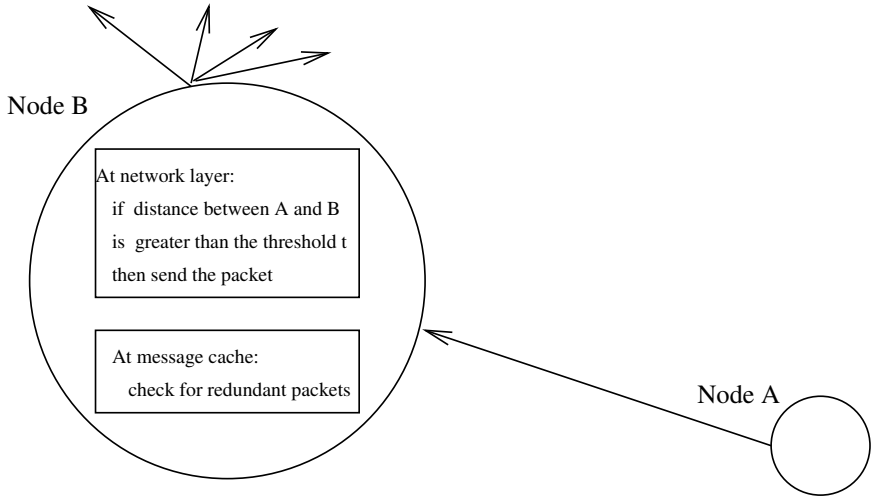


FIG. 3.2. *EBSF protocol*

transmitted from a source node to a destination node, the network layer (Layer 3) is responsible for end-to-end (source to destination) packet delivery, whereas the data link layer (Layer 2) is responsible for node-to-node (hop-to-hop) packet delivery. When a node receives a packet, the MAC layer of the data link layer sends the packet to the network layer for transmission. Efficient Broadcasting by Selective Forwarding (EBSF) is implemented in the network layer.

When a packet is forwarded to the network layer for transmission, the IP header in the Internet Protocol has the source node and the destination node coordinates. Based on these coordinates, the distance between the source and destination nodes cam be easily computed. In blind flooding, whenever a message is received and needs to be sent further, at the network layer the packet is broadcast to all the nodes within its transmission range once it is received from the MAC layer.

In EBSF protocol, when a packet reaches the network layer of a node for broadcasting some computation is performed as follows (see Figure 3.2). Based on the information about the source and the destination taken from the IP header, EBSF computes the distance between the source and the destination. When this distance is less than the threshold distance, the node discards the message. If this distance is greater than the threshold distance, then EBSF broadcasts the message to the nodes at a distance equal to or greater than the threshold distance.

When a node A broadcasts p packets to other nodes in the network, each node has a message cache, called MSGCACHE, which stores information about messages the node has received. When a node receives a packet, it checks with the MSGCACHE whether the packet has already been received. In affirmative, the packet is discarded; otherwise, EBSF forwards it to the network layer for transmission. In this manner, the redundant packets are discarded. MSGCACHE is implemented as a linked-list data structure. In EBSF, at any node every packet is checked for message redundancy using MSGCACHE and then send it to the network layer for broadcast. The network layer will decide, based on the threshold distance between the source node and destination nodes, whether to broadcast or not, as follows. Based on the information about the source and the destination in the IP header, the distance between the source node and the destination node is calculated. When this distance is less than the threshold distance, the node discards the message. If this distance is greater than the threshold distance, the node will broadcast the message to the nodes at a distance equal or greater than the threshold distance. This process is repeated until all the nodes in the network receive all the packets originated from the source node.

For further enhancement, we partition the neighbors of a node into two types, *communicational neighbors* and *geographical neighbors*. Communicational neighbors are the ones within the node's transmission range.Geographical neighbors of a node in a grid network are the nodes located at a distance of one grid unit: a corner node has two geographical neighbors and a border node has three geographical neighbors; all other nodes have four geographical neighbors. For example, in Figure 3.3, the geographical neighbors of node E are nodes B, D, F, H. The corner node G has two geographical neighbors, nodes D and H. The border node H has three geographical neighbors G, E, I.

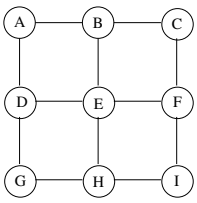


FIG. 3.3. *Geographical Nodes*

TABLE 4.1
Layers of GLOMOSIM

Layers	**Protocols**
Mobility	Random waypoint, Random drunken, Trace based
Radio Propagation	Two ray and Free space
Radio Model	Noise Accumulating
Packet Reception Models	SNR bounded, BER based with BPSK / QPSK modulation
Data Link (MAC)	CSMA, IEEE 802.11 and MACA
Networking (Routing)	IP with AODV, Bellman-Ford, DSR, Fisheye, LAR scheme 1, ODMRP
Transport	TCP and UDP
Application	CBR, FTP, HTTP and Telnet

We have enhanced EBSF to exclude to broadcast to the geographical neighbors of a node situated within the threshold distance, and thereby decreasing the number of transmissions. For example, given a sender node S, if some node R is at a distance greater than or equal to the threshold distance of node S, then more likely the geographical nodes of node R are also at threshold distance from node S. In this case, in EBSF node R and the geographical neighbors of R are allowed to broadcast the message received from node S. When node R and its geographical neighbors may have common nodes which are at a threshold distance from them, these nodes will receive the same messages from node R and its geographical neighbors, which would lead to redundant transmissions. Hence, with the enhanced EBSF algorithm we restrict the geographical neighbors of a node to not broadcast and thereby reduce redundant transmission, which is evident from the simulation results.

4. Simulations. The tool we have used for the simulation is GLOMOSIM. GLOMOSIM is a scalable simulation environment for wireless and wired systems, designed using the parallel discrete-event simulation capability provided by Parsec. Parsec compiler is similar to the C compiler with some added functions. The simulation statistics are stored in the "bin" directory of GLOMOSIM as "glomo.stat" file. The layers in GLOMOSIM are shown in Table 1.

GLOMOSIM contains the following directories: `application` contains code for the application layer, `bin` for executable and input or output files, `doc` contains the documentation, `include` contains common include files, `java gui` contains the visual tool, `Mac` contains the code for the MAC layer, `main` contains the basic framework design, `network` contains the code for the network layer, `radio` contains the code for the radio layer, `scenarios` contains some example scenarios, `tcplib` contains libraries for TCP, and `transport` contains the code for the transport layer.

In Figure 4.1 we present the files that we have been modified in GLOMOSIM.

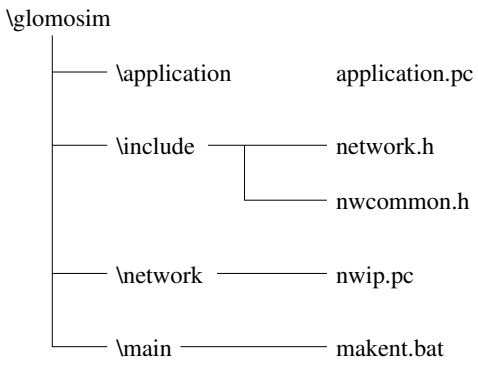


FIG. 4.1. *Files modified in GLOMOSIM*

All routing decisions are taken in the network layer, so we embed our protocol EBSF in the network layer of each node of the network. Whenever the packet enters the network layer, the packet will be handled by EBSF. For our protocol EBSF to work in the network layer of GLOMOSIM, we have modified and included the files EBSF.H and EBSF.PC in the directory `network` of GLOMOSIM. The program written in the file EBSF.PC contains the code for threshold computation and the code to determine whether to broadcast or to discard a message.

Whenever the packet reaches the network layer, it will be handled by protocol EBSF as follows (see Figure 4.2). At the beginning, the initialization function $RoutingEBSFInit()$ is called and defines a structure called $GlomoRoutingEBSF$ and allocates memory to it. Also the stats (statistics to be printed to determine the enhancements) are initialized, the sequence table for the node, and message cache. The top value is initialized to 0.

A routing function $RoutingEBSFRouter()$ determines the routing action to be taken and handles the packet accordingly (if it is from UDP or MAC). If the packet is from UDP, then the node is the source node and the data is sent, i. e., the function $RoutingEBSFSendData()$ is called. If the data comes from the MAC layer, then the decision is made as whether to send it to UDP or drop the packet. This decision is made in the function $RoutingEBSFHandleData()$.

The function $Finalize()$ initializes the statistics part of the protocol. When this function is called, it collects the statistics from the file "glomo.stat" in the "bin" directory and formats the statistics such that number of data transmitted, the number of data originated, and the number of data received for each node are printed.

The function $RoutingEBSFHandleData()$ is called whenever the node receives the packet from the MAC layer. This function checks with the message cache and it decides whether to transmit or discard the packet. The nodes that are with in the transmission range will receive the packet and only the nodes that are greater than threshold distance and lesser than transmission range will transmit the packet. If the node is at a distance lesser than a threshold value from a transmitting node, then the packet is discarded.

The message cache is implemented as a linked-list, to which an element is inserted whenever a new message arrives at a node; the element contains the source address and the message's sequence number. The function $LookupMessageCache$ searches the message cache to see whether the message already exists, using its sequence number. The function $InsertMessageCache$ inserts a message into the cache if it is not already present there. Other packet functions are $GLOMO\_MsgAlloc$, $GLOMO\_MsgAddHeader$, $GLOMO\_MsgRemoveHeader$, and *GLOMO_MsgPack-etAlloc*.

The PARSEC compiler, called `pcc`, accepts all the options supported by the C compiler, and also supports separate compilation. C programs (files with .c suffix) and object files (files with .o suffix) can also be compiled and linked with PARSEC programs. PARSEC programs are usually given an extension `.pc`. PARSEC supports separate compilation of entities.

Figure 4.3 shows the GUI of the GLOMOSIM simulator for a network of 49 nodes arranged as a grid of 7x7 and in which node 35 transmits messages to all other nodes.

The threshold distance is defined as $N \times transmission\_range$ of a node, where N is a real number between 0 and 1.

We have implemented and tested our protocol for a network with $n = 49$ nodes (a perfect grid) and various values of the threshold (see Figure 4.4).

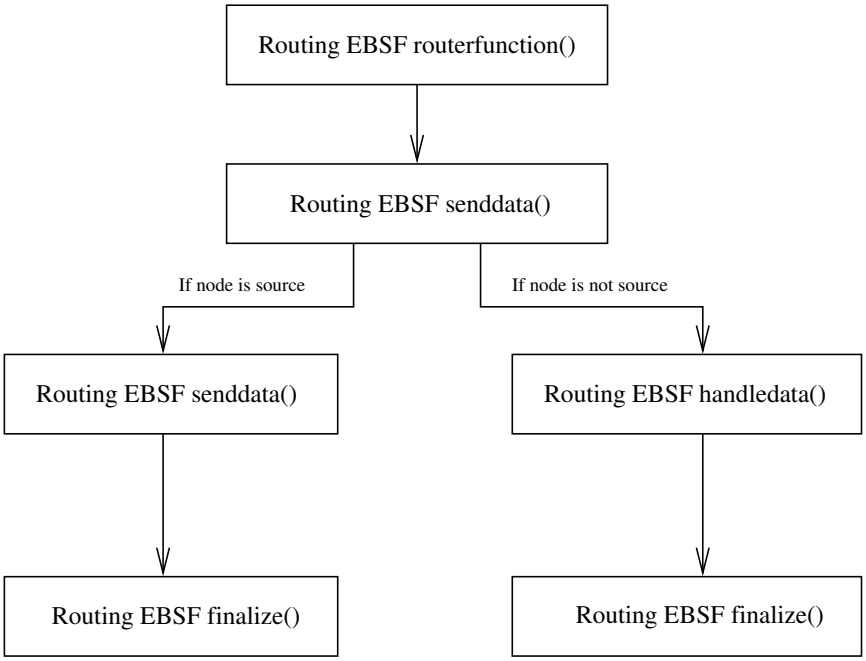


FIG. 4.2. *Module Diagram of EBSF*

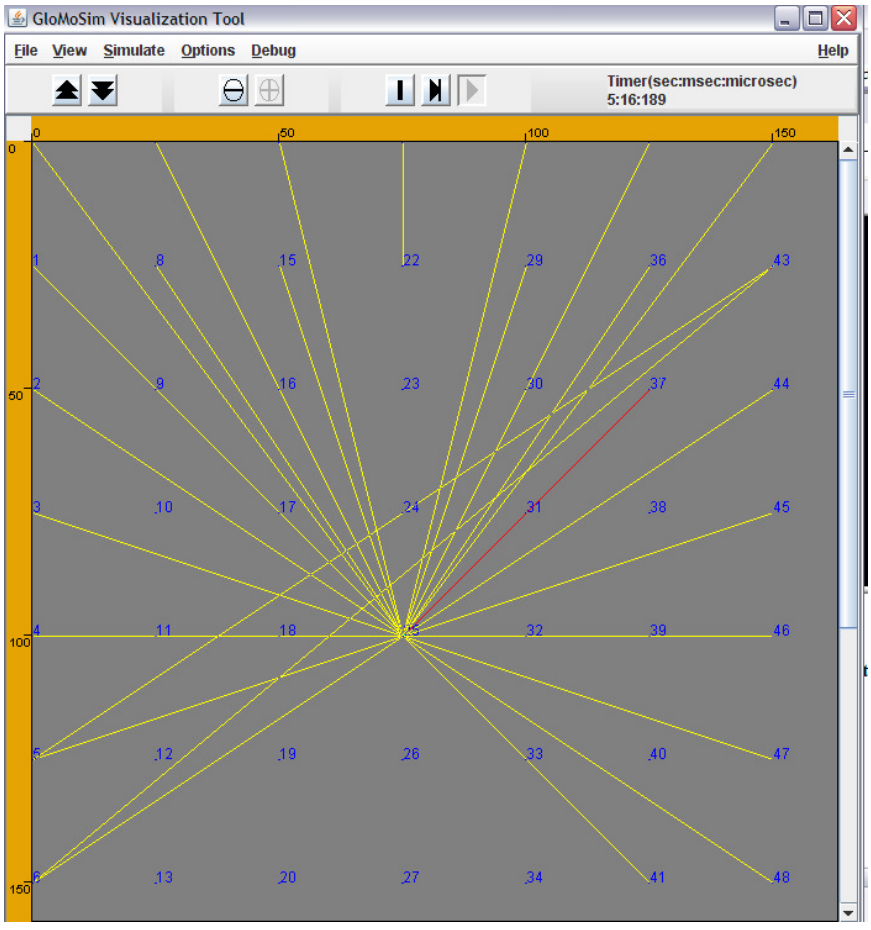


FIG. 4.3. *GUI of GLOMOSIM for a 49-node network; node 35 is the sender*

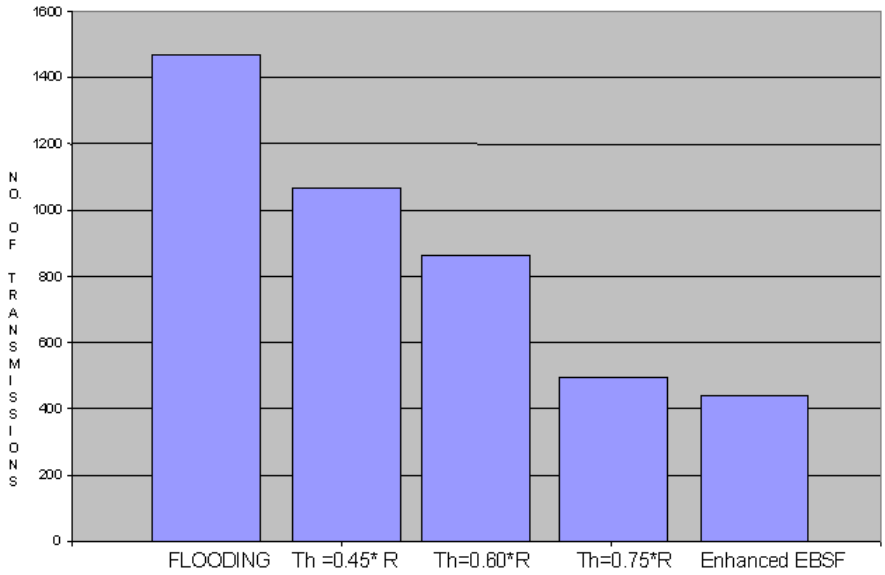


FIG. 4.4. *Performance Analysis with Different Threshold Values*

The performance statistics are presented in Table 4.2.

5. Conclusion. Building efficient protocols for ad hoc networks is a challenging problem due to the dynamic nature of the nodes. Unlike traditional fixed-infrastructure networks, in MANETs there is no centralized control over the network or over the size of the network. The proposed Efficient Broadcasting by Selective Forwarding has a number of advantages over other approaches considered in the literature. First of all, it selects a minimal number of nodes to retransmit, thus it utilizes the bandwidth efficiently. Secondly, it minimizes the number of unnecessary transmissions and therefore it reduces the number of redundant packets. Thirdly, it minimizes the overall power consumption since only the selected nodes use their power for re-transmission. The selected nodes are chosen based on the threshold value; the threshold value can be tuned to show the performance enhancements. Higher the threshold value, more optimized results are obtained. EBSF does not impose any bandwidth overhead and reduces the power consumption drastically. The efficiency of EBSF remains very high even in large networks. Overall, the proposed protocol shows that broadcasting can be enhanced greatly by choosing only an optimal set of nodes for transmission and thus avoiding redundant transmissions and at the same time ensuring data delivery to all the nodes in the network.

This protocol could be integrated with any routing protocol for finding a route in mobile ad-hoc networks with minimal power consumption and without imposing any bandwidth overhead.

Current research in wireless networks focuses on networks where nodes themselves are responsible for building and maintaining proper routing (self-configure, self-managing). Our algorithm does not adapt to topology changes; this is a topic of future research. If nodes are missing from the grid, the threshold value needs to be decreased; in case new nodes are added, the threshold value needs to increase to keep the performance of the protocol. This increasing or decreasing has to be done dynamically, and better in a non-centralized manner. If new nodes are added to the network, then the set of selected nodes may need to be recomputed. It is desirable to minimize the number of such computations while stile preserving the properties of EBSF.

Acknowledgments. The authors would like to thank the anonymous reviewers for the valuable comments that have improved the quality of the paper.

TABLE 4.2
Performance Statistics

Node	**Flooding**	**EBSF** threshold=0.45*R	**EBSF** threshold=0.6*R	**EBSF** threshold=0.75*R
0	30	0	0	0
1	30	30	30	30
2	30	0	0	0
3	30	0	0	0
4	30	30	30	0
5	30	30	30	30
6	30	21	7	8
7	30	0	0	0
8	30	0	0	0
9	30	0	0	0
10	30	30	0	0
11	30	30	30	0
12	30	30	30	30
13	30	20	8	10
14	30	30	0	0
15	30	0	0	0
16	30	30	0	0
17	30	30	0	0
18	30	30	30	30
19	30	30	30	30
20	30	22	9	8
21	30	30	30	0
22	30	30	30	0
23	30	30	30	0
24	30	30	30	30
25	30	30	30	30
26	30	18	14	4
27	30	22	16	2
28	30	30	30	30
29	30	30	30	30
30	30	30	30	30
31	30	30	30	30
32	30	22	16	10
33	30	22	16	9
34	30	25	18	8
35	30	20	13	12
36	30	22	10	9
37	30	26	9	6
38	30	27	12	6
39	30	21	16	8
40	30	24	17	7
41	30	27	19	12
42	30	25	17	13
43	30	25	19	13
44	30	28	21	10
45	30	26	20	9
46	30	22	22	12
47	30	25	21	12
48	30	27	22	22

REFERENCES

[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *J. Comput. Syst. Sci.*, 43:290–298, October 1991.

[2] X. Chen, M. Faloutsos, and S. Krishnamurthy. Distance Adaptive (DAD) broadcasting for ad hoc networks. In *Proceedings of MILCOM*, vol. 2, October 2002.

[3] S.-J. Choi, K.-H. Kwon, and S.-J. Yoo. An efficient cross-layer based flooding algorithm with retransmission node selection for wireless sensor networks. In *22nd International Conference on Advanced Information Networking and Applications - Workshops (AINA Workshops 2008)*, pp.941-948, March 25-28, 2008.

[4] I. Gaber and Y. Mansour. Broadcast in radio networks. In *Proceedings of SODA*, pages 577–585, January 1995.

[5] GLOMOSIM. Installation of GLOMOSIM 2.03 in windows XP. `http://www.cs.ndsu.edu/~ahmed/GLOMOSIM.html`

[6] GLOMOSIM. `http://pcl.cs.ucla.edu/projects/GLOMOSIM/`

[7] GLOMOSIM. `http://www.sm.luth.se/csee/courses/smd/161\_wireless/glomoman.pdf`

[8] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *Proceedings of ESA*, 1996.

[9] J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of 6th IEEE International Conference on Universal Personal Communications (ICUPC)*, pages 562–566, October 1997.

[10] A. M. Hanashia, A. Siddique, I. Awan, and M. Woodward. Performance evaluation of dynamic probabilistic broadcasting for flooding in mobile ad hoc networks. Simulation Modelling Practice and Theory, 17(2):364–375, February 2009.

[11] D. B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994.
[12] N. Karthikeyan, V. Palanisamy, and K. Duraiswamy. Optimum density based model for probabilistic flooding protocol in mobile ad hoc network. European Journal of Scientific Research, 39(4):577–588, 2010.
[13] Y.-B. Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of ACM/IEEE Mobicom*, pages 66–75, October 1998.
[14] Y. B. Ko and N. H. Vaidya. Location-aided routing in mobile ad hoc networks. *Technical report* 98-012, Texas A&M University, 1998.
[15] Y.-B. Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6:307–321, 2000.
[16] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *Proceedings of the ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2000.
[17] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of ACM MOBICOM*, pages 151–162, August 1999.
[18] V. K. Paruchuri, A. Durresi, and R. Jain. Optimized flooding protocol for ad hoc networks. IEEE Transactions on Parallel and Distributed Systems, 15(11):1027–1040, November 2004.
[19] W. Peng and X. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proceedings of MOBIHOC*, 2000.
[20] W. Peng and X. Lu. Ahbp: an efficient broadcast protocol for mobile ad hoc networks. *Journal of Science and Technology*, 2002.
[21] C. Perkins and S. Das. Ad hoc on-demand distance vector (aodv) routing. http://tools.ietf.org/html/rfc3561, July 2003.
[22] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying: an efficient technique for flooding in mobile wireless networks. Technical Report Technical Report 3898, INRIA - Rapport de recherche, 2000.
[23] J. Sucec and I. Marsic. An efficient distributed networkwide broadcast algorithm for mobile ad hoc networks. Technical Report CAIP Technical Report 248, Rutgers University, September 2000.
[24] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the ACM MOBIHOC*, pages 194–205, 2002.
[25] J. Wu and F. Dai. Broadcasting in ad hoc networks based on self-pruning. In *Proceedings of IEEE INFOCOM*, 2003.

Edited by: Costin Bădică, Adina Magda Florea
Received: March 4, 2010
Accepted: March 31, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 53–62. http://www.scpe.org
ISSN 1895-1767

EXPLORING CARRIER AGENTS IN SWARM-ARRAY COMPUTING

BLESSON VARGHESE* AND GERARD MCKEE†

Abstract. How can a bridge be built between autonomic computing approaches and parallel computing systems? The work reported in this paper is motivated towards bridging this gap by proposing a swarm-array computing approach based on 'Intelligent Agents' to achieve autonomy for distributed parallel computing systems. In the proposed approach, a task to be executed on parallel computing cores is carried onto a computing core by carrier agents that can seamlessly transfer between processing cores in the event of a predicted failure. The cognitive capabilities of the carrier agents on a parallel processing core serves in achieving the self-ware objectives of autonomic computing, hence applying autonomic computing concepts for the benefit of parallel computing systems. The feasibility of the proposed approach is validated by simulation studies using a multi-agent simulator on an FPGA (Field-Programmable Gate Array) and experimental studies using MPI (Message Passing Interface) on a computer cluster. Preliminary results confirm that applying autonomic computing principles to parallel computing systems is beneficial.

Key words: swarm-array computing, intelligent agents, carrier agents, autonomic computing

1. Introduction. Inspirations from nature have led computing scientists to focus on biologically inspired computing paradigms. Amorphous computing [1], evolutionary computing [2] and organic computing [3] are such areas that focus on abstracting designs from nature. Lately, autonomic computing inspired by the autonomic human nervous system [4] is the emphasis of distributed computing researchers which is considered in this paper.

With the aim of building large scale systems [5], reducing cost of ownership [6] [7] and reallocating management responsibilities from administrators to the computing system itself [8][9][10], autonomic computing principles have paved necessary foundations towards self-managing systems. Self-managing systems are characterized by four objectives, namely self-configuration, self-healing, self-optimizing and self-protecting and four attributes, namely self-awareness, self-situated, self-monitoring and self-adjusting [4][11][12].

Autonomic computing researchers have adopted six different approaches, namely emergence-based, component/service-based, control theoretic based, artificial intelligence, swarm intelligence and agent based approaches to achieve self-managing systems.

The emergence based approach for distributed systems considers complex behaviours of simple entities with simple behaviours without global knowledge [13]. Autonomic computing research on emergence based approaches is reported in [13] and [14].

The component/service based approach for distributed systems employ service-oriented architectures, and is implemented in many web based services. These approaches are being developed for large scale networked systems including grids. Autonomic computing research on component/service based approaches is reported in [15], [16] and [17].

The control theoretic based approach aims to apply control theory for developing autonomic computing systems. The building blocks of control theory are used to model computing systems and further used to study system properties. Using a defined set of control theory methodologies, the objectives of a control system can be achieved. Research on control theoretic based approaches applied to autonomic computing is reported in [18] and [19].

The artificial intelligence based approaches aim for automated decision making and the design of rational agents. The concept of autonomy is realized by maximizing an agent's objective based on perception and action in the agent's environment with the aid of information from sensors and in-built knowledge. Work on artificial intelligence approaches for autonomic computing is reported in [20] and [21].

The swarm intelligence based approaches focus on designing algorithms based on collective behaviour of swarm units that arise from local interactions with their environment. The algorithms considered are population-based stochastic methods executed on distributed processors. Autonomic computing research on swarm intelligence approaches is reported in [22] and [23].

The agent based approaches for distributed systems is a generic technique adopted to implement emergence, component/service, artificial intelligence or swarm intelligence based approaches. The agents act as

*Active Robotics Laboratory, School of Systems Engineering, University of Reading, Whiteknights Campus, Reading, Berkshire, United Kingdom, RG6 6AY (b.varghese@student.reading.ac.uk).

†School of Systems Engineering, University of Reading, Whiteknights Campus, Reading, Berkshire, United Kingdom, RG6 6AY (g.t.mckee@reading.ac.uk).

autonomic elements or entities that perform distributed task. The domain of software engineering considers agents to facilitate autonomy and hence have a profound impact on achieving the objectives of autonomic computing. Research work based on multi-agents supporting autonomic computing are reported in [9], [24] and [25].

However, though all of the autonomic computing approaches above aim towards the objectives of autonomic computing, few researchers have applied autonomic computing concepts to parallel computing systems. This is surprising since most distributed computing systems are closely associated with the parallel computing paradigm. The benefits of autonomy in computing systems, namely reducing cost of ownership and reallocating management responsibilities to the system itself are also relevant to parallel computing systems.

How can a bridge be built between autonomic computing approaches and parallel computing system? The work reported in this paper is motivated towards bridging this gap by proposing a swarm-array computing approach, that aims to achieve autonomy for distributed parallel computing systems.

Swarm-array computing is biologically inspired by the theory of autonomous agents in natural swarms that are abstracted and implemented in parallel computing systems. This technique considers the computational resource as a swarm of resources and the task to be executed as a swarm of sub-tasks. Hence, the approach considers complex interactions between swarms of sub-tasks and swarms of resources. These interactions bring about the notion of intelligent agents or swarm agents carrying the sub-tasks and intelligent cores or swarm of cores executing the sub-task.

In this paper, a swarm-array computing approach is proposed as a solution that aims to apply autonomic concepts to parallel computing systems and in effect achieve the objectives and attributes of self-managing systems. Unlike another swarm-array computing approach reported in [26], the approach proposed in this paper considers the task to be executed on parallel computing cores as a swarm of autonomous agents.

The remainder of the paper is organized as follows. Section 2 considers the proposed swarm-array computing approach. The second approach is of focus in this paper. Section 3 investigates the feasibility of the proposed approach by considering simulations. Section 4 presents the implementation of the proposed approach on a computer cluster. Section 5 concludes the paper by considering future work.

2. Swarm-Array Computing Approach. Swarm-array computing is a swarm robotics inspired approach that is proposed as a path to achieve autonomy in parallel computing systems. The foundations of swarm-array computing are the existing computing paradigms of parallel and autonomic computing. There are three approaches based on intelligent cores, intelligent agents and a hybrid approach based on both intelligent cores and intelligent agents that bind the swarm-array computing constituents together [26].

In this paper, the focus is on the second approach based on intelligent agents. The aim of the 'Intelligent Agent based' approach is to demonstrate that the cognitive capabilities of an agent complementing its intelligence can be used to achieve the objectives and attributes of autonomic computing.

In the intelligent agent based approach, a task to be executed on a parallel computing system is decomposed into sub-tasks and mapped onto agents that carry these tasks onto nodes or cores for execution. The agent and the sub-problem are independent of each other; in other words, the agents only carry the sub-tasks or act as a wrapper around the sub-task independent of the operations performed by the task.

In the proposed approach, an agent has capabilities similar to the capabilities of a natural agent presented above. Intelligence of an agent in the computing environment is demonstrated in four different ways. Firstly, an agent is aware of its environment, that is the nodes or cores on which it can carry a task onto, other agents in its vicinity and agents with which it interacts or shares information. Secondly, an agent can situate itself on a node or core that may not fail soon and can provide necessary and sufficient consistency in executing the task. Thirdly, an agent can predict core failures by consistent monitoring (for example, power consumption and heat dissipation of the cores can be used to predict failures). Fourthly, an agent is capable of shifting gracefully from one core to another, without causing interruption to the state of execution, and notifying other interacting agents in the system when a core on which a sub-task being executed is predicted to fail.

Hence, objectives such as self-configuration, self-healing and self-optimizing and attributes such as self-awareness, self-situated, self-monitoring and self-adjusting are inherently obtained in the proposed approach by the cognitive capabilities demonstrated by the agent.

3. Simulation Studies. Simulation studies were pursued to validate and visualize the proposed intelligent agent based approach. Various simulation platforms were considered, namely network simulators, which could predict behaviours of data packets in networks, and multi-agent simulators, that could model agents and their

behaviours in an environment. Since FPGA cores are considered in this paper, network simulators were not an appropriate choice. The approach proposed in this paper considers executing cores as agents; hence a multi-agent simulator is employed. This section is organized into describing the experimental environment, modelling the experiment and experimental results.

With the objective of exploring swarm-array computing, FPGAs are selected as an experimental platform for simulating the proposed approaches. FPGAs are a technology under investigation in which the cores of the computing system are not geographically distributed. The cores in close proximity can be configured to achieve a regular grid or a two dimensional lattice structure. Another reason of choice to look into FPGAs is its flexibility for implementing reconfigurable computing.

The feasibility of the proposed swarm-array computing approach was validated on the SeSAm (Shell for Simulated Agent Systems) simulator. The SeSAm simulator environment supports the modelling of complex agent-based models and their visualization [27][28].

The environment has provisions for modelling agents, the world and simulation runs. Agents are characterized implemented in the form of an activity diagram by a reasoning engine and a set of state variables. The state variables of the agent specify the state of an agent. The world provides knowledge about the surroundings in which the agent is situated. A world is also characterized by variables and behaviours and defines the external influences that can affect the global behaviour of the agent. Simulation runs are defined by simulation elements, namely situations, analysis lists, simulations and experiments that contribute to the agent-based model being constructed.

As considered in Section 2, the swarm-array computing approach considers the computing platform and the problem/task. An FPGA is modelled in the SeSAm environment such that the hardware cores are arranged in a 5 X 5 regular grid structure. The model assumes serial bus connectivity between individual cores. Hence, a task scheduled on a core can be transferred onto any other core in the regular grid.

The breakdown of any given task into subtasks is not considered within the problem domain of swarm-array computing. The simulation is initialized with sub-tasks scheduled to a few cores in the FPGA grid. Each sub-task carrying agent consistently monitors the hardware cores. This is possible by sensory information (in our model, temperature is sensed consistently) passed onto the carrier agent. In the event of a predicted failure, the carrier agent displaces itself to another core in the computing system. The behaviour of the individual cores vary randomly in the simulation. For example, the temperature of the FPGA core changes during simulation. If the temperature of a core exceeds a predefined threshold, the subtask being executed on the core is transferred by the carrier agent onto another available core that is not predicted to fail. During the event of a transfer or reassignment, a record of the status of execution of the subtask maintained by the carrier agent also gets transferred to the new core. If more than one sub-task is executed on a core predicted to fail, each sub-task may be transferred to different cores.

Figure 3.1 is a series of screenshots of a random simulation run developed on SeSAm for nine consecutive time steps from initialization. The figure shows the executing cores as rectangular blocks in pale yellow colour. When a core is predicted to fail, i. e., temperature increases beyond a threshold, the core is displayed in red. The subtasks wrapped by the carrier agents are shown as blue filled circles that occupy a random position on a core. As discussed above, when a core is predicted to fail, the subtask executing on the core predicted to fail gets seamlessly transferred to a core capable of processing at that instant.

The simulation studies are in accordance with the expectation and hence are a preliminary confirmation of the feasibility of the proposed approach in swarm-array computing. Though some assumptions and minor approximations are made, the approach is an opening for applying autonomic concepts to parallel computing platforms.

4. Implementation. In this section, a cluster-based implementation of the intelligent agent approach is considered. The cluster used for the research reported in this paper is one among the high performance computing resources available at the Centre for Advanced Computing and Emerging Technologies (ACET), University of Reading, United Kingdom [29] [30]. The cluster is primarily used for the purpose of teaching and performing multi-disciplinary research. The cluster consists of a head node and 33 compute nodes. All nodes are connected via a Gigabit ethernet switch and communicate via the standard TCP protocol.

The cluster-based implementations reported in this paper are based on the Message Passing Interface (MPI) [31], a standardized application programming interface (API) used for parallel and/or distributed computing. Open MPI [32] [33] version 1.3.3, an open source implementation of MPI 2.0 is employed on the cluster. An

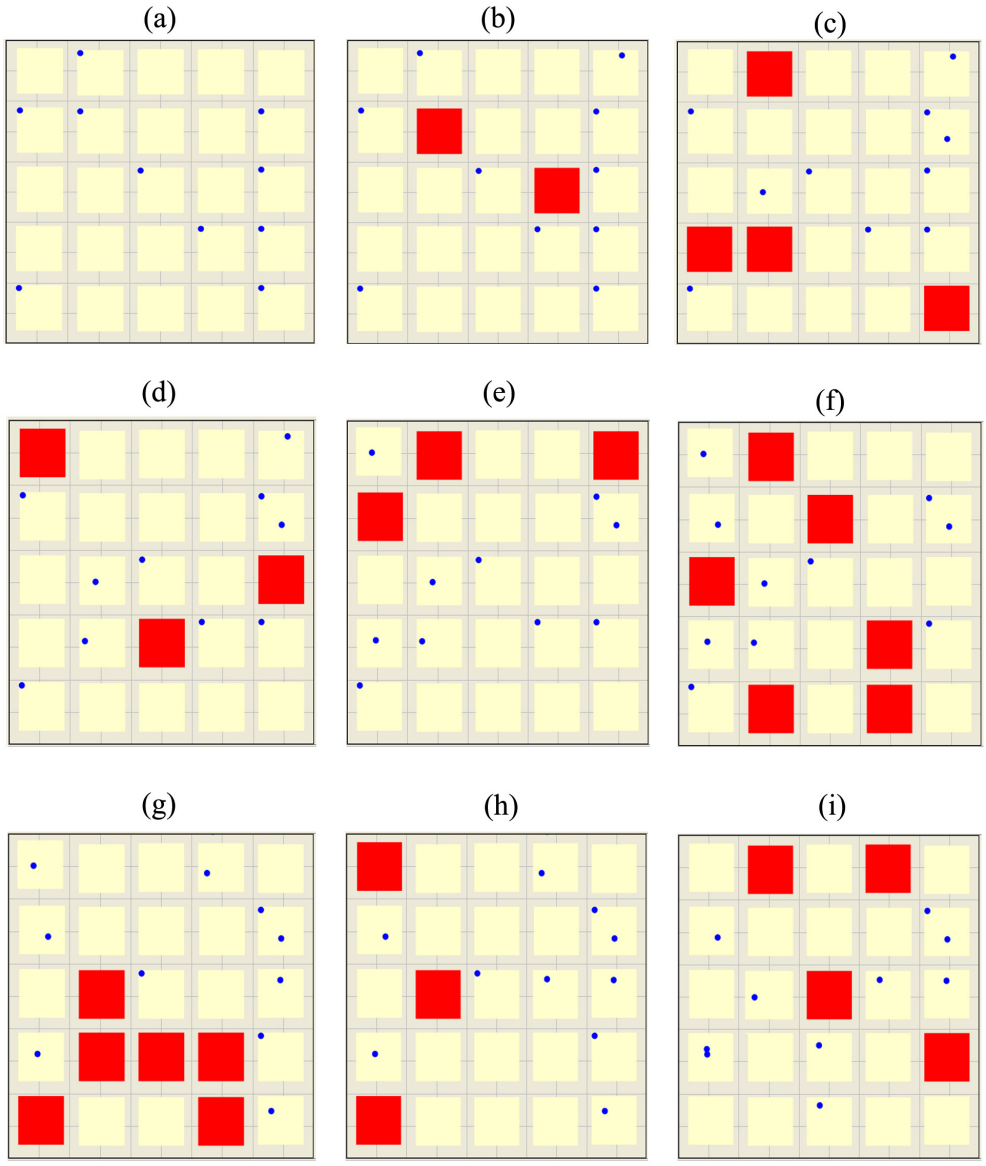


FIG. 3.1. *Sequence of nine simulation screenshots (a) - (i) of a simulation run from initialization on the SeSAm multi-agent simulator. Figure shows how the carrier agents carrying sub-tasks are seamlessly transferred to a new core when executing cores fail.*

important feature of MPI 2.0, dynamic process creation and management, is of potential for exploration in the context of swarm-array computing.

The MPI dynamic process model permits the creation and management of a set of processes both when an MPI application begins and after the application has started. The management of newly created processes include cooperative termination of a process, communication between newly created processes and existing MPI application, and establishing communication between two independent processes. MPI routines such as MPI_COMM_SPAWN is used to create a new MPI process and establish communication from an existing MPI application. On the other hand, MPI routines such as MPI MPI_COMM_ACCEPT and MPI_COMM_CONNECT can be used to establish communication between two independent processes. More MPI specific details on dynamic process model can be obtained from [31] [34].

To analyse the proposed intelligent agent based approach a parallel reduction algorithm is considered. Parallel reduction algorithms implement the bottom-up approach of binary trees [35], and are of interest in the

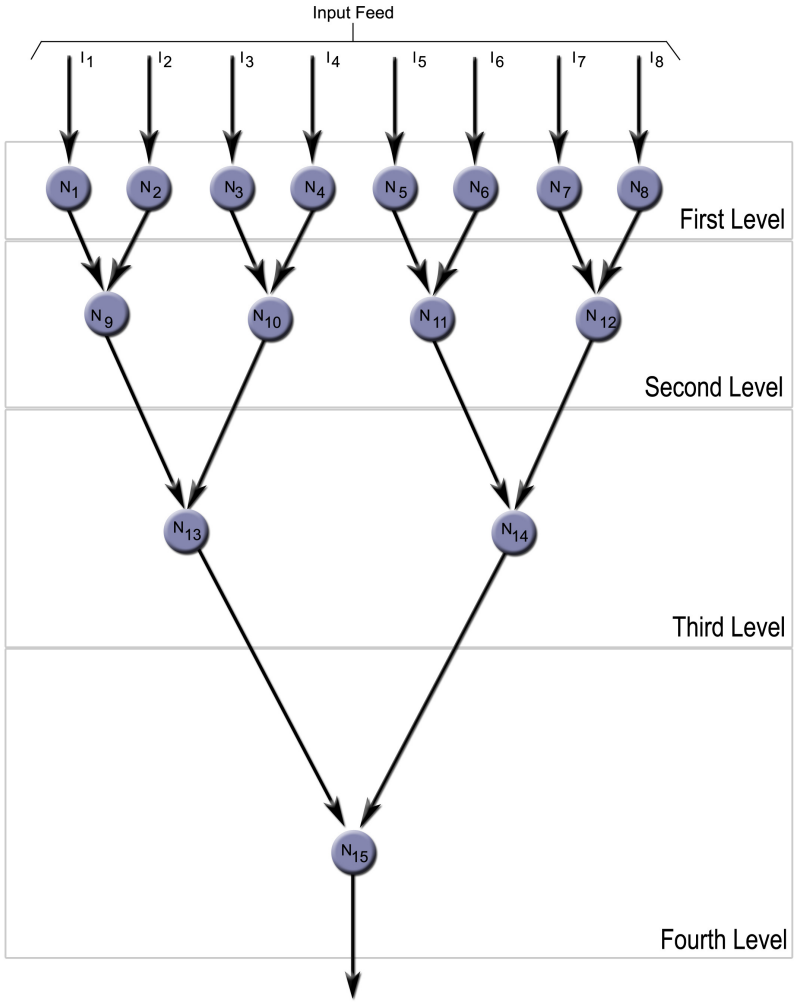


FIG. 4.1. *Illustration of the parallel summation algorithm.*

context of applying autonomic computing concepts to parallel computing systems due to two reasons. Firstly, the computing nodes of a parallel reduction algorithm tend to be critical. The execution of the algorithm stalls or produces an incorrect solution if any node information is lost. Secondly, parallel reduction algorithms are employed in critical applications such as space applications. Such applications require autonomic distributed systems.

Parallel summation is an exemplar of parallel reduction algorithm considered in this paper. If this class of algorithms do not incorporate fault tolerance concepts, then if a computing node fails due to an unpredictable event, the execution of the algorithm would stall.

The general concept of the algorithm is illustrated in figure 4.1. The algorithm works in four sequential levels. The first level comprising nodes $N_1 - N_8$ receives a live input feed of data $I_1 - I_8$. The second level comprising nodes $N_9 - N_{12}$ receives data from the first level, adds the data received and yields the result to the third level nodes N_{13} and N_{14}. The fourth level, adds data received from the third level nodes and produces the final result.

For a given time step, every node in a level operates in parallel. Each node is characterized by input dependencies (process or processor a node is dependent on for receiving an input), output dependencies (process or processor a node yields data to as output) and data contained in the node. The first level nodes have one input dependency and one output dependency. For instance, node N_1 has one input dependency I_1 and node N_9 as its output dependency. However, the second, third and fourth levels have two input dependencies and one

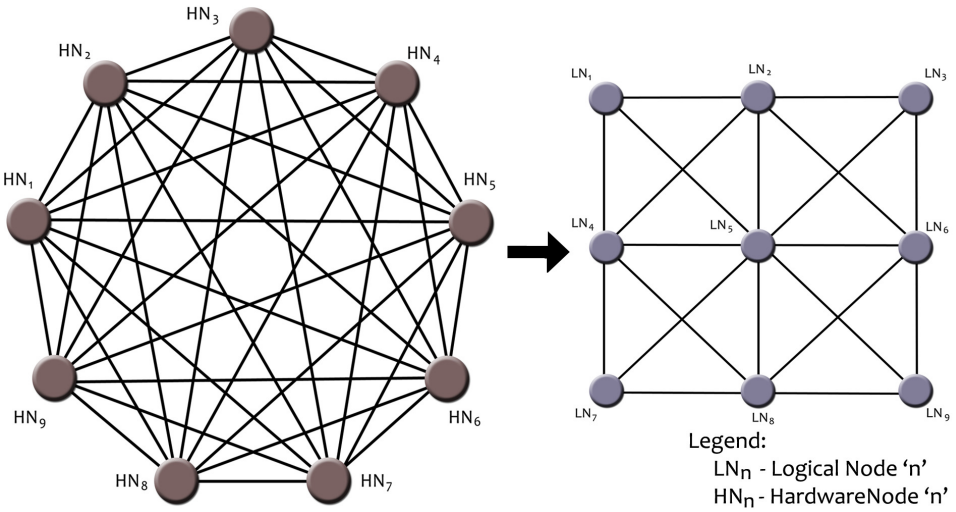


FIG. 4.2. *Mapping hardware nodes of the cluster to logical nodes in the abstracted layer.*

output dependency. For instance, node N_{13} of the third level has nodes N_9 and N_{10} as input dependencies and node N_{15} as output dependency. The data contained in a node is either the input data for the first level nodes or a calculated value (sum of two value in the case of a parallel summation algorithm) stored within a node.

A layer that abstracted the hardware resource layer, otherwise referred to as the abstracted layer had to be implemented. The hardware resource layer comprises physical nodes of the cluster and is connected via a switch, thereby forming a fully connected mesh topology. However, the abstracted layer is obtained when the physical nodes are abstracted as logical nodes. This is possible by implementing rules/policies. The policies are such that a process can only communicate with a vertically, horizontally or diagonally adjacent process, effectively leading to a grid topology on the abstracted layer. For example, nine nodes forming a fully connected mesh topology in figure 4.2 is abstracted to a grid topology in the abstraction layer.

The intelligent agents implemented in the parallel summation algorithm are with respect to the cognitive capabilities of agents considered in Section 2. The agents on the abstracted layer are created such that they carry input and output dependencies and data. Since, parallel summation is relatively less complex when compared to other computational algorithms, the agents carry little information and have only few dependencies.

Each process executing on a node also gathers some sensory information to predict whether a node is likely to fail. The sensory information enables an agent to know its own surroundings on the computational environment, hence demonstrating the first cognitive capability considered in Section 2.

In the implementation presented in this paper node temperatures are simulated. When the temperature of a node rises beyond a threshold, the process executing on that node predicts a failure and hence spawns a process on an adjacent core in the abstracted layer. In this case, an agent gathers sensory information on rising temperature than can likely impair or deteriorate its functioning, thereby demonstrating the third cognitive capability considered in Section 2. When rising temperature is detected, an agent has the potential to identify a node in the computational environment on which a new process can be spawned, thereby demonstrating the second cognitive capability considered in Section 2.

The agent on the abstracted core expected to fail shifts to the adjacent core on which the new process was spawned. An agent is capable of passing from one node to another, thereby demonstrating the fourth cognitive capability considered in Section 2.

The dependency information carried by the agent that was shifted to the new core is employed to reinstate the state of execution of the algorithm. The data for summation contained in the agent, either obtained from a previous level or a calculated value to be yielded to the next level, ensures that information is not lost and does not affect the final solution in critical applications.

Since the agents possess cognitive capabilities autonomic computing objectives such as self-configuration, self-healing and self-optimizing and autonomic computing attributes such as self-awareness, self-situated, self-monitoring and self-adjusting are inherently achieved. Hence, the approach implemented above incorporates

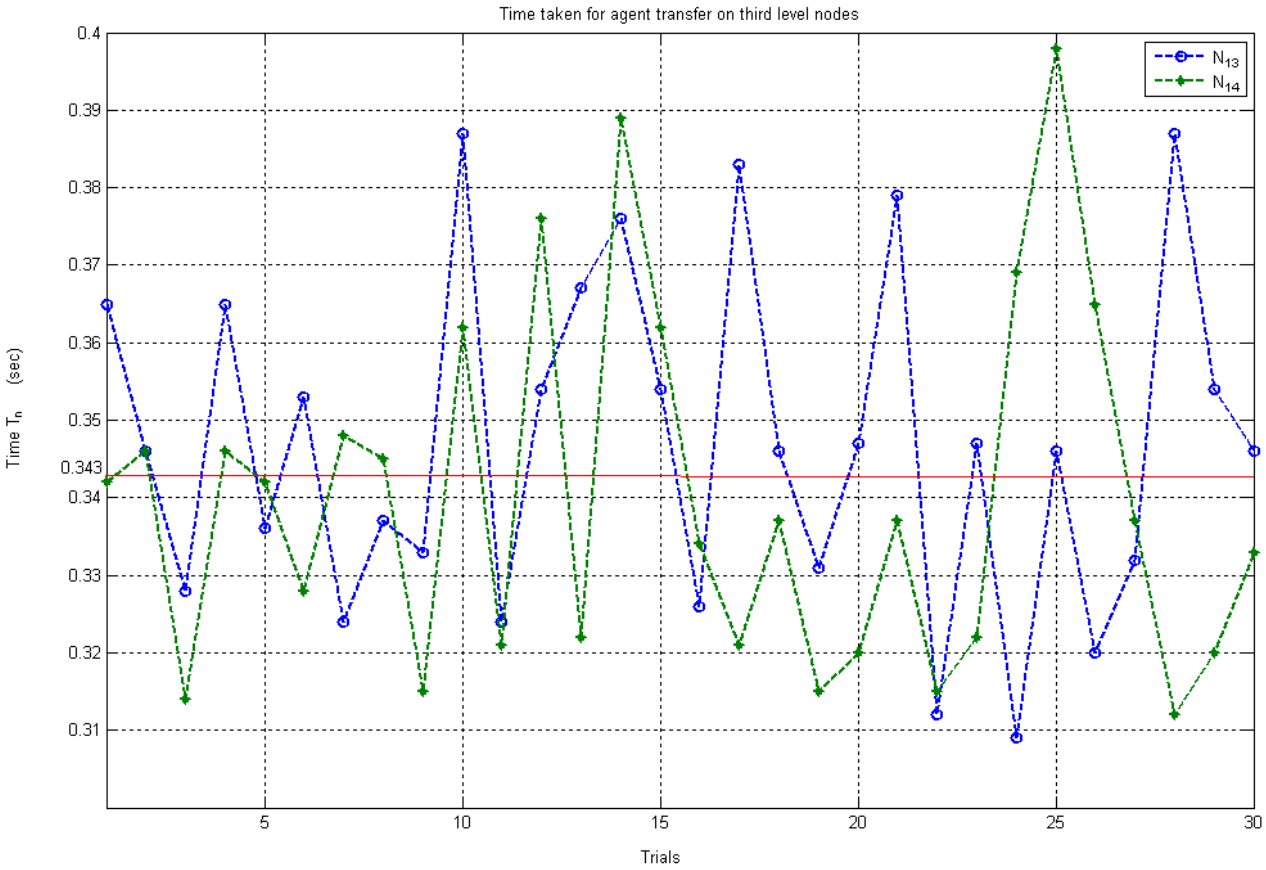


FIG. 5.1. *Time taken for an agent transfer from a computational node in the third level to an adjacent node. Mean time for agent transfer in third level nodes* $MT_{L_3} = 0.343$ *sec.*

concepts of the intelligent agent approach in swarm-array computing and is a preliminary step towards applying autonomic computing concepts to parallel computing systems.

The method proposed is an ample demonstration, though not highly sophisticated, that accommodates the concepts of intelligent agents leading towards achieving a few autonomic computing objectives and attributes.

5. Results. T_{N_n}, the time taken by an agent to transfer from a node N_n predicted to fail onto an adjacent node in the abstracted layer and re-establish all process dependencies for seamless execution was noted. Nodes N_{13} and N_{14} as shown in figure 4.1 are the third level computational nodes of the parallel summation algorithm, and hence are the only nodes considered for calculating T_{N_n} in this paper. Thirty different trial runs were performed to gather the statistic.

Figure 5.1 is the plot that shows T_{N_n} for the third level nodes N_{13} and N_{14} for 30 different trials.

Further, MT_{N_n}, the mean time of T_{N_n} for a particular node was calculated. This metric yields information on the mean time taken by an agent to transfer from a node N_n predicted to fail onto an adjacent node in the abstracted layer and re-establish all process dependencies for seamless execution. MT_{N_n} is calculated as follows:

$$MT_{N_n} = \left(\sum_{TR=1}^{30} T_n \right) \Big/ 30, n = 13, 14 \tag{5.1}$$

and TR being independent trials.

$MT_{L_p}, p = 3$, the mean time taken for an agent transfer from all nodes predicted to fail in the third level of the parallel summation algorithm onto an adjacent node in the abstracted layer is calculated as follows:

$$MT_{L_3} = \frac{1}{2} \sum_{n=13}^{14} MT_{N_n} \tag{5.2}$$

The mean time for an agent transfer from a computational node in the third level to an adjacent node in the abstracted layer is obtained as $MT_{L_3} = 0.343sec$, indicated by a red axis line in figure 5.1.

This statistic is the time taken for reinstating execution after a predicted third level node failure. If other approaches such as traditional checkpointing with human adminstration was employed, reinstating execution would be atleast in the order of minutes. This brief comparison reveals that the intelligent based approach is effective than traditional methods. Hence, applying autonomic computing concepts to parallel computing systems is beneficial.

In short, though preliminary results obtained through simple experiments are presented, the intelligent agent based approach of swarm-array computing proposed in this paper is promising and paves a path for bridging autonomic computing concepts and parallel computing systems.

6. Discussion & Conclusion. The impact that swarm-array computing can bring about can be foreseen by taking into account the industrial or business perspective and research perspective. From the industrial viewpoint, achieving autonomy in parallel computing systems is productive. The path towards autonomy can be equated to increasing reliability of geographically dispersed systems and hence reduction in total cost for maintenance. From the research perspective, achieving mobility of swarm agents in a heterogeneous parallel computing environment opens a new avenue to be explored. Moreover, swarm-array computing can be proposed as a new approach for closer examination and investigation.

From an application oriented point of view, swarm-array computing can be more assuring for applications that demand reliability. One potential application that can be influenced includes space applications. Space crafts employ FPGAs, a special purpose parallel computing system that are subject to malfunctioning or failures of hardware due to 'Single Event Upsets' (SEUs), caused by radiation on moving out of the protection of the atmosphere [36]–[38]. One solution to over-come this problem is to employ reconfigurable FPGAs. However, there are many overheads in using such technology and hardware reconfiguration is challenging in space environments. In other words, replacement or servicing of hardware is an extremely limited option in space environments. On the other hand software changes can be accomplished. In such cases, the swarm-array computing approach can provide solutions based on agent mobility and minimize overheads in software uploading and exclude requirement to reconfigure hardware.

In this paper, a swarm-array computing approach based on intelligent agents that act as carriers of decomposed tasks has been explored. Foundational concepts of a swarm-array computing approach namely intelligent agent based approach is considered. The feasibility of the proposed approach is validated on a multi-agent simulator. Experimental results obtained from a cluster based implementation of a parallel summation algorithm that implements concepts of intelligent agents is presented. Though only preliminary results are presented in this paper, the approach gives ground for expectation that autonomic computing concepts can be applied to parallel computing systems and hence open a new avenue of research in the scientific community.

Future work will aim to study the other swarm-array computing approaches considered in section 2. Efforts will also be made towards implementing the approaches in real time on other parallel computing systems using other existing middleware for parallel computing systems.

REFERENCES

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, *Amorphous computing*, Communications of the ACM, 43(5) (2000).

[2] S. R. Hedberg, *Evolutionary Computing: the spawning of a new generation*, IEEE Intelligent Systems and their Applications, Vol. 13, Issue 3 (2008), pp. 79–81.

[3] H. Schmeck, *Organic Computing - A New Vision for Distributed Embedded Systems*, in the Proceedings of the 8th IEEE Symposium on Object-Oriented Real-Time Distributed Computing, 2005, pp. 201–203.

[4] M. G. Hinchey and R. Sterritt, *99% (Biological) Inspiration*, in the Proceedings of the 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, 2007, pp. 187–195.

[5] P. Lin, A. MacArthur and J. Leaney, *Defining Autonomic Computing: A Software Engineering Perspective*, in the Proceedings of the Australian Software Engineering Conference, 2005, pp. 88–97.

[6] R. Sterritt and M. Hinchey, *Autonomic Computing - Panacea or Poppy-cock?*, in the Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005, pp. 535–539.
[7] R. Sterritt and D. Bustard, *Autonomic Computing - a Means of Achieving Dependability?*, in the Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003, pp. 247–251.
[8] M. R. Nami and M. Sharifi, *Autonomic Computing a New Approach*, in the Proceedings of the First Asia International Conference on Modelling and Simulation, 2007, pp. 352–357.
[9] M. Jarrett and R. Seviora, *Constructing an Autonomic Computing Infrastructure using Cougaar*, in the Proceedings of the 3rd IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, 2006, pp. 119–128.
[10] S. Lightstone, *Foundations of Autonomic Computing Development*, in the Proceedings of the 4th IEEE Workshop on Engineering of Autonomic and Autonomous Systems, 2007.
[11] T. Marshall and Y. S. Dai, *Reliability Improvement and Models in Autonomic Computing*, in the Proceedings of the 11th International Conference on Parallel and Distributed Systems, 2005, pp. 468–472.
[12] T. M. King, D. Babich, J. Alava, P. J. Clarke and R. Stevens, *Towards Self-Testing in Autonomic Computing Systems*, in the Proceedings of the 8th International Symposium on Autonomous Decentralized Systems, 2007, pp. 51–58.
[13] R. J. Anthony, *Emergence: a Paradigm for Robust and Scalable distributed applications*, in the Proceedings of the International Conference on Autonomic Computing, 2004, pp. 132–139.
[14] F. Saffre, J. Halloy, M. Shackleton and J. L. Deneubourg, *Self-Organized Service Orchestration Through Collective Differentiation*, IEEE Transactions on Systems, Man and Cybernetics, Part B (2006), pp. 1237–1246.
[15] A. Zeid and S. Gurguis, *Towards Autonomic Web Services*, in the Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.
[16] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci and M. Trubian, *Resource Management in the Autonomic Service-Oriented Architecture*, in the Proceedings of the IEEE International Conference on Autonomic Computing, 2006, pp. 84–92.
[17] M. Parashar, Z. Li, H. Liu, V. Matossian and C. Schmidt, *Enabling Autonomic Grid Applications: Requirements, Models and Infrastructure*, Lecture Notes in Computer Science, Self-Star Properties in Complex Information Systems, Springer Verlag. Vol. 3460, 2005, pp. 273–290.
[18] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser and D. Phung, *Self-Managing Systems: A Control Theory Foundation*, in the Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005, pp. 441–448.
[19] Q. Zhu, L. Lin, H. M. Kienle and H. A. Muller, *Characterizing Maintainability concerns in Autonomic Element Design*, in the Proceedings of the IEEE International Conference on Software Maintenance, 2008, pp. 197–206.
[20] J. O. Kephart and W. E. Walsh, *An Artificial Intelligence Perspective on Autonomic Computing Policies*, in the Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks, 2004, pp. 3–12.
[21] A. Peddemors, I. Niemegeers, H. Eertink and J. de Heer, *A System Perspective on Cognition for Autonomic Computing and Communication*, in the Proceedings of the 16th International Workshop on Database and Expert Systems Application, 2005, pp. 181–185.
[22] M. Hinchey, Y. S. Dai, C. A. Rouff, J. L. Rash and M. Qi, *Modeling for NASA Autonomous Nano-Technology Swarm Missions and Model-Driven Autonomic Computing*, in the Proceedings of the 21st International Conference on Advanced Information Net-working and Applications, 2007, pp. 250–257.
[23] L. M. F.-Carrasco, H. T.-Marin and M. V.-Rendon, *On the Path Towards Autonomic Com-puting: Combining Swarm Intelligence and Excitable Media Models*, in the Proceedings of the 7th Mexican International Conference on Artificial Intelligence, 2008, pp. 192–198.
[24] H. Guo, J. Gao, P. Zhu and F. Zhang, *A Self-Organized Model of Agent-Enabling Autonomic Computing for Grid Environment*, in the Proceedings of the 6th World Congress on Intelligent Control and Automation, 2006, pp. 2623–2627.
[25] J. Hu, J. Gao, B.-S. Liao and J.-J. Chen, *Multi-Agent System based Autonomic Computing Environment*, in the Proceedings of the International Conference on Machine Learning and Cybernetics, 2004, pp. 105–110.
[26] B. Varghese and G. T. McKee, *Towards Self-ware via Swarm-Array Computing*, in the Proceedings of the International Conference on Computational Intelligence and Cognitive Informatics, Paris, France, 2009, pp. 178–184.
[27] F. Klugl, R. Herrler and M. Fehler, *SeSAm: Implementation of Agent-Based Simulation Using Visual Programming*, in the Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems, Japan, 2006, pp. 1439–1440.
[28] SeSAm website: `http://www.simsesam.de`
[29] Center for Advanced Computing and Emerging Technologies (ACET) website: www.acet.reading.ac.uk
[30] High Performance Computing at ACET website: `http://hpc.acet.rdg.ac.uk/`
[31] W. Gropp, E. Lusk and A. Skjullum, *Using MPI-2: Advanced Features of the Message Passing Interface*, MIT Press (1999).
[32] OpenMPI website: `http://www.open-mpi.org/`
[33] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham and T. S. Woodall, *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, in the Proceedings of the 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.
[34] MPI Tutorial: `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`
[35] M. J. Quinn, *Parallel Computing Theory and Practice*, McGraw-Hill Inc. (1994).
[36] M. V. O'Bryan, C. Poivey, S. D. Kniffin, S. P. Buchner, R. L. Ladbury, T. R. Oldham, J. W. Howard Jr., K. A. LaBel, A. B. Sanders, M. Berg, C. J. Marshall, P. W. Marshall, H. S. Km, A. M. Dung-Phan, D. K. Hawkins, M. A. Carts, J. D. Forney, T. Irwin, .C. M. Seidleck, S. R. Cox, M. Friendlich, R. J. Flanigan, D. Petrick, W. Powell, J. Karsh and M. Baze, *Compendium of Single Event Effects Results for Candidate Spacecraft Electronics for NASA* in the Proceedings of the IEEE Radiation Effects Data Workshop, 2006, pp. 19–25.

[37] E. Johnson, M. J. Wirthlin and M. Caffrey, *Single-Event Upset Simulation on an FPGA*, in the Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, USA, 2002.
[38] S. Habinc, *Suitability of Reprogrammable FPGAs in Space Applications*. Report for the European Space Agency by Gaisler Research under ESA con-tract No. 15102/01/NL/FM(SC) CCN-3, 2002.

Edited by: Costin Bădică, Viorel Negru
Received: March 6, 2010
Accepted: March 31, 2010

Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 11, Number 1, pp. 63–71. http://www.scpe.org

GROUP-BASED INTERACTIONS FOR MULTIUSER APPLICATIONS

CARMEN MORGADO*, JOSÉ C. CUNHA*, NUNO CORREIA*, AND JORGE CUSTÓDIO*

Abstract. We propose a group-based model for the development of interactive applications where multiple users communicate and collaborate with each other and share information. The model defines the organization of a system in terms of entities and groups, each one with a well-defined interface, that mediates all the interactions. Entities can dynamically enter and leave groups, and distinct forms of communication among the group members are allowed, namely based on messages, events and a shared space. This model provides support for explicit groups and a novel concept of implicit groups, which allows the system to automatically adapt the group membership according to the dynamic changes in user profiles. We discuss applications of the model and illustrate its use to support example scenarios in local communities.

Key words: group-based model, interactive applications, shared spaces

1. Introduction. The recent increase in the development of applications exploring user mobility, interaction and information sharing, motivates the need for more expressive abstractions, models and mechanisms to enable a transparent and flexible specification of group-based interactions between users. The relevance of group models [3] has been recognized in multiple domains like collaborative work systems, multi-agents and more recently interactive Web applications for social networking. We describe a model that supports group abstractions to dynamically manage and organize small and medium location based communities of users. The MAGO model—Modeling Applications with a Group Oriented approach [11, 12], and its computing platform facilitates the organization, access and sharing of contents by multiple users, and enables their dynamic interactions. By considering groups as confined spaces for user interactions, the model allows to exploit geographic proximity of users, their associated locality, privacy and access control issues. We identify three main contributions of this work:

1. the identification of the main requirements of applications that led to the development of our model;
2. the possibility to dynamically adjust and control group membership, as well as to define policies to guide the implicit and dynamic group formation according to user profiles;
3. the support of a combination of forms of interactions that are required by real life applications, including peer-to-peer, multicast, asynchronous event notification, and access to a shared space, internal to each group.

In section 2 we identify requirements found in current emerging interactive applications. In section 3 we present the MAGO model, main concepts and functionalities. In section 4 the system architecture is described. In section 5 we describe application examples for small place-based communities. Finally, conclusions and current research directions are presented.

2. Motivation and requirements. We identified a diversity of scenarios where groups can be used to ease the modeling and development of applications where users form groups, to establish interactions and to share information. Examples arise in working places, travels, tourism spots, shopping centers or any other places where "social interactions" may exist. Places like schools campus, museums, shopping centers, restaurants and coffee shops, or airports, are examples where the users may tend to use groups as a way to interact, share information or to achieve a common goal. In most of these places people stay there for a limited period of time and are open to interact with each other. Another requirement is the need to structure the users space, so the services offered can be personalized based on the users preferences.

Similar requirements are also found in applications that manage user collaboration, connections and context awareness [10]. Examples of such applications are on the social software domain Facebook, MySpace or Hi5, and applications promoting the direct communication among users like ICQ, Skype, MSN Messager or Google Talk. On the information sharing domain there are general purpose applications like Flicker, Ringo or YouTube, or domain specific applications like canal*ACCESSIBLE [15] to help the navigation of people with locomotion disabilities using photos captured with mobile devices, or CONFECTIONARY [14], to create narratives based on user content, such as photos and videos. Also many applications exploit user mobility, allowing users to upload and access contents using handheld devices. Context awareness is a key issue in such applications [10]. In

*CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal, (cpm@fct.unl.pt, {jcc, nmc, jorge.custodio}@di.fct.unl.pt).

the tourism area we can find many examples of interactive applications, like Tech4Tourism [2] where the tourists can navigate through the maps gathering contextualized information (audio, video, photos or explanation texts) based on their actual position and personal interests; InStory [4] allows the users to have more interaction with the system and with each other, sharing information and grouping themselves based on different activities; or Marble [13] where the museum visitors can have access to information concerning the art objects nearby. Although all the above can benefit in some way from the use of group based concepts, none of them however, has the possibility of dynamic creation of groups, or offers an integrated platform for communications and data sharing.

Concerning the development of models that meet the needs for application structuring, information sharing and the establishment of interaction among the application entities, there is a diversity of proposals. Group models have been proposed for application structuring since the 1980's, in the context of distributed systems [16], and have been further explored at distinct levels of abstraction and with different goals, namely concerning CSCW/groupware system [17] and multi-agent systems.

Currently the group concept remains a significant approach to support the organization of an application in terms of collections of dynamic entities, each group defining a confined space for interactions among group members. From the above mentioned applications requirements, different forms of interactions should be supported, with distinct semantics concerning synchronous versus asynchronous behavior, including event-based, direct versus indirect interactions, and supported by shared or distributed memory.

However, in most of the proposals, groups are associated with forms of message-based communication among the group members. Such existing group systems already provide significant guarantees concerning, for example message delivering orderings, the management of group view consistency and the handling of failure situations. Such guarantees greatly contribute to ease distributed application development.

In order to provide other forms of interactions, other proposals have considered forms of asynchronous and indirect communication between entities based on shared spaces, easing the dynamic and loosely-coupled interactions among asynchronous entities. One significant proposal is the Linda model [9] that has been used in a large diversity of projects, to support coordination of distributed processes based on a shared tuple space.

In order to support some level of structuring, at the application level, some proposals have been made [18] allowing an application to be organized in terms of multiple tuple spaces. The shared tuple space concept remains a relevant approach to model information sharing functionalities, in two distinct dimensions: (i) to allow access to persistent information repositories, that maintain information on the users identification and profiles, and also access to contents; (ii) to allow anonymous asynchronous and indirect interactions among the application entities through the shared tuples, during execution.

In our proposal, we explore a distinct approach that relies on the group concept, which is interpreted as a confined interaction space, integrating in an unified way multiple forms of interactions, in order to provide improved flexibility and expressiveness to the applications. The model ensures the consistency of the group views, with respect to the group membership modifications, and the communication events (message multicast and access to the shared space).

A distinctive goal of our proposed model is to offer, in the same platform, the following functionalities:

- Dynamic management of users and their profiles;
- Flexible and dynamic creation of explicit and implicit groups;
- Dynamic group membership;
- Transparent access to personal and group information;
- Access to external information systems for manipulation of different kinds of media;
- User interactions and task coordination.

3. The MAGO model and characteristics. As a solution to the above identified problems we propose the MAGO model [11, 5] which has the following main characteristics:

- Users are represented as elementary entities and are modeled as Java objects;
- Groups are units for application composition with a well-defined interface. An Universal Group includes all entities and groups defined in the system. Besides, each entity can belong to multiple (explicit ou implicit) groups;
- Groups are dynamic, allowing multiple entities entering and leaving during a group life cycle;
- A group allows distinct forms of communication, namely, direct point-to-point, multicast (messages and asynchronous events), and shared memory;

This model relies on the transparent management of the consistency of views that are observed by the group members, concerning the events related to messages, events, and access to shared state, and event causality. The MAGO model offers, to the developers, a set of primitives and services specially designed to support group interactions within a dynamic environment. The primitives fall into three main classes, that we succinctly describe in the next subsections.

3.1. Entities and groups. This includes primitives for managing entities and groups and dynamic group membership. When creating a new group its defining characteristics are specified including: name; admission policy; maximum number of members admitted; type of group; activation and deactivation deadlines. In order to model situations of disconnected operations of entities that are group members, an entity can be put in an online/offline state. When online, the entity can interact with the members of the group; when offline, the entity although still belonging to the group, cannot be accessed.

3.2. Communications. To reflect the different kinds of possible interactions the model combines three forms of communication within a group space:

- Direct: point-to-point communication, based on a direct call to an interface method of an entity or group.
- Events: based on a publish-subscribe mechanism [7], and is used to multicast messages inside a group space. Group members can subscribe to any event type that has been advertised on the group, and receive a notification on event publication. An event can be associated to messages or to changes on internal structures.
- Shared space: a form of communication that supports a shared space associated to each group. This mechanism is based on the concept of a Linda [8] tuple space with corresponding primitives to insert (*update*), remove (*get*) and consult (*consult*). Additionally we have defined a primitive (*find*) that allows the search of multiple tuples that match the search parameters specified. Each tuple has a name, owner information, the type and application specific information. The model uses the event-based mechanism to allow the notification of group members, about changes on the shared space.

3.3. Implicit groups. MAGO also provides a higher level group concept that enables the definition and automatic management of groups based on user properties and attributes (*implicit groups*). The implicit group membership can vary over time and is based on the users currently on the system, their characteristics, and the rules previously defined by the application at group creation time. Such rules are simple conjunctions of user attributes and their satisfaction is evaluated by a search engine, always looking for candidate members that match the specified rules.

The action of joining an implicit group occurs in three distinct situations:

- when a new implicit group is created and its characteristics are defined;
- when a new entity enters the system, and specifies its attributes;
- when an entity changes its attributes (this last action can also lead to an implicit action of leaving an implicit group).

To support the above model functionalities we developed a system architecture, that is described next.

4. System architecture. A layered architecture supports the proposed model as shown in figure 4.1. The MAGO architecture relies on a low-level Java API—JGroupSpace [6], which is an extension to JGroups [1], and offers primitives for group management, message and event multicast, and an implementation of the Linda tuple space. The MAGO model offers a higher level semantics for interactive applications concerning messages, event publication and subscription, access to the shared tuple space, and also provides the concept of implicit groups and an interface to an information system.

An external information system can store the contents produced and shared by the users, and also information related to user and group attributes, which is used to manage implicit groups. The information system offers the possibility of data persistence related to the system organization and contents information. Our system offers an interface that supports access to the basic functionalities of the information system (see figure 4.2). The current prototype supports access to an ORACLE Database system.

The model was implemented on an object oriented platform, where the applications are responsible for the instantiation of elementary entities (objs) and then have access to the MAGO functionalities/primitives (figure 4.3).

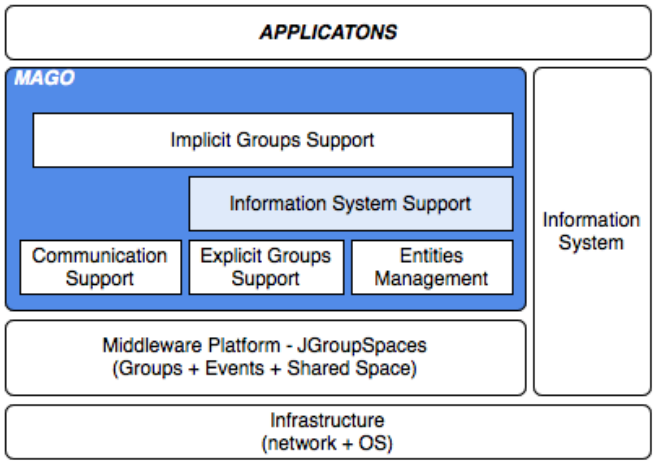


FIG. 4.1. *System layers and modules—MAGO components are included in the gray area in the figure. The bottom layer offers basic communication and system operation support, responsible for connections and management between devices. Middleware layer is responsible for group communications, events and shared space. Information system gives access and manages persistent data manipulated by the system.*

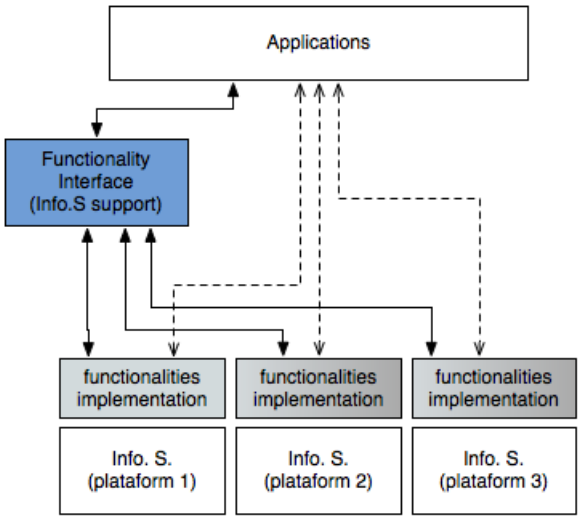


FIG. 4.2. *Interaction between applications our interface and the information system.*

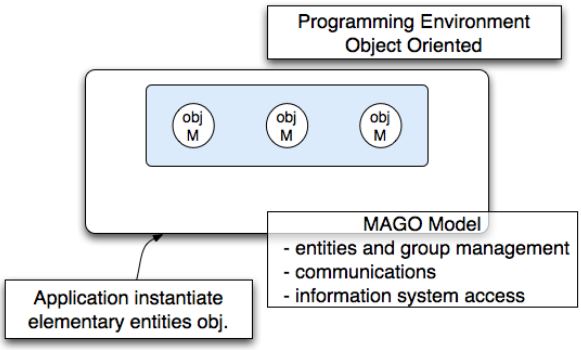


FIG. 4.3. *Interaction between applications our interface and the information system.*

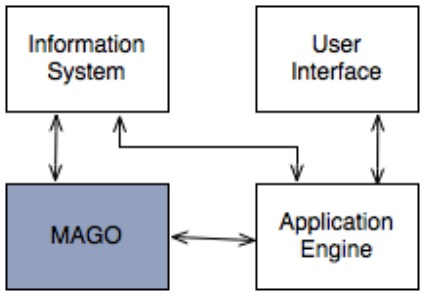


FIG. 5.1. *Applications basic modules.*

A working prototype of the MAGO architecture and an user interface giving access to the model primitives were implemented in Java, and used to support the experimentation with real applications (a detailed description can be found in [11]).

5. Application examples. The MAGO model promotes on organization where an application is based on four building blocks (figure 5.1): user interface; application engine; information system; and the MAGO module.

This organization is independent of the functionalities offered. Basically applications need to manage:

- the registration of users within the system;
- the users and groups profiles;
- the creation, remotion and group membership;
- different kinds of communication within a group space;
- the production and access to shared data;
- the coordination of tasks within a group space.

In order to assess the adequacy of the model functionalities and its usability, we experimented with already existing real world applications. With this experimentation we illustrate how our system can be used to model and extend functionalities in existing applications. It is also illustrated how the system is easily integrated into already developed architectures.

There is a diversity of scenarios where grouping of entities can be used to ease the modeling and development of applications. As mentioned before, examples of such application scenarios arise in work places, travels, tourism spots, shopping centers or any places where we have "social interactions".

Within these environments users usually want to form groups (in an explicit or implicit mode), to establish interactions and to share information. Places like a tourism spot or a school campus are examples of places where the users may tend to use groups as a way to interact, share information to achieve a common goal, to play a game or simply as a way to meet people that share similar preferences.

5.1. Tourism scenario. We describe how MAGO supports a tourism scenario in a local community where groups are used to facilitate user activities and interactions in a cultural heritage setting. This work extends a real world application previously developed in the InStory project [4]. Tourism applications usually deal with interaction between multiple users, information dissemination and content creation and sharing. In this scenario many users have mobile devices, for example PDAs, cell phones or small laptops and are likely to interact with others to establish groups, and share information and content like photos or videos. Based on the existing InStory architecture we integrated our model as a way to simplify the support for existing functionalities and to develop new ones, where the users (visitors and managers) were modeled as elementary entities (figure 5.2).

In the following several examples of groups are given for this application.

Visitor groups. These groups are used to distribute information and share photos (or other content) taken during a tour. Assuming that all the visitors have already registered into the system, by invoking the *register* primitive, a typical sequence of actions is:

- a visitor guide creates a new explicit group with name "tour14" using the primitive create (`g1=create(..., "tour14", EXPLICIT, ...)`);
- a visitor (`u1`) can join the previously created group (g1) using the primitive `join(u1,g1, ...)`);

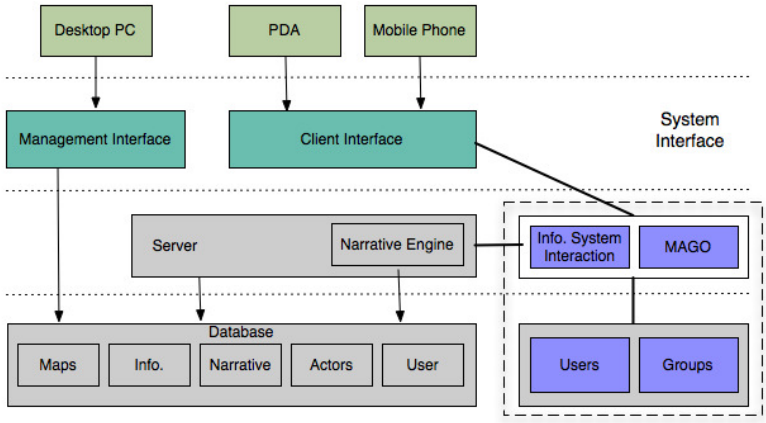


FIG. 5.2. *InStory architecture (an already existing system architecture) adapted to incorporate the MAGO model.*

- the visitor can access group functionalities:
  - sharing a photo and notify all group members: first update a file to the information system, then create a data object (`data_obj`) with information specified by the application (for example with the reference to the data file on the information system and a location reference), and finally update data on the "tour14" group shared space notifying all the group members (`update(u1,g1,, NOTIFY,data_obj)`);
  - other members can now access the photo, based on notification information: first create a data object with the notification information, then consult (without blocking) the data on "tour14" group shared space (`consult(u1,g1,,, NO_BLOCK, data_obj)`) and extract the information system reference to the photo, from the data object, finally get the file from the information system and visualize it on the local device;
  - disseminate information to all group members: spread a message to the group, by invoking the send primitive (`send(u1, g1, SPREAD, "tour start in 5m")`).

  In this example, the data files such as the ones containing photos and videos are stored in the information system maintained by the application.

Implicit groups to personalize the information. While the above functionalities could rely on the explicit specification and management of the mentioned groups, that could become a quite cumbersome task due to the highly dynamic nature of the applications. Our defined concept of implicit groups incorporated in the MAGO model is very useful to manage the dynamic changes in user profiles and system configuration, as illustrated in the following scenarios.

First a system manager (acting as the tour guide) creates an implicit group defining the value attributes that all the group members must match.

- define the attributes list (`lst=[nationality,"italian"]`);
- create the new group (`create(id_usr,"itVisit",IMPLICIT,-,-,lst)`);
- update the group information on the information system.

After having created the group, all the users that match the specified attributes are automatically joined to the group. Now the new group can be used, for example, to send personalized messages. If a new visitor has attributes, that match the implicit group attributes, he/she automatically becomes a member of the group.

Implicit groups and location dependent contents. As in the previous example a system manager creates an implicit group, this time with attributes `lst=[local, "Templar Well"]`. Having created this group, each time a visitor arrives at the specified location, the location attribute changes to "Templar Well", and the visitor becomes a member of the group, allowing access to the group resources. Having arrived at that location, the visitors can interact with other group members, using the group communication mechanisms offered by the model, and can also update and access the contents of the group shared space.

When a visitor leaves that location (by changing the location attribute) he/she will be automatically removed from the group "Templar Well", by the system.

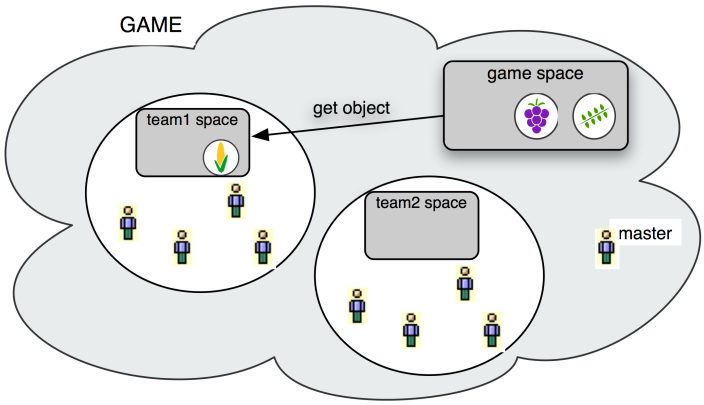


FIG. 5.3. *Mobile game scenario*

Use in a mobile game scenario. We defined a mobile game scenario where groups of users compete with each other and try to be the fastest on collecting virtual artifacts that were previously defined and distributed on the gardens and palace by the game manager (figure 5.3). The team that collects more artifacts wins the game.

For modeling this application we defined several groups: one for each team and one for the game space. The artifacts are characterized by a name, a position and other information related to the game functionality. At the beginning of the game they are on the shared space of the game group.

When arriving at a specified position a player asks the game group if an artifact is available, by invoking a get primitive (`get(player1, game, ...)`). If this operation succeeds (artifact was on the game group shared space), this means that the artifact gets removed from the game space. Then the player must put the object on the player's team space and notify the other team members that a new object is available (`update(player1,team1,..,NOTIFY,ObjArt)`). When executing this get operation, if no artifact is collected this means that another player has already collected that object.

In a similar way the players can collect clues instead of artifacts that help progressing in the game. During the game the team players can also communicate directly with each other or with the teams (*send()*), or leave clues for their team members (`update(player1,team1,PUBLIC,NO_NOTIFY,ObjClue)`).

The game manager can also communicate at any time with all the players and teams, using the same primitives. For instance, for sending a message to all the game players announcing the end of the game, a send primitive will be used directed to the group game (`send(master,game,SPREAD,"Game Over")`).

5.2. School campus. With this case study, like in the previous one we start from an already existing system and incorporate MAGO as a way to extend some of the application functionalities (figure 5.4). In this case study, the users (professors and students) were modeled as elementary entities.

The establishment and generation of groups is usually based on the users professional activities and/or personal characteristics, interests and hobbies, such as (5.5):

- group of teachers or computer science teacher (CS teachers),
- group of computer science student (Joe Group),
- players of the football team (FTeam),
- groups associated with classes or courses (Programming (P1) Class, Math Class),
- group of users of room 123.

Or groups associated with the users preferences and/or profiles, like for instance:

- fans of action movies,
- students looking for a room/apartment to rent/share,
- students looking for a part-time job,
- people that have cars and are going to town,
- people that need a ride to town.

These groups are used for spreading information (for example: about a change on a due date for an assignment), for scheduling meetings (ex: agreement on a meeting date) and share media data (ex: put a video

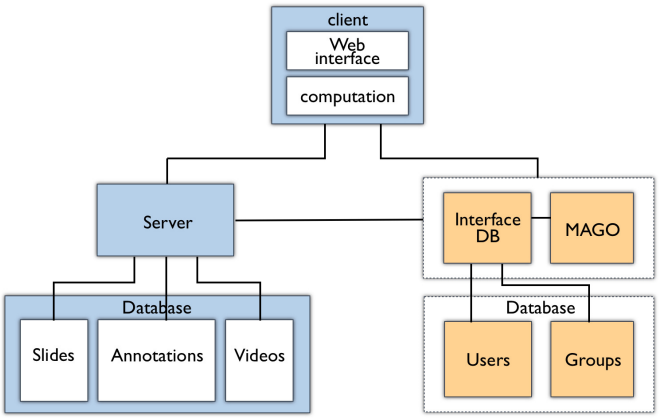


FIG. 5.4. *VideoStore architecture (an already existing system architecture) incorporated with the MAGO model.*

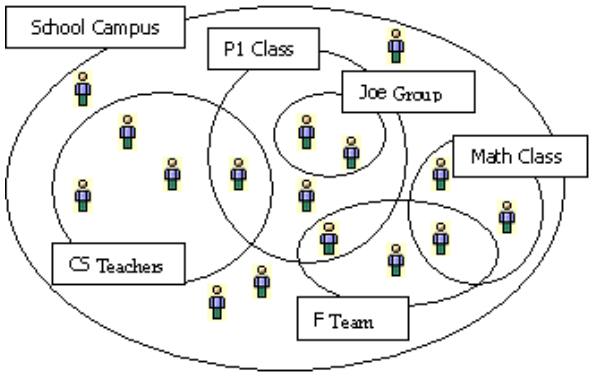


FIG. 5.5. *Possible groups in a school campus scenario*

and/or slides of a class that will be available to the group of students of that specific class). Next we illustrate some of the possibilities of the use of explicit and implicit groups in this campus scenario.

Explicit groups. For example to form and manage the explicit group for the user Joe (Joe Group), a typical sequence of actions that will be performed is as follows:

- Create the group "JoeG": this action is launched by user Joe and is made by invoking the create primitive: `joegID=create(joeID,"JoeG",EXPLICIT,by_creator,5)`. This will create a group limited to 5 members and where new members admission is granted only by the group creator Joe;
- Join the group "JoeG": this action is performed by group candidates by invoking a join primitive (`(join(maryID,joegID)`. After this action Mary will become a member of the group "JoeG", if authorization to join was granted;
- After a group creation, the members can use it to share and access group data, to disseminate information to all group members, or simply to communicate directly with each other:
  - share data, using the update primitive: `update(maryID,joegID, ,NOTIFY,data_object`. This instruction will announce to all group members that new data was updated. The data can be accessed by all group members with a consult primitive (`consult(joeID, joegID, , ,NO_BLOCK, data_object)`. The `data_object` can be of any type, for instance a text, a video or a photo.
  - send message to the group, using the send primitive with the `spread` option (`send(joeID, JoeG, SPREAD, "Hello all")`: this will disseminate the message to all group members.

Implicit groups. If for instance a teacher wants to create a group of all the school members that live in Lisbon, he/she should create an implicit group, where the matching rule that must be verified by all the members will be: `"home=Lisbon"`. In this case the create primitive will have to specify the attributes list `list=[home,"Lisbon"]`. So the create primitive will have the form: `homeID=create(t1,"LisbonG",IMPLICIT, , ,list)`

As already mentioned in the tourism case study, the implicit groups are also useful to create groups associated to location, like for instance the group of people that currently are near "Building1" (`list=[local, "Building1"]`). With this group created, each time a system user arrives at the specified location, his/hers location attributes change and the user will automatically became a member of that group. This group can be used in order to give information about an event that is taking place at that location. The group can also be used for the users to communicate and share data with each other.

6. Conclusions and ongoing work. Emerging interactive applications increasingly exhibit requirements for collaboration and information sharing in multiuser dynamic environments. In order to model those applications, we are exploiting group based abstractions, according to the MAGO approach. The model supports a framework for group management, allowing the structuring of an application in terms of a dynamic collection of entities. Furthermore, groups provide confined interactions spaces, offering different forms of communication (messages, events and shared space), which were found suitable to model the typical application interaction patterns. In order to ease the modeling process, MAGO supports a new concept of implicit groups, allowing an automated management of groups, based on dynamic identification of user profiles.

As part of ongoing work MAGO is being used to design and implement other applications with more complex services, namely in the tourism and cultural heritage domains, and in learning environments.

Acknowledgments. The authors thank the FCT/MCTES for the support given to the development of this work.

REFERENCES

[1] Bela Ban, *JavaGroups—Group Communication Patterns in Java*, Technical Report,Cornell University, 1998.
[2] Luca Chittaro, Demis Corvaglia, Luca De Marco, Silvia Gabrielli, and Augusto Senerchia, *Tech4Tourism*, `http://hcilab.uniud.it/t4t/index_en.html`, 2006.
[3] Gregory V. Chockler, Idid Keidar, and Roman Vitenberg, *Group communication specifications: a comprehensive study*, ACM Comput. Surv., 33 (2001), pp 427–469, ACM Press.
[4] Nuno Correia, Luis Alves, Helder Correia, Luis Romero, Carmen Morgado, Luís Soares, José C. Cunha, Teresa Romão, A. Eduardo Dias,and Joaquim A. Jorge, *InStory: a system for mobile information access, storytelling and gaming activities in physical spaces*, ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp 102–109, ACM Press.
[5] José C. Cunha, Carmen P. Morgado, and Jorge F. Custódio, *Group Abstractions for Organizing Dynamic Distributed Systems*, Euro-Par 2008 Workshops—Parallel Processing, pp 450–459, Springer-Verlag
[6] Jorge Custódio, *JGroupSpace- Support for group oriented distributed programming (in Portuguese)*, Master Thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2008.
[7] Patrick Th. Eugster and Pascal A. Felber and Rachid Guerraoui and Anne-Marie Kermarrec, *The many faces of publish/subscribe*, ACM Comput. Surv., 35 (2003), pp.114–131.
[8] David Gelernter, *Generative communication in Linda*, ACM Trans. Program. Lang. Syst., 7 (1985), pp. 80–112, New York, NY, USA.
[9] Nicholas Carriero and David Gelernter, *Linda in context*, Commun. ACM, 32 (1989), pp. 444–458, New York, NY, USA.
[10] R. Kernchen, D. Bonnefoy, A. Battestini, B. Mrohs, M. Wagner, and M. Klemettinen, *Context-Awareness in MobiLife*, Proceedings of the 15th IST Mobile and Wireless Communication Summit, 2006.
[11] Carmen Morgado, *A group-based model for interactive distributed applications (in Portuguese)*, Ph.D. thesis, Department of Computer Science, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, 2009.
[12] Carmen Morgado, Nuno Correia, and J. C. Cunha, *A group-based approach for modeling interactive mobile applications (poster)*, International ACM Conference on Supporting Group Work—Group07, Sanibel, USA, 2007.
[13] Carmen Santoro, Fabio Paterno, Giulia Ricci, and Barbara Leporini, *A Multimodal Mobile Museum Guide for All*, Mobile Interaction with the Real World (MIRW 2007)—Workshop @ MobileHCI, 2007.
[14] MediaLab, *CONFECTIONARY*, `http://mf.media.mit.edu/confectionary`, Accessed July 2009.
[15] canal*accessible, *zexe.net*, `http://www.megafone.net/BARCELONA`, Accessed August 2009.
[16] Kenneth P. Birman, *The process group approach to reliable distributed computing*, Communication ACM, 36 (1993), pp 37–53.
[17] David C. Yen, H. Joseph Wen, Binshan Lin, and David C. Chou, *Groupware: a strategic analysis and implementation*, Industrial Management and Data Systems, 99 (1999), pp 64–70.
[18] P. Wyckoff and S. W. McLaughry and T. J. Lehman and D. A. Ford, *TSpaces*, IBM Systems Journal, 37, April, 1998.

Edited by: Costin Bădică, Viorel Negru
Received: March 6, 2010
Accepted: March 31, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 73–80. http://www.scpe.org
ISSN 1895-1767

PARALLEL QUASIRANDOM APPLICATIONS ON HETEROGENEOUS GRID

A. KARAIVANOVA, E. ATANASSOV, T. GUROV, S. IVANOVSKA, M. DURCHOVA*

Abstract. In this paper we present error and performance analysis of quasi-Monte Carlo algorithms for solving multidimensional integrals (up to 100 dimensions) on the grid using MPI. We take into account the fact that the Grid is a potentially heterogeneous computing environment, where the user does not know the specifics of the target architecture. Therefore parallel algorithms should be able to adapt to this heterogeneity, providing automated load-balancing. Monte Carlo algorithms can be tailored to such environments, provided parallel pseudorandom number generators are available. The use of quasi-Monte Carlo algorithms poses more difficulties. In both cases the efficient implementation of the algorithms depends on the functionality of the corresponding packages for generating pseudorandom or quasirandom numbers. We propose efficient parallel implementation of the Sobol sequence for a grid environment and we demonstrate numerical experiments on a heterogeneous grid. To achieve high parallel efficiency we use a newly developed special grid service called *Job Track Service* which provides efficient management of available computing resources through reservations.

1. Introduction. Quasi-Monte Carlo methods and algorithms are proved to be efficient in many areas ranging from physics to economy. They are based on quasirandom (also called low discrepancy) sequences while Monte Carlo methods are based on pseudorandom numbers [3]. As a rule the computational schemes of the quasi-Monte Carlo methods are simple and convenient for use in applied computing. There has been renewed interest in quasi-Monte Carlo in recent times, the primary reason for this is that quasi-Monte Carlo methods usually give much better rate of convergence than Monte Carlo ones, and, in the same time, preserve the good parallel properties of Monte Carlo applications.

Monte Carlo and quasi-Monte Carlo are often used for solving computationally intensive problems. Studying their performance on different parallel and distributed computer systems is an important practical problem. For example, problems with dimension 100 and more are typical for financial mathematics. For such problems it is natural to use parallel and Grid systems. Nowadays, from the other hand, the computational Grid proved to be a very efficient computing model [12]. The Grid aims ultimately to turn the global network of computers into one vast computational resource [4]. Grid computing in general is a special type of parallel computing. Technically Grid coordinates resources which are not a subject to central administrative control and utilizes general-purpose protocols. Another distinction is that a Grid could in principle have access to parallel computers, clusters, farms, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalized form of distributed computing.

One major advantage of Monte Carlo methods is that they are usually very easy to be parallelized. This is, in principal, also true of quasi-Monte Carlo methods. However, the successful parallel implementation of a quasi-Monte Carlo application depends crucially on various quality aspects of the parallel quasirandom sequences used [5, 6]. Much of the recent work on parallelizing quasi-Monte Carlo methods has been aimed at splitting a quasirandom sequence into many subsequences which are then used independently on various parallel processes, for example in [1, 7]. This method works well for the parallelization of pseudorandom numbers, but due to the nature of quality in quasirandom numbers, this technique has some difficulties. Our algorithms are based on scrambling which is suitable for heterogeneous computing environments.

In the present paper we propose and study parallel Sobol sequence application in heterogeneous grid environment. Sobol sequence is one of the most popular quasirandom sequences. We first present the algorithm for fast generation of scrambled Sobol sequence, parallelization techniques, and some numerical results for solving multidimensional integrals which show the quality of our generation. In the next section we present grid performance results for a quasi-Monte Carlo method intended for heavy computations (dimension 100). Our tests proved the applicability of the presented scrambled Sobol sequence in grid environment. Moreover, the MPI code is executed simultaneously on two different grid clusters (one with Mirynet interconnect, the other with conventional interconnect) and high parallel efficiency has been achieved. For the purpose of efficient implementation of MPI applications a special grid service called *Job Track Service* (JTS) has been developed and its initial version has been tested here.

*IPP-BAS, Acad. G. Bonchev St., bl. 25A, Sofia 1113, Bulgaria, E-mails: anet, emanouil, gurov, sofia, mabs@parallel.bas.bg

2. Monte Carlo and Quasi-Monte Carlo Integration. Consider an integral on the unit cube $I^s = [0,1]^s$ in s dimensions,

$$I[f] = \int_{I^s} f(x)dx, \tag{2.1}$$

of a Lebesgue integrable function $f(x)$, and note that this integral can be expressed as the expectation of the function f, $I[f] = E[f(x)]$, where x is a uniformly distributed vector in the unit cube.

Consider the following approximation of the integral (2.1):

$$I_N[f] = \frac{1}{N}\sum_{n=1}^{N} f(x_n). \tag{2.2}$$

If $\{x_n\}$ is a sequence sampled from uniform distribution, equation (2.2) is called (crude) Monte Carlo quadrature formula. The integration error, defined as

$$\epsilon_N[f] = |I[f] - I_N[f]|, \tag{2.3}$$

has a standard normal distribution, with expectation

$$E[\epsilon_N[f]] = \sqrt{Var(f)}N^{-1/2}. \tag{2.4}$$

An exact upper bound for (2.2) is given by Koksma-Hlawka inequality,

$$\epsilon_N[f] \leq V[f]D_N^*, \tag{2.5}$$

in which V[f] is the variation of f in the Hardy-Krause sense and D_N^* is the star discrepancy of the sequence $\{x_n\}$, [3]. Equation (2.5) is valid for any function with bounded variation and any choice of sequence [3], however the best results are obtained with low discrepancy (also called quasirandom) sequences for which

$$D_N^* \leq c(\log N)^k N^{-1},$$

where c and k are constants independent of N but dependent on s.

In this paper we use scrambled Sobol sequence. The purpose of using scrambled sequences in quasi-Monte Carlo methods is twofold. Primarily, it provides a practical method to obtain error estimates for QMC based on treating each scrambled sequence as a different and independent random sample from a family of randomly scrambled quasirandom numbers [1, 11]. Thus, randomized QMC overcomes the main disadvantage of QMC while maintaining the favorable convergence rate of QMC. Secondarily, scrambling gives us a simple and unified way to generate quasirandom numbers for parallel, distributed, and Grid-based computational environments [4, 5, 6].

The test functions for numerical integration in this paper are:

$$F_1 = \prod_{i=1}^{s}\left({x_i}^3 + \frac{3}{4}\right),$$

$$F_2 = \prod_{i=1}^{s}|4x_i - 2|.$$

The first of test function is described in [8]. The second is known as Roos and Arnold's example and it is suggested as test function in [9].

3. Sobol sequences. Sobol sequence proved to be one of the most efficient sequences for quasi-Monte Carlo integration, [2, 3, 10]. The original sequence is not suitable for grid implementation, [5, 6]. Here we present a fast algorithm for generation of a scrambled Sobol sequence suitable for parallel and grid implementations. We use the Definition below, which covers most digital (t, m, s)-nets in base 2. The Sobol sequence is a (t, s)-sequence in base 2 and is a particular case of this definition.

Definition. *Let $A_1, \ldots, A_s$ be infinite matrices $A_k = \{a_{ij}^{(k)}\}$, $i, j = 0, 1, \ldots$, with $a_{ij}^{(k)} \in \{0, 1\}$, such that $a_{ii}^{(k)} = 1$ for all i and k, $a_{ij}^{(k)} = 0$ if $i < j$. The $\tau^{(1)}, \ldots, \tau^{(s)}$ are sequences of permutations of the set $\{0, 1\}$. Each non-negative integer n may be represented in the binary number system as*

$$n = \sum_{j-0}^{r} b_i 2^i.$$

Then the n-th term of the low-discrepancy sequence σ is defined by

$$x_n^{(k)} = \sum_{j=0}^{r} 2^{-j-1} \tau_j^{(k)} (\oplus_{i=0}^{j} b_i a_{ij}^{(k)}),$$

where by "$\oplus$" we denote the operation "bit-wise addition modulo 2".

Next lemma explains how we generate consecutive terms of the sequence.

Lemma. *Let σ be a sequence or net satisfying the above Definition, and let the non-negative integers n, p, m be given. Suppose that we desire the first p binary digits of elements in σ with indices of the form $2^m j + n < 2^p$. This implicitly defines a set of compatible $j's$. Thus the only numbers we need to compute are*

$$y_j^{(k)} = \lfloor 2^p x_{2^m j+n}^{(k)} \rfloor.$$

The integers $\{v_r^{(k)}\}_{r=0}^{\infty}$, which we call "twisted direction numbers", are defined by

$$v_r^{(k)} = \sum_{t=0}^{p-1} 2^{p-1-t} \oplus_{j=m}^{p-1} a_{tj}^{(k)}.$$

Suppose that the largest power-of-two that divides $2^m(j+1) + n$ is l, i. e. $2^m(j+1) + n = 2^i(2K+1)$. Then the following equality holds

$$y_{j+1}^{(k)} = y_j^{(k)} \oplus v_i^{(k)}.$$

The main algorithm for generating scrambled Sobol sequence is described in [1]. Here we study the quality and the applicability of the generated sequence in grid environment. The algorithm allows consecutive terms of the scrambled sequence to be obtained with essentially only two operations per coordinate: one floating point addition and one bit-wise xor operation (this omits operations that are needed only once per tuple). This scrambling is achieved at no additional computational cost over that of unscrambled generation as it is accomplished totally in the initialization. In addition, the terms of the sequence are obtained in their normal order, without the usual permutation introduced by Gray code ordering used to minimize the cost of computing the next Sobol element. This algorithm is relatively simple and very suitable for parallel and grid implementation.

The mathematical explanations can be found in [1] and [2], here we present the algorithm in pseudo code.

- Input initial data:
  - if the precision is single, set the number of bits b to 32, and the maximal power of two p to 23, otherwise set b to 64 and p to 52;
  - dimension s;
  - direction vectors $\{a_{ij}\}, i = 0, p, j = 1, \ldots, s$ representing the matrices $A_1, \ldots, A_d$ (always $a_{ij} < 2^{i+1}$);
  - scrambling terms $d_1, \ldots, d_s$ - arbitrary integers less than 2^p, if all of them are equal to zero, then no scrambling is used;
  - index of the first term to be generated - n;
  - scaling factor m, so the program should generate elements with indices $2^m j + n$, $j = 0, 1, \ldots$.
- Allocate memory for $s \star l$ b-bit integers (or floating point numbers in the respective precision) $y_1, \ldots, y_s$.
- Preprocessing: calculate the twisted direction numbers v_{ij}, $i = 0, \ldots, p-1$, $j = 0, \ldots, s$:
  - for all j from 1 to s do
  - for $i = 0$ to $p - 1$ do

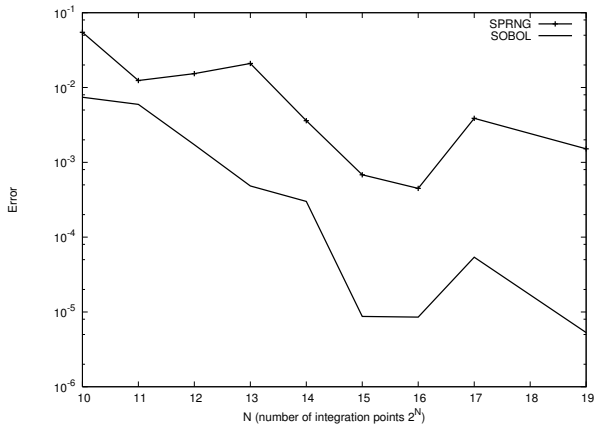


FIG. 3.1. *Accuracy of numerical integration of* **Test function** F_1 *with sample size* N *and dimension 10 using pseudorandom numbers and Sobol sequence.*

  - if $i = 0$, then $v_{ij} = a_{ij}2^{p-m}$,
    else $v_{ij} = v_{i-1j}\mathbf{xor}(a_{i+m,j} \star (2^{p-i-m}))$;
- Calculate the coordinates of the n^{th} term of the Sobol sequence (with the scrambling applied) using any known algorithm (this operation is performed only once). Add 1 to all of them and store the results as floating point numbers in the respective precision in the array y.
- Set the counter N to the integer part of $\dfrac{n}{2^m}$.
- Generate the next point of the sequence:
  - When a new point is required, the user supplies a buffer x with enough space to hold the result.
  - The array y is considered as holding floating point numbers in the respective precision, and the result of subtracting 1 from all of them is placed in the array x.
  - Add 1 to the counter N.
  - Determine the first nonzero binary digit k of N so that $N = (2M + 1)2^k$ (on the average this is achieved in 2 iterations).
  - Consider the array y as an array of b-bit integers and updated it by using the k^{th} row of twisted direction numbers:
    for $i = 1$ to d do $y_i = y_i\mathbf{xor}v_{ki}$.
  - Return the control to the user. When a new point is needed, go to beginning of this paragraph (Generate the next point of the sequence).

Numerical results for approximate calculation of 10 and 20 dimensional integrals with Monte Carlo and quasi-Monte Carlo using scrambled Sobol sequence are shown on Figure 3.1 and Figure 3.2 (test function F_1) and Table 3.1 (test function F_2). Let us mention that usually dimension 20 is considered as too high for quasi-Monte Carlo treatment. Here we see that the quasi-Monte Carlo algorithm with scrambled Sobol sequence (Sobol in the Table3.1) gives better results than Monte Carlo (SPRNG available on [14] in the Table 3.1). The good properties of the sequence motivated us for using this generator in our grid applications.

Parallelization.

There are three basic ways to parallelize quasirandom number sequences:

- *Leap-frog*—the sequence is partitioned in turn among the processors like a deck of cards dealt to card players.
- *Sequence splitting or blocking*—the sequence is partitioned by splitting it into non-overlapping contiguous subsections.
- *Independent sequences*—each processor has its own independent sequence.

The first and second schemes produce numbers from a single quasirandom sequence. The third scheme needs a family of quasirandom sequences. Scrambling techniques can generate such a stochastic family of quasirandom sequences from one original quasirandom sequence. Numerical calculations presented in this paper are done using scrambling and blocking. In this way we can exactly repeat the results for comparisons.

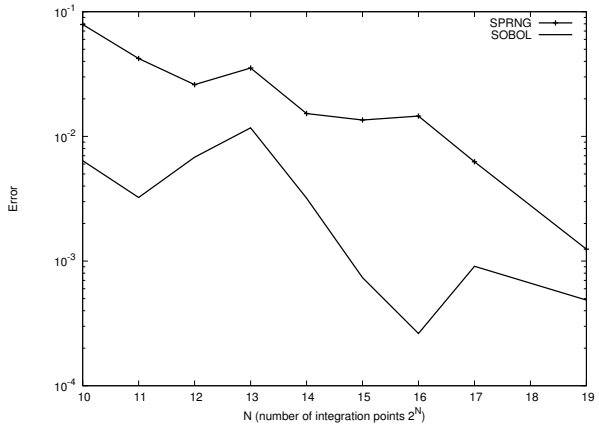


FIG. 3.2. *Accuracy of numerical integration of* **Test function** F_1 *with sample size N and dimension 20 using pseudorandom numbers and Sobol sequence.*

TABLE 3.1
Accuracy of numerical integration of **Test function** F_2 *with sample size N when using pseudorandom numbers and Sobol sequence.*

N	dim	SPRNG	Sobol
2^{10}	10	7.033e-02	1.220e-02
	20	3.164e-01	2.114e-01
2^{11}	10	2.622e-02	2.504e-02
	20	2.851e-02	1.331e-02
2^{12}	10	1.106e-01	2.225e-02
	20	9.003e-02	4.579e-02
2^{13}	10	4.254e-02	3.722e-02
	20	1.454e-01	1.403e-01
2^{14}	10	4.777e-02	1.667e-02
	20	1.064e-02	8.385e-02
2^{15}	10	3.482e-02	1.456e-02
	20	3.751e-02	1.514e-01
2^{16}	10	2.622e-02	2.919e-03
	20	1.528e-02	3.104e-02
2^{17}	10	2.098e-02	2.059e-03
	20	2.966e-02	8.379e-03
2^{19}	10	2.024e-03	2.475e-05
	20	1.190e-02	1.716e-02

4. Quasi-Monte Carlo Computations on the Grid. We take into account the fact that the Grid is a potentially heterogeneous computing environment, where the user does not know the specifics of the target architecture. Therefore parallel algorithms should be able to adapt to this heterogeneity, providing automated load-balancing. Monte Carlo algorithms can be tailored to such environments, provided parallel pseudorandom number generators are available.
The use of quasi-Monte Carlo algorithms poses more difficulties. In both cases the efficient implementation of the algorithms depends on the functionality of the corresponding packages for generating pseudorandom or quasirandom numbers.

4.1. Grid-enabled Parallel Implementation of the Algorithm for Numerical Integration. Our tests are implemented on the two main Grid clusters in IPP-BAS which are included in the Pan-European (EGEE) and regional (SEE-GRID) grid infrastructures and which have completely different characteristics.

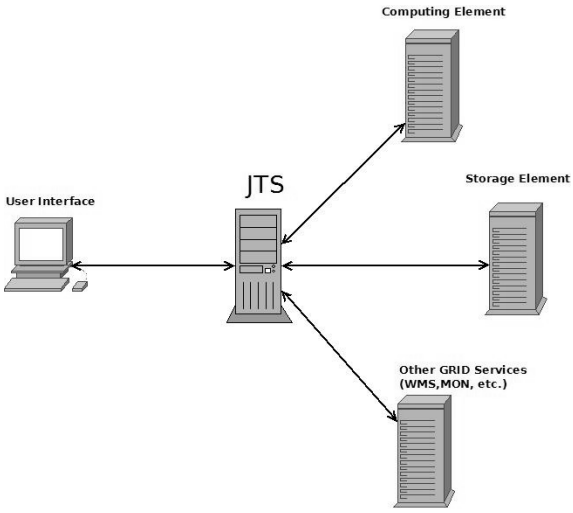


FIG. 4.1. *JTS working scheme.*

EGEE (www.eu-egee.org) grid infrastructure serves a wide range of European scientific communities and is currently the largest, most widely used multidisciplinary grid infrastructure in the world [12]. The EGEE grid uses gLite (www.glite.org), an open source middleware distribution that combines components developed in various related projectsin particular Condor and Globusvia the Virtual Data Toolkit.

The parameters of the grid clusters used in our experiments are: The BG03-NGCC grid cluster has 25 worker nodes, 2 x Intel Xeon E5430 2.66 GHz Quad Core CPU (total 200 Cores, $>$ 400 kSI2000). Each node has 16 GB RAM. The BG04-ACAD has 40 worker nodes with 2 CPU Opteron 2,4 GHz (total 80 cores $>$ 120 KSI2000), 4GB RAM for each node, and low-latency Myrinet interconnection for MPI jobs as in [13]. Here we present grid-enabled parallel implementation of the standard Monte Carlo and quasi-Monte Carlo methods for numerical integration. In our grid tests we use the function F_1 with dimension $s = 100$.

The parallelization is based on the master-slave paradigm, with some ramifications. We divide the work into chunks, corresponding to the sub-domains, which are requested from the master process by the slave processes. In order to increase the efficiency, the master also performs computations, while waiting for communication requests. Thus we achieve overlap of computations and communications, and we do not lose the possible output of the master process. When using low-discrepancy sequences, we take care in both master and slave processes to fast-forward the generator exactly to the point that is needed. The scrambling that is provided by the generators enables a posteriori estimation of the error in the quasi-Monte Carlo case.

4.2. Job Track Service. In order to run MPI application simultaneously on different grid clusters we use newly developed service called Job Track Service (JTS) which allows CPU reservation for parallel jobs, and collecting and analysis of the results of these parallel jobs. JTS (see Fig. 4.1) implements messaging system to allow Quality of Service (QoS) for jobs, gathering and analysis of performance data and provision of application specific info from central point.

Some details about JTS:

- The core of the JTS is a JTS server. In this initial version there is only one instance of this server, located at IPP.
- The JTS server accepts connections via AMQP (Advanced Message Queuing Protocol, www.amqp.org), XMPP (Extensible Messaging and Presence Protocol, www.xmpp.org), and SOAP.
- The grid clients are running on the Grid Computing Elements. They connect with the JTS server using AMQP and send/receive messages, related to jobs.
- The users can send additional performance information or can manipulate jobs in a desired way. For instance, they can tag jobs as higher priority and they can direct jobs to VO reservations, available to their application.

The JTS provides efficient management of available computing resources through reservations. Systems of this kind can interoperate and probably will become the standard 2-3 years from now. The potential for

TABLE 4.1
Parallel efficiency measurements for $N = 10^9$ points - T_1 is the theoretically optimal time and T_2 is the measured time.

	T_1	T_2	Efficiency
SPRNG	117.10	129.87	90.17%
Sobol	14.37832	15.34	93%

TABLE 4.2
Time for **test function** F_1 *with* 10^8 *points.*

	1 CPU (BG03)	1 CPU (BG04)
SPRNG	785.64	533.90
Sobol	65.79	95.65

standards-based (AMQP, STOMP, HTTP, XMPP) integration of information from different services and easy presentation to end user/site admin/VO admin can enhance the utilization and experience while using the Grid infrastructure. This trend is visible in the Cloud computing world (see, e.g., ElasticFox, S3Fox).

In our computations we use the JTS in order to ensure simultaneous startup of the MPI jobs on the two chosen clusters.

4.3. Numerical Experiments. Our numerical tests are performed for 100-dimensional integrals which is usual dimension for many applications in financial mathematics. The presented results are from parallel implementation of our algorithm on *20 CPU cores on BG04-ACAD* and *32 CPU cores on BG03-NGCC* simultaneously. The start of these MPI jobs on both clusters has been achieved by calling the Job Track Service on these two clusters.

In Table 4.1. we compare the estimated time for running our algorithm on the Worker Nodes of the grid sites BG03-NGCC and BG04-ACAD in parallel, in case of perfect parallel efficiency, with the measured execution time. Here T_1 is theoretically optimal time, T_2 is measured time and the efficiency is the ratio $\frac{T_1}{T_2}$. We used the MPICH-G2 implementation of MPI, which is the most general approach for running parallel jobs on computational grids.

We obtained roughly the same accuracy with low-discrepancy sequences as with pseudorandom numbers, due to the high effective dimension. The CPU time in Table 4.2 of our implementations of the quasi-Monte Carlo versions was significantly smaller, due to the efficient generation algorithms. Quasi-Monte Carlo algorithms are more difficult to parallelize due to the nature of the sequences, but this can be done if the generators of the low-discrepancy sequences have the necessary functionality.

5. Conclusion and future work. The purpose of this paper is to present and study quality of a fast generator of scrambled Sobol sequence, and its grid implementation. The comparative study of Monte Carlo and quasi-Monte Carlo method using Sobol sequence for computing very high-dimensional integrals (s=100) show the advantages of our generator even in heterogeneous computing environment. In any case high dimensional problems are good candidates for execution in the Grid environment, although the use of quasi-Monte Carlo or true random number generators poses certain technical challenges, related to the parallelization of the computations or the repeatability of results.

As a future work we plane to use other scrambling methods for Sobol sequence and its usage in large problems.

6. Acknowledgment. This work has been supported by the National Science Fund of Bulgaria through grants DO02-146 and DO02-115.

REFERENCES

[1] E. ATANASSOV: A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence, *Numerical Methods and Applications, Springer*, vol. 2542 ,p. 83, 2003.

[2] S. IVANOVSKA, E. ATANASSOV, A. KARAIVANOVA: A Superconvergent Monte Carlo Method for Multiple Integrals on the Grid, *LNCS, Springer-Verlag, Berlin-Heidelberg,* vol. 3516, p. 735, 2005.

[3] R. Caflisch: Monte Carlo and quasi-Monte Carlo methods, *Acta Numerica,* vol. 7, p. 1, 1998.
[4] J. Foster, C. Kesselmann: *The Grid: Blueprint for a New Computing Infrastructure,* Morgan Kaufmann, 1998.
[5] H. Chi and E. Jones: Generating Parallel Quasirandom Sequences by using Randomization, *Journal of distributed and parallel computing,* vol. 67(7), p. 876, 2007.
[6] H. Chi and M. Mascagni: Efficient Generation of Parallel Quasirandom Sequences via Scrambling, *Lecture Notes in Computer Science, Springer,* vol. 4487,p. 723, 2007.
[7] B. C. Bromley: Quasirandom Number Generation for Parallel Monte Carlo Algorithms, *Journal of Parallel Distributed Computing,* vol. 38(1), p. 101, 1996.
[8] W. Schmid, A. Uhl: Techniques for parallel quasi-Monte Carlo integration with digital sequences and associated problems, *Math. and Comp. in Sim.,* vol. 55, p. 249, 2001.
[9] A. Owen: The Dimension Distribution and Quadrature Test functions, *Statistica Sinicai,* vol. 13, p. 1, 2003.
[10] S. Chaudhary: *Acceleration of Monte Carlo Methods using Low Discrepancy Sequences,* Dissertation, University of California, Los Angeles, 2004.
[11] G. Okten and B. Tuffin and V. Burago: A central limit theorem and improved error bounds for a hybrid-Monte Carlo sequence with applications in computational Finance, *Journal of Complexity,* vol. 22(4), p. 453, 2006.
[12] I. Bird, B. Jones, K. Kee: The organization and management of grid infrastructures, *IEEE Computer*, vol.9, p. 36, 2009.
[13] MPI website, `http://www-unix.mcs.anl.gov/mpi/`
[14] SPRNG: Scalable Parallel Random Number Generator, `http://sprng.cs.fsu.edu/`

Edited by: Dana Petcu
Received: January 12, 2010
Accepted: January 30, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 1, pp. 81–97. http://www.scpe.org
ISSN 1895-1767

OSYRIS: A NATURE INSPIRED WORKFLOW ENGINE FOR SERVICE ORIENTED ENVIRONMENTS*

MARC EDUARD FRÎNCU† AND DANA PETCU†

Abstract. We present a nature-inspired rule-based workflow platform capable of self adaptation by following an event-condition-action approach and by integrating a dynamic resource selection relying on rule based scheduling heuristics. A language for workflows that is simple and easy to understand is introduced. The language also supports task semantics and ontology definition.

Key words: nature inspired workflow, workflow language, automatic workflow generation, scheduling mechanism, workflow management

1. Introduction. Nature has proven to be a good source of inspiration for computer software paradigms within work spreading from membrane computing [35], cellular-automata [37] and genetic algorithms [9] to neural networks [5]. However little attention, most of which is purely theoretical with no concrete validation of the model [32, 33], has been given to creating a nature inspired workflow language and engine. Nature provides excellent examples of orchestrating evolution rules and we need not go further than the chemical reaction laws to find such ones. In a chemical reaction, solutions are usually transformed into others by using substitution, synthesis, chemical decomposition, or other methods [25]. Yet the reactions only occur when there exist the right reactants which can be in turn the result of other reactions. When binding together these reactions what we obtain is a chain reaction. Chain reactions are the counterparts of workflows in computer science. An example of such a chain reaction is the photochemical reaction between hydrogen and chlorine where the chlorine atoms react with hydrogen until other reactions consume the remaining free atoms of chlorine or hydrogen.

From a computer perspective the advantages of such an approach include implicit parallelism and non-deterministic reaction order which allow: faster rule execution; decentralized approach as the reactions can occur without the supervision of a central entity; less workflow constructs (such as the parallel construct); simpler syntax (only the reaction rule is required); possibility of expressing self-adaptivity to failures as rules etc. Together with a Distributed Environment (DE) where each task would be executed on a different machine this approach could provide the basis for a self-adaptive and dynamic workflow platform in which the engine behaves as a natural entity.

This paper describes a workflow platform, called OSyRIS (Orchestration System using a Rule based Inference Solution) and a concrete usage example. Sections 2 and 3 will present a brief state of art on workflow engines and respectively a formalisation based on a chemical model. The OSyRIS design (detailed in Section 4.1) is founded on the concepts described in [32, 33] which provided the idea of creating workflows based on chemical reactions. An implementation called SiLK (Simple Language for worKflows) of the chemical formalism presented in the previous citations is also presented in Section 4.2. The platform is adapted to cope with DE such as Grids by introducing support for Web Services (WS). It offers self-adaptivity to resource changes (Sections 4.4 and 4.5) and allows automatic workflow generation (Section 4.3) based on a given goal, some task semantics and an initial rule base. Additional helper modules such as a visual workflow designer (Section 4.6) and administrator (Section 4.7) are also presented. The platform is validated against a real case scenario (Section 5) which requires the orchestration of multiple image processing operations for manipulating large satellite images.

2. Related Work and Issues Regarding Workflows. A lot of work has been accomplished towards achieving self-adapting WS orchestration and many elaborated solutions exist in the form of advanced languages [4, 23, 40, 43], complex workflow engines [1, 16, 31, 39, 41], graphical tools [42] or workflow task planners [16]. Few authors have recently concentrated thier efforts [3, 7, 13, 27] towards the dynamic enactment of scientific workflows including automatic error handling; the runtime resource allocation and task planning; or the adaptivity to changes in workflow logic and system characteristics.

*This work has been partially supported by the European Space Agency PECS Contract no. 98061 GiSHEO: On Demand Grid Services for High Education and Training in Earth Observation.
†West University Of Timisoara, Blvd. Vasile Parvan No 4, 300223, Romania ({mfrincu, petcu}@info.uvt.ro)

A lot of recent work focuses on workflow solutions aiming at improving classic solutions like Condor [39], YAWL [40], Pegasus [16], Taverna [34], Triana [15] or ActiveBPEL [1]. A new trend which relies on Event-Condition-Action (ECA) driven solutions has emerged in relationship with AgentWork [29], VIDRE [31] FARAO [41] or Drools Flow [6].

ECA based engines allow an alternative declarative approach by introducing [41]: languages having *intuitive formal semantics* through the use of a limited set of primitives, *direct support for business and science policies*, *flexibility* by allowing self-adaptation through the use of rules which permit alternative execution paths, *adaptability* through the insertion and/or retraction of rules and *reusability* thanks to their property of being isolated from the process context. Although non-ECA approaches can also be modified to cope with these advantages, ECA solutions are preferred due to their concept of rule based programming. Non ECA engines usually have limited predefined and built-in control constructs such as sequence, parallel, split, join, loops or events. In contrast, ECA engines have one single built-in control construct, namely the inference rule, all the other constructs naturally deriving from it. ECA approaches also offer other advantages like separation of logic (rules) from data (objects), declarative programming, scalability or centralization of knowledge, etc.

All the existing workflow engines deal with the same problems including changes in data integrity issues, error handling, and resource availability changes.

Data integrity issues can be solved either by ensuring proper service selection or by allowing user intervention. Workflow engines like ActiveBPEL, Condor or gEclipse do not have integrated modules to cope with the latter. These engines require a redeployment of the workflow in order to allow user intervention. Still work on this topic exists and our paper [11] presents a solution in which ActiveBPEL workflow tasks are enhanced with pause/resume/cancel operations. Triana is an example which allows to stop/resume/reset workflows and change input parameters to tasks from the interface.

Proper service selection can also be achieved in several ways: by using Hoare semantics [26] in order to automatically check if a workflow can produce the desired outcome by its actual implementation and to synthesize a workflow implementation; by using ontologies [4, 14] to describe tasks and their relationships as well as the compositional rules; by using an AI planner [44]; or, given a desired goal and some task semantics, by using a backward chaining mechanism for selecting the proper task chaining as it will be detailed in this paper.

Error handling is of importance too. Workflow errors can be caused either by logic failures (e.g. wrong workflow design) or physical failures (e.g. network or resource failures). As far as the latter are concerned Taverna halts the execution of the entire workflow in case a service invocation fails [17]; ActiveBPEL provides a try-catch mechanism for handling invocation exceptions; Triana also has an in-built mechanism for preventing the system crash in case of workflow errors; YAWL uses ripple down rules [3] in order to cope with execution errors; FARAO relies on a monitor-plan-act cycle based on the Adaptive Service Oriented Architecture [24] which allows semi-autonomously service adaptation; and Pegasus handles errors either by rescheduling a crashed task or by creating a rescue DAG. Paper [26] presents a solution in which engine failures are dealt with by spliting the workflow into several smaller ones, part of the initial rule base.

Some workflow engines such as YAWL [3], Taverna [34], Triana [15] or ActiveBPEL [13] allow runtime service selection. This is highly important in case of resource failures. However in the case of the previously listed examples, neither this step is separated from the core of the engine nor this functionality is easily achieved [27]. Unlike these workflow engines, the Pegasus system [16] does allow the integration of custom selection or scheduling policies, which makes task execution much more efficient. We argue that scheduling policies are essential since they can cope with resource, network or task failures and can augment the workflow execution time by periodical task reassignments. These reassignments are necessary because of the possible delays in task execution inflicted by local schedulers running on the resource where the services are located on. The same problem arises in the ECA driven engines which deal mostly with task and data dependencies and tend to ignore the planning aspect [6, 29].

Section 3 will show that nature inspired workflow paradigms have the advantage of offering a non-deterministic and inherently concurrent environment where no serialization is imposed.

3. Nature Inspired Workflows. It is only recently that attention has been given to nature inspired workflows although among their advantages we can enumerate: elimination of explicit parallel execution; easy implementation of decentralized engines; large dynamism with regard to runtime workflow or resource changes; the possibility of parallel executing parameter sweeps on the same workflow; clear separation of workflow logic from data; simple syntax with only one explicit construct; non-deterministic task execution, etc.

Among the few formalisations of a nature inspired workflow enactment we notice the work of Banâtre [7] and Németh [32, 33]. Their work relies on a chemical metaphor based on the Gamma (General Abstract Model for Multiset Manipulation) [8] language. This concept follows the chemical model in which there is no concept of centralization, serialization, ordering and the computations are accomplished in a non-deterministic way. γ-calculus is a formal definition of the Gamma paradigm and the fundamental structure is represented by the multiset M. Terms (or molecules) are represented by: variables x, γ abstractions (reactive molecules) $\gamma\langle x\rangle\lfloor C\rfloor.M$ (where C is the condition required for reducing the abstraction), multisets M_i and solutions $\langle M\rangle$.

The reduction of the γ-abstraction is performed by the reactions inside a solution $\langle M\rangle$. The reduction simply means capturing molecules that match given parameters (variables) and replace them with instantiated values inside the specified solution. More specifically $\gamma\langle x\rangle\lfloor C\rfloor.M$ means replacing every occurrence of x in solution $\langle x\rangle$ by M if C.

Using γ-calculus, a task execution can be modeled as:

$$\gamma\langle T_j : x, \omega_T\rangle.(\gamma\langle id : r, Res, \omega_{Res}\rangle.execute\, T_i\ using\, x\ on\, r) \tag{3.1}$$

Based on the previously described Gamma formalism we introduce the following workflow notation:

$$\mathcal{W} = (V, \mu, M_1, \ldots, M_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), s_0) \tag{3.2}$$

where: V is the set of tasks (or molecules), μ represents the solution structure containing all the other n solutions, $M_i, i = \overline{1,n}$, stand for the multisets over V belonging to the n solutions, R_i symbolizes the reactive molecules (or simply the reactions), $\rho_i, i = \overline{1,n}$ represent priorities among the reactions and s_0 is the outermost solution.

Given the previous workflow notation (Relation 3.2) we easily notice that the definition closely resembles that of the P-System [46]. The sole major difference is that of using chemical solutions instead of cellular membranes.

Although Relation 3.1 offers a representation for task execution within chemical solutions it is difficult to implement an easy to use computer language based on it. In addition we can further refine the representation by modeling the concept of conditional execution through a function similar to the one presented in Relation 3.3:

$$\epsilon : M_i^* \xrightarrow{C,\rho_i} M_i \tag{3.3}$$

where $M_i^* = M_i \bigcup \{T_I\}$ and task T_I is an initialization task required by the engine in order to start the workflow execution by creating the instance of the first actual task.

Using Relation 3.3, workflows can also be defined as transitions between different tasks, with ϵ being a transition function.

Having a set of reactions is however not sufficient as we could have an incomplete chaining which would not lead to the desired outcome. As a consequence we define the notion of *consistent workflow* as: a workflow where the chain of reaction leading the initial input to its outcome (goal) is unbroken. Any workflow not obeying the previous rule is in our assumption *inconsistent*.

PROPOSITION 3.1. *Given a consistent workflow, the previously defined ϵ function is surjective.*

Proof. Obvious as given a consistent workflow each task part of the co-domain M_i has at least one predecessor task from the domain M_i^*. □

The previous proposition is useful for both validating workflows using a backwards chaining method and generating them starting from a given goal. Section 4.3 will detail these aspects.

Given a solution which can contain several others, reactions occur in a parallel and non-deterministic way. They also proceed automatically to the next surrounding solution once all reactions within the current one have finished.

Section 4.2 will expose a rule based workflow language built on top of the previously outlined model and introduce a simple notation based on the ϵ function that follows on an ECA approach. The proposed notation for task execution is simpler and thus more user friendly than the Gamma formalism introduced by Németh [32, 33].

4. The OSyRIS Platform. The proposed platform consists of several components such as the workflow engine, a language interpretor, a scheduling module, an automatic workflow generator and a visual workflow designer and administrator. Figure 4.1 presents an overview of the architectural components used by OSyRIS

as well as their inter-connections. Its modular design allows each major component (engine, visual designer, workflow generator and administrator) to work independently from each other. The following sections will detail each component.

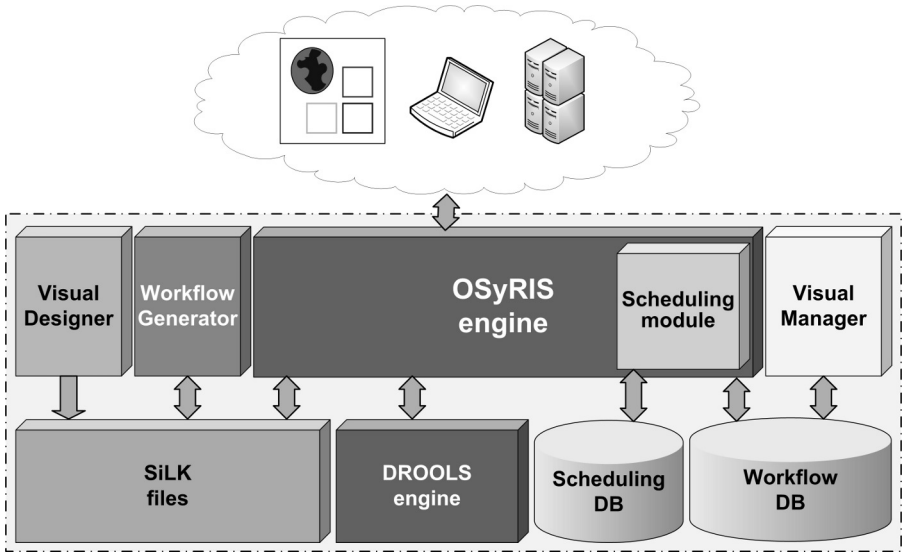


FIG. 4.1. *Architecture of the OSyRIS platform*

The OSyRIS (Orchestration System using a Rule based Inference Solution) workflow engine was designed in order to provide a solid and reliable workflow platform for complex image processing requests. It has proven to be useful in a wider range of problems than the initial scope of binding together image processing operators, examples including modeling scheduling algorithms as detailed in Section 4.5. The platform also had to be capable of adapting to resource and network failures or changes by allowing task reallocation to the best available service and permitting workflow restarts from previous failure points. Task reallocation is essential as it deals with large images up to several gigabytes each. Taking advantage of both the ECA based approaches regarding task composition [6, 29, 31, 41] and the nature inspired workflow formalisation (Section 3), a simple and yet expressive language called SiLK has been developed to cope with the problem of mixing together into a workflow different image processing operators. Details about the language capabilities will be further explained in Section 4.2.

ECA allows the separation of workflow logic from the workflow engine without having to manage with the complexity of workflow constructs. Engines following the ECA approach only have to understand a single construct that models a rule-based transition from several tasks to others. The transition is usually triggered by some events and conditions. Starting from this construct a wide range of classic constructs can be built without modifying the engine itself. Furthermore this approach allows to easily modify at runtime the workflow structure by inserting and/or retracting execution rules.

What differentiates OSyRIS from other solutions [1, 6, 16, 29] is the nature inspired approach and the integration of a scheduling module based on the same SiLK language and governed by the same engine. This mechanism allows for workflow tasks to be assigned using a chemical paradigm in which multiple scheduling heuristics are chosen at runtime based on the conditions existing at that moment for triggering certain rules. Further details on this topic will be addressed in Section 4.4.

4.1. The OSyRIS Engine. The OSyRIS engine (see Figure 4.1) is built on top of the Drools Expert [6] inference engine which uses a modified object oriented version of the RETE [18] algorithm for rule matching. The engine can be either embedded inside any Java application or exposed as a WS which can be later invoked by any SOAP enabled client. It also allows users to submit SiLK written workflows which are then translated into DROOLS rules. Any syntactic errors are handled at this point.

Before executing particular workflows two abstract classes need to be implemented. The first one is called *OSyRISwf* and represents the application specific extension of the OSyRIS workflow engine. It offers basic functionality such as executing the workflow, querying its result (the result of the last executed task in the

workflow), finding the output of a particular workflow task, retrieving the workflow (or task) status. Each workflow is uniquely identified by an ID which can be used when an asynchronous workflow execution is considered. An asynchronous execution means that the workflow is either started in a different thread or exposed as a WS, each of them periodically queried for information. Besides these features users can implement their own custom extensions according to the application needs. The second one, called *Executor*, handles specific application details such as discovering proper resources for task execution and communication between services and the engine. Once they are implemented the OSyRIS engine handles the actual rule execution.

An issue which cannot be neglected and which all workflow engines have to deal with is represented by the inter-task communication. In our case the OSyRIS engine offers two kinds of communication formats: SOAP based and message queue based. SOAP is predominantly used in cases where the engine is exposed as a WS while a message queue service is a preferable choice when the engine is embedded inside Java applications. OSyRIS uses Apache ActiveMQ [38], an implementation of the Java Message Service 1.1 [2] specification, as message queue system and provides an interface for users to extend by adding their own functionality. The communication is handled by the *Executor* class which is responsible for sending requests and receiving answers from services running tasks in a user specified format.

When using an ECA approach, the output result of a workflow represents the output of the lastly executed task after a chain of rules has been fired. This task could be different from the actual final task when inconsistent workflows (see Section 3) are created or incompatible tasks are joined together. Task incompatibility means that Right Hand Side (RHS) tasks linked to Left Hand Side (LHS) tasks cannot use the output of the latter ones as input. For instance an image processing task receiving as input an image will not be compatible with a task producing a number when the former needs to send its output to the latter. This problem can also occur due to unavailable specialized services for solving RHS tasks and can be avoided at design time, by applying model checking or by ensuring that a user using a visual tool cannot generate such incompatibilities. Runtime checks can also be produced but they will mainly lead to exceptions which halt the workflow execution in case no other valid rule can be triggered.

Much of the workflow information is stored into a database (see Figure 4.1) for later query and reports. This information includes task execution times, input and output data for each task, number of instances for each task, execution status (both for workflows and tasks) and used resources. The stored information can be later on used either for creating statistics on resource usage and task characteristics or for restarting previously failed workflows. The database information is partly used by the Workflow Manager described in Section 4.7.

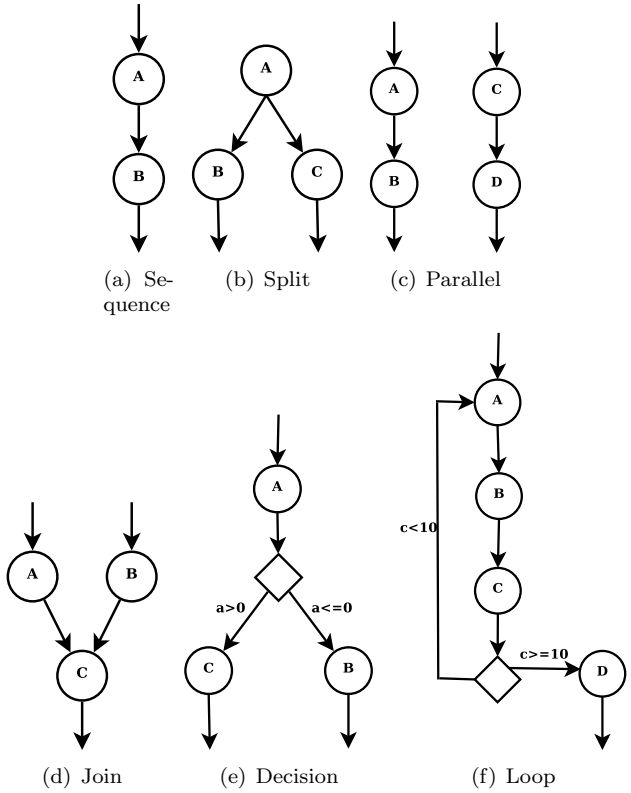
The execution path is also stored in a log. This log can be used for handling events such as workflow execution failures or when taking scheduling decisions by the task scheduling component discussed in Section 4.4.

4.2. The SiLK Language. Workflow languages need not be verbose but have a simple syntax so that inessential syntax overhead should be avoided and be expressive enough to be able to cope with a wide range of constructs and situations without needing to introduce new constructs. The Simple Language for worKflows (SiLK) is the result of several design attempts involving both XML [43] and rule based workflow languages [6] as means of expressing OSyRIS workflows. The goal was to develop a simplified language for handling image processing requests and yet not lose its applicability to other fields.

XML based approaches [40, 43] ignore syntax simplicity as they add too much unnecessary information and offer a limited set of control constructs making them inherently verbose. Therefore it is the user who has to modify both the language and the parser whenever new constructs are required. As an alternative we also studied ECA languages such as the HOCL language [32] whose syntax we found too complicated for actual user oriented usage and more suitable for formalism description.

SiLK was first described in an incipient form in our paper [19]. It offers elementary constructs such as *sequence*, *parallel*, *split*, *join*, *decision* or *loop* (see Figure 4.2).

Besides these constructs it also offers the possibility to define tasks. Tasks are the correspondents of molecules within chemical reactions and are viewed as black boxes with certain mandatory *attributes* and an arbitrary number of *meta-attributes* attached to them. In order to maintain the generality of the language the only mandatory attributes related which can be assigned to tasks are represented by input/output ports and a special meta-attribute which will be addressed in greater detail in what follows. Each task belonging to a workflow must have at least one input and one output port defined by using these very two terms as keywords (e.g. *i1:input*). Besides these attributes, users can also define their own workflow specific meta-attributes. Meta-attributes are defined as a *"name"="value"* pair.



(a) Sequence (b) Split (c) Parallel

(d) Join (e) Decision (f) Loop

FIG. 4.2. *Several types of elementary workflow constructs*

The following code fragment shows how we can define a task with one input initialized with a predefined value, one output port and one meta-attribute:

```
A := [i1:input="image.jpg", o1:output, "processing"="grayscale"];
```

The value is admitted for storage inside the meta-attributes or as initial value for a certain port. It can contain either a plain text to be sent directly to the service or a pointer (ID) to a database (table, element, etc.) or file (line etc.) which is interpreted by the custom *Executor* class. The value of the pointer is either forwarded to the WS for handling or stored in a database.

Another use for meta-attributes is to define and use task semantics and ontologies. Ontologies require users to define a set of concepts or tasks in our case, attributes related with them and relationships between them. SiLK allows to easily define task semantics or ontologies by using task attributes, meta-attributes, and task relationships through the use of inference rules (as it will be detailed in what follows).

A SiLK file represents not only a workflow formalization but also an actual task ontology. In addition to this definition of relationships users can also embed them within task definitions as meta-attributes. Meta-attributes also allow users to define task relationships without using inference rules. This approach is however not safe as it can lead to problems such as inconsistent workflows due to user errors. As an alternative a backwards chaining generating module will be detailed in Section 4.3.

Within a SiLK file task definitions must precede rules which are checked by the OSyRIS engine for validity against already known tasks.

SiLK rules are defined based on the ϵ function defined in Relation 3.3. The new notation allows us to express transitions from one multiset of tasks to another as follows: `LHS -> RHS | condition, salience` syntax where the *condition* and the *salience* are optional. The RHS and LHS are made up of tasks separated by *commas* and linked with each other through the use of variables attached to ports:

```
A[a=o1], B[b=o1] -> C[i1=a#i2=b]
```

In the previous example a and b represent variables which bind the output ports *o1* of tasks A and B to the input ports *i1* and *i2* of task C. More than one port can be bound to a variable inside a rule and in this

case the bindings are separated by the # character. If more than one task exists in the RHS then each of them gets executed in parallel by the OSyRIS engine inside a separate thread.

Any RHS task linked to multiple variables must have every variable linked to a different input port. Generally it can be said that it is prohibited to have multiple incoming edges attached to a single input port. This restriction does not stand in case of LHS tasks where we can have the same output port linked to more than one RHS input ports as it happens for a *split* construct.

As previously mentioned a rule can have optional *condition* and *salience*. The salience is represented by an integer number (negative or positive) and translates the importance of the rule with large values making the rules more important and increasing their chance of execution in case multiple rules are ready for firing at the same time. The condition can be any logical statement which uses integer, floating point or string operands. The following example shows a set of rules which mimic a loop construct by using conditions:

```
A[a=o1] -> B[i1=a];
B[b=o1] -> C[i1=b];
C[c=o1] -> A[i1=c] | c < 10;
C[c=o1] -> D[i1=c] | c >= 10;
```

The loop construct can also be viewed as a chain of reactions where we eventually end up with the initial reactant which would trigger the entire chain all over again.

Inside rules users can also manipulate task behaviours. More precisely the behaviour of a LHS task refers to whether its *instance* gets consumed or not after triggering a rule or to how many *instances* get created when a RHS task completes its execution. The default behaviour is *consume=true* for LHS tasks and *instances=1* for RHS ones. The following example shows how we can tell the engine not to consume the LHS task and to create 5 instances of the resulting RHS task:

```
A[a=o1#consume=false] -> C[i1=a#instances=5];
```

Task instances are nothing else than a programming construct to represent the notion of multisets.

Due to the fact that by using rules we allow for implicit parallelism by introducing the concept of task instances we can also create explicit rule sequencing. As an example we can consider the two rules where LHS task A has only one initial instance and each of them produces another instance of task A. In this case the rules cannot fire simultaneously and one of them needs to wait for the other to produce the necessary task instance. When creating new task instances these are added to the already existing ones. Considering two rules which create 2 respectively 3 instances of the same task A with no instance of it being consumed in the process we end up with 5 instances of the same task. The number of task instances can also be set during task definition by using a special meta-attribute called *instances* which receives a numerical value. The default behaviour is of creating zero instances during task definition. Task instances also allow for users to specify loop constructs with a finite number of steps such as the `for` in Java. Thus the exit condition is marked by the number of task instances which continue to trigger rules:

```
# Rule 1:
A[a=o1] -> B[i1=a#instances=10], C[i1=a];
# Rule 2:
C[c=o1], B -> C[i1=c];
```

Moreover task instances are also useful when considering scheduling heuristics based on task replication as it allows multiple instances of the same task to execute on different services. For instance the following lines of code show how 5 instances of the same task B are created and only a single one is being used afterwards for executing task C:

```
A[a=o1] -> B[i1=a#instances=5];
B[b=o1] -> C[i1=b];
```

The correct destruction of the remaining four tasks needs however to be dealt with from inside the custom *Executor* class.

Task instances also allow to easily manipulate workflow related operations such as pause, resume, restart and abort. This is primarily due to the fact that task instances are the main rule triggers in OSyRIS.

As previously mentioned by using a single rule construct it is possible to simulate basic workflow constructs such as *sequence*, *parallel*, *split*, *join*, single or multiple *decision*, *synchronization*, *loop* or *timers*. All of them

except the *timer* can be constructed using variations of rules containing one or more input/output ports and conditions.

Timers are not explicitly defined in the SiLK language but can be simulated by using so called idle services which receive as argument a numerical constant representing time in milliseconds and return after that particular time period.

When dealing with reactions within a solution we can also assume that we have more than one sub-solution contained in the main solution. Paper [7] details how reactions within such a complex solution occur. Starting from a sub-solution, reactions consume the task instances inside it and only then they proceed to the parent solutions triggering other reactions and so forth until all possible reactions have triggered. SiLK offers the possibility to mimic this behaviour by allowing users to specify rule domains. Each rule domain is identified by an integer number and a rule is considered by default to belong to the domain 0, unless it is explicitly attached to one. At any given time only one domain can be active and only the rules inside it can execute. However users can define transition rules where some LHS tasks belong to a domain different from the other tasks. In this case, after firing the engine, the platform will automatically select the new domain (to which the RHS tasks belong to) as the active one.

Figure 4.3 shows three rules belonging to a workflow. The first two belong to domain 1 and the third one belongs to domain 2. Assuming that the first rule in domain 1 can fire it will determine the activation of domain 2 and the subsequent execution of rule 3, and not rule 2 which belongs to domain 1. The next code fragment shows the representation in SiLK of the example:

```
# Rule 1 in domain 1:
1 : A[a=o1] -> B[i1=a], 2:C[i1=a];
# Rule 2 in domain 1:
1 : C[c=o1] -> D[i1=c];
# Rule 3 in domain 2:
2 : C[c=o1] -> E[i1=c];
```

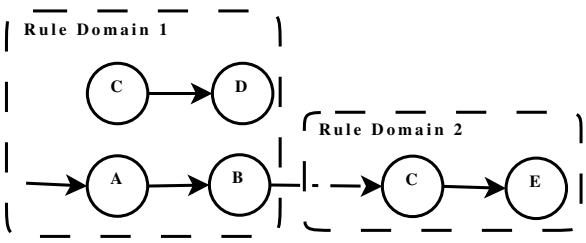


FIG. 4.3. *Switching between rule domains*

As it can be noticed from the previous example, SiLK rule domains allow constructing rule sets without inducing a strict hierarchy. Thus SiLK domains are more complex than the solutions introduced by [7] (see Section 3) as they allow for the workflow to switch freely between multiple sets of rules when necessary. On the other hand solutions allow transitions only when their reactions are no longer active.

4.3. Automatic Workflow Generator. There are cases when the user does not know or simply does not have the time to design the chain of rules making the desired workflow. In this case it is mandatory to have the possibility to automatically generate the workflow given its goal. OSyRIS offers the *WorkflowExtractor* class as part of the Workflow Generator module (see Figure 4.1) for cases where automatic generation is required. There are three basic approaches to this problem: using task semantic information; relying exclusively on rules; mixing the last two.

The first one concerns using task semantic information [14, 26] stored inside the task definitions from the rule pool. This approach has a major disadvantage as not all tasks might have the necessary semantic information required for backwards chaining. Moreover in absence of a standard for representing ontologies different users might use different information for representing tasks so the engine might consequently fail to recognize a relationship between them. The next rule set shows how additional information can be added inside task definitions into a simple ontology. While XML text can be embeded inside the meta-attributes the example uses a plain text approach and is actively used in the project concerning image processing mentioned

in Section 4. The format serves to identify corresponding services through an UDDI, based on the functions they expose. For example:

```
A0:=[o1:output="image_normal", ‘‘instances"="1"];
A:=[i2:input, o1:output, "processing"="image_red
      extract-band(band,image_normal)",
 ‘‘argument-list"="<band=red>"];
B:=[i2:input, o1:output, "processing"="image_infrared
      extract-band(band,image_normal)",
 ‘‘argument-list"="<band=infrared>"];
C:=[i1:input, i2:input, o1:output, "processing"=
 ‘‘image_ndvi compute-ndvi(image_red,
       image_infrared)"];
```

In the previous case the information details the processing each task is going to do, its output type and the arguments of the call. Ports have indices corresponding to the position of the function arguments (i.e input port *i2* is linked to the second argument of the *extract-band* function called *image*) and optionally arguments can have initial default values as specified by the *argument-list* meta-attribute. It can be also noted that we have chosen a C-like function prototype naming for describing task processing. Starting from the task meta-attributes description the backwards chaining creates corresponding workflows. In order to make this happen the engine needs to be able to parse and interpret relevant meta-attribute information (it is up to the user to implement this custom handler). One solution is found corresponding to the simple example previously described:

```
A0[a=o1] -> A[i2=a], B[i2=a];
A[a=o1], B[b=o1] -> C[i1=a#i2=b];
```

The second approach is relying solely on rules described in the rule base. While this approach is somewhat easier as the rules already offer a precedence relationship between tasks, it could provide faulty workflows as the rule base extends by receiving more and more rules which could conflict with older existing ones.

The last approach represented by a mixture of the previous two is currently under consideration for future implementation.

When applying the two approaches described above, it is necessary to exist an initial rule base containing all the allowed relations between tasks based on either rules or meta-attributes. From this rule base can be created a set of disjoint graphs that are essential when obtaining the list of relevant workflows for the given goal. The backward chaining phase is generated by using a depth first algorithm. The workflow execution phase begins with the user choosing the desired goal from a list of available tasks extracted from the RHS tasks inside the rule base. Then the final rules are extracted from the rule base using backwards chaining. After several workflow solutions have been generated, the user can choose one of them depending on available input and then execute it by using the OSyRIS engine.

As an example we can consider the following code that shows a simple rule base:

```
A:=[i1:input="initial input A", o1:output,
 "proc"="operation-A"];
B:=[i1:input, o1:output, "proc"="operation-B"];
C:=[i1:input, o1:output, "proc"="operation-C"];
D:=[i1:input, o1:output, "proc"="operation-D"];
E:=[i1:input, o1:output, "proc"="operation-E"];
M:=[i1:input, o1:output, "proc"="operation-M"];
F:=[i1:input, i2:input, o1:output,
 "proc"="operation-F"];
P:=[i1:input="initial input P", o1:output,
 "proc"="operation-P"];

A[a=o1] -> B[i1=a],C[i1=a];
B[b=o1] -> D[i1=b];
D[d=o1] -> E[i1=d];
C[c=o1] -> E[i1=c];
E[e=o1],C[c=o1] -> F[i1=e#i2=c];
```

```
M[m=o1] -> F[i1=m];
P[p=o1] -> M[i1=p];
```

This rule base can be represented as the graph seen in Figure 4.4.

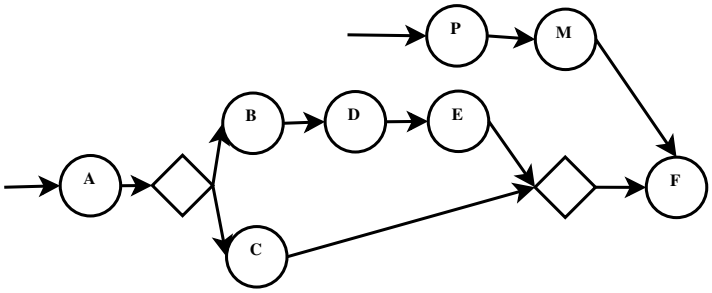


FIG. 4.4. *Rule Base Chaining Example*

Given the desired output task F the following two possible workflows are automatically created:

```
M:=[i1:input, o1:output, "proc"="operation-M"];
F:=[i1:input, i2:input, o1:output,
 "proc"="operation-F"];
P:=[i1:input="initial input P", o1:output,
 "proc"="operation-P"];

M[m=o1] -> F[i1=m];
P[p=o1] -> M[i1=p];
```

and respectively:

```
A:=[i1:input="initial input A", o1:output,
 "proc"="operation-A"];
B:=[i1:input, o1:output, "proc"="operation-B"];
C:=[i1:input, o1:output, "proc"="operation-C"];
D:=[i1:input, o1:output, "proc"="operation-D"];
E:=[i1:input, o1:output, "proc"="operation-E"];

A[a=o1] -> B[i1=a],C[i1=a];
B[b=o1] -> D[i1=b];
D[d=o1] -> E[i1=d];
C[c=o1] -> E[i1=c];
E[e=o1],C[c=o1] -> F[i1=e#i2=c];
```

It can be noticed that there have not been not produced two different workflows one for the rule chain $A \rightarrow C \rightarrow F$ and another for $A \rightarrow B \rightarrow D \rightarrow E \rightarrow F$ as task F is produced by the join of tasks E and C.

Backwards chaining is possible as the ϵ function (see Relation 3.3) is right-invertible, a property which follows from its surjectivity (see Proposition 3.1). Thus a rule chain could be determined, if existing, starting from the desired goal.

The workflow generator produces merely *abstract workflows* [32] where only the logic of the problem is expressed without attaching any means of solving it. In order to accomplish the latter, a mapping of tasks on services is required. This action produces a *concrete workflow* which can then be executed by the engine. In order to obtain full dynamism and autonomy the conversion between abstract and concrete workflows must take place during runtime by means of a resource selection and scheduling mechanism similar to the one presented in Section 4.4 and Section 4.5.

4.4. Resource Selection and Discovering. Resource selection is important in any workflow system as it can lead to unbalances across the DE. To overcome the problem, DEs usually try to offer task balancing or scheduling techniques. OSyRIS implements several ways for selecting resources, but it is up to the user to select

the best suitable choice for the situation at hand. The task selection API is implemented inside the Scheduling module as seen in Figure 4.1.

The first method involves the usage of an external *Resource Selector*. This approach relies on the fact that each OSyRIS task is exposed by an external WS. An intermediate *Broker Service* added between the actual service and the engine will be responsible for dispatching tasks coming from the engine to appropriate resources. Scheduling capability can be optionally integrated inside the broker.

The second method is to implement additional functionality and link it directly with the custom built *Executor* class. In this case the scheduling passed from the *Broker Service* to an additional class called by the application specific *Executor* class. After taking a decision the *Executor* will send the task directly to the WS without using any intermediate brokers.

The last one enables adding resource selection decisions inside rules. When using it the scheduling decision will be inserted directly inside the rules. Two different approaches can be also considered in this case. The first one requires modifying the *SiLK2Rules* class responsible for translating the SiLK rules into DROOLS rules and using the *WFResource* collection which contains a list of available services. In this case a distinct rule condition must be added inside each of the OSyRIS generated DROOLS rules. This condition will enclose all the required scheduling decisions and will return one or more potential services where the task can be safely executed. The second approach implies using a separate service similar to the *Broker Service*. The sole purpose of the service is to deal with scheduling other tasks. In addition the service will be treated as normal task that is attached to the LHS of each rule. Its output consisting of a set of available services will be transmitted as input to all the RHS tasks. The service itself relies on an OSyRIS engine instance where rules represent scheduling heuristics (see Section 4.5).

Besides service selection, the service discovery plays an important role too as it enables the engine to have a pool of services to choose from. Similar to Taverna [34], services can be discovered using various techniques such as UDDIs, direct URL input, previous workflow introspection, semantic searches, etc. OSyRIS uses by default UDDIs but extendible plug-ins can be easily integrated to cope with the rest of the choices.

4.5. Scheduling mechanism. Scheduling is closely related to resource discovering and usually involves assigning tasks to resources using certain rules. OSyRIS offers these as a single module (see Figure 4.1). Following the nature inspired approach OSyRIS offers the possibility to express scheduling heuristics in the SiLK language. In this way users can define their own scheduling heuristics by adding them to the heuristics rule base. Each such scheduling heuristics could be located in its own SiLK domain. Following this approach resource selection can be augmented by selecting heuristics dynamically at runtime based on known criteria such as dimension of dependent data, network and resource heterogeneity, etc.

Resource selection is essential as it is known [10, 12, 28] that, given the grid characteristics, scheduling heuristics perform differently in dissimilar scheduling scenarios. Based on previous work and knowledge on the network, resource and task characteristics, the best fitted scheduling heuristics could be selected during runtime from a general rule base by adding conditions for triggering the reactions inside the corresponding solution. To achieve this we must first define the tasks and rules each scheduling heuristics has. Figure 4.5 shows such a decomposition for the case of the DMECT heuristics [21].

The DMECT scheduling heuristics is represented by the following fragment of code:

```
while (exist tasks in T_list) do {
  foreach (task in T_list) do
    if (task is new) then
      assign task to random queue;
  foreach (queue in queue_list) do
    if (queue not sorted
      sort tasks in current queue descending by TWT;
  repeat {
    foreach (queue in queue_list) do
      foreach (task in current queue) do
        if (task exceeds time constraint) then {
          find new queue based on condition;
          find position to insert such that tasks
           remain ordered descending by TWT;
```

```
          place task on new queue at the found position;
        }
  } until (not task has been moved)
}
```

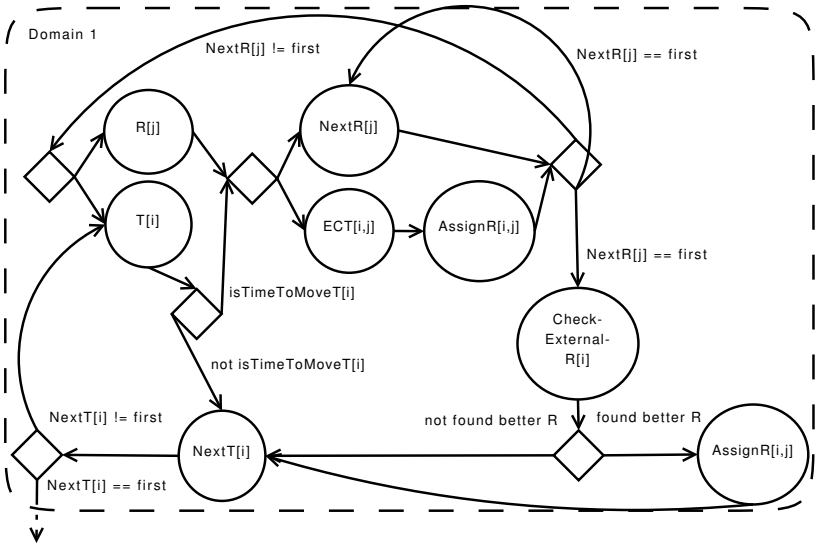


FIG. 4.5. *DMECT workflow*

Briefly, DMECT simply checks for each task if a certain waiting time limit has been exceeded, and in case it has, it tries to relocate the task to a better resource queue. The reallocation is done so that the tasks remain ordered with the oldest task in front of the queue and the youngest at the end.

The rule base in this case would be comprised of the following tasks and rules:

```
R:=[i1:input,o1:output,"proc"="get-current-resource"];
T:=[i1:input,o1:output,"proc"="get-current-task"];
ECT:=[i1:input,i2:input,o1:output,o2:output,
 "proc"="compute-run-time"];
NextR:=[i1:input,o1:output,"proc"="get-next-resource"];
NextT:=[i1:input,o1:output,"proc"="get-next-task"];
AssignR:=[i1:input,i2:input,o1:output,o2:output,
 "proc"="assign-task-to-resource"];
CheckExternalR:=[i1:input,o1:output,o2:output,
 "proc"="check-other-resources"];
IsTimeToMove:=[i1:input,i2:input,o1:output,
 "proc"="check-can-move-task"];

1:R[r=o1],T[t=o1] -> IsTimeToMove[i1=t#i2=r];
1:IsTimeToMove[a=o1] -> NextR[i1=r],ECT[i1=t#i2=r] | a=="yes";
1:ECT[t=o1#r=o2] -> AssignR[i1=t#i2=r];
1:NextR[r=o1],AssignR[t=o1#r1=o2] -> R[i1=r],T[i1=t] | r!="first";
1:NextR[r=o1],AssignR[t=o1#r1=o2] -> NextR[i1=r],CheckExternalR[i1=t]
                    | r=="first";
1:CheckExternalR[t=o1#r=o2] -> RelocateT[i1=t#i2=r] | r=="null";
1:CheckExternalR[t=o1#r=o2] -> NextT[i1=t] | r !="null";
1:IsTimeToMove[a=o1] -> NextT[i1=t] | a!="yes";
1:NextT[t=o1] -> T[i1=t] | t!="first";
1:NextT[t=o1] -> 2:\dots  | t=="first";
```

The same transformation into rules can be operated on other existing heuristics as well [45].

Offering the scheduling module as part of the OSyRIS platform can rise several questions. How is the data about services centralized? How does this impact on the overall schedule since only the tasks in the executing workflow are visible for each engine instance? To answer these issues we need to consider the centralized database used by the OSyRIS platform (see Figure 4.1). The database stores all the data about the workflows including information about tasks and services (including discovery registries). Each workflow has access to the same collective data and provides the scheduling mechanism with a global view on the overall schedule. As a consequence a scheduling decision, taken by a specific workflow, uses the data on all other running workflows. This approach allows each workflow to have its own planner and at the same time offers a global view on the status of the resources and of the workflow tasks.

OSyRIS can handle a distributed scenario where multiple engines access the same pool of services and use different databases. In this case a decentralized agent approach is needed. Each agent oversees a single engine and exchanges information about tasks and services with others. Therefore each time a scheduler tries to take a decision it has a global perspective over the environment configuration including service locations and submitted tasks. Such an overview allows each engine instance to take the best decisions in case failures of the network fabric or in service availability occur. Work in this area is currently under development with prototype platform described in our paper [20].

4.6. The Visual Workflow Designer. Visual workflow design can help users by allowing them to seamlessly create both syntactically and semantically correct workflows. The OSyRIS platform offers such an option (see Figure 4.1). The users can create workflows without needing any knowledge regarding the SiLK language. The graphical representation can be saved, loaded and exported to various formats including SiLK which can then be used for input to the engine. The user can create workflows by using two elementary visual constructs, tasks and links between them which can be placed on the canvas using a drag and drop technique. By default each task has one input and one output port but the user can attach more together with optional meta-attributes. Conditions can be optionally attached to links in order to simulate decision and loop constructs and any restriction imposed by the SiLK language (as described in Section 4.2) are dealt with at this design stage. Moreover types can be added to them by specifying whether they are used to create synchronization rules or not. Figure 4.6 shows this interface together with a simple workflow made up of a sequence of image processing tasks.

4.7. The Workflow Manager. Workflow tasks are executed by WS located in various geographical places. This can lead to problems such as: response delays due to either time required to transmit large amount of data or network overuse; task failures due to resource unavailability because of service failures etc. Even though these problems can be overcome by taking adaptive measures such as dynamic task rebalancing, task cloning or other methods, it is important to overview and keep track of all these problems for later statistics, rescheduling and other administration policies. OSyRIS provides an administrative interface (see Figure 4.1 and Figure 4.7) where can be observed information such as: the status of each workflow task (not running, running, completed, paused or aborted); the service which executes them and if available the location; how long it took to execute them or how often a certain service is used.

5. Case Study: Composing Image Processing Operations. OSyRIS was initially developed for the GiSHEO e-learning environment [22]. The goal of GiSHEO is to offer an easy to use environment where students can apply various processing operations on large satellite images (several hundreds of gigabytes each). As it has been shown [36] image operation composition can be successfully used in natural sciences where it can offer relevant information such as the position of ancient human settlements, flood coverage and evolution, vegetation areas etc.

The OSyRIS engine has the role of receiving and orchestrating client requests comprising of a workflow made up of several image processing operations. A typical client request consists of a workflow description expressed in SiLK. The engine receives the request, checks the workflow validity, sends back a workflow id token and fires the rules. Each workflow task receives and produces one or more satellite images depending on the underlying processing operation. Once the workflow finishes the result is sent back to the client together with the id token. Due to the large size of the images the workflow will not handle the actual images but will instead reference them using unique identifiers. Based on each reference the image processing WS will take the image from a repository, process it and return a new reference to the resulted image.

A particular requirement that OSyRIS needs to cope within GiSHEO is that of handling initial inputs consisting of several images. This usually happens when the user selects from the interface a region larger

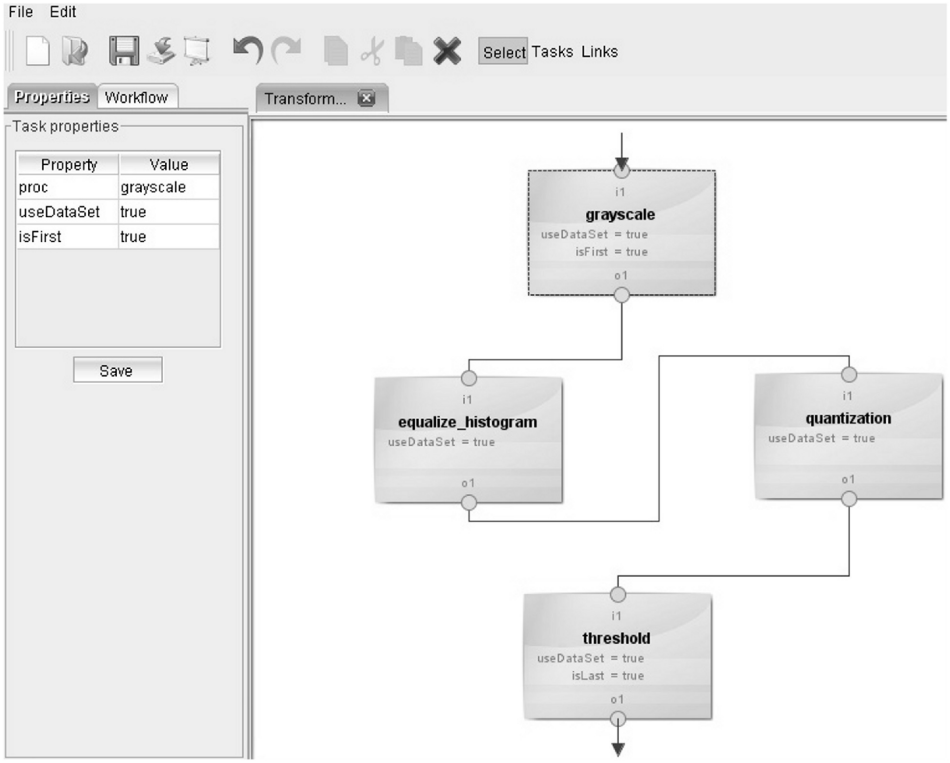


FIG. 4.6. *User interface for the Visual Workflow Designer*

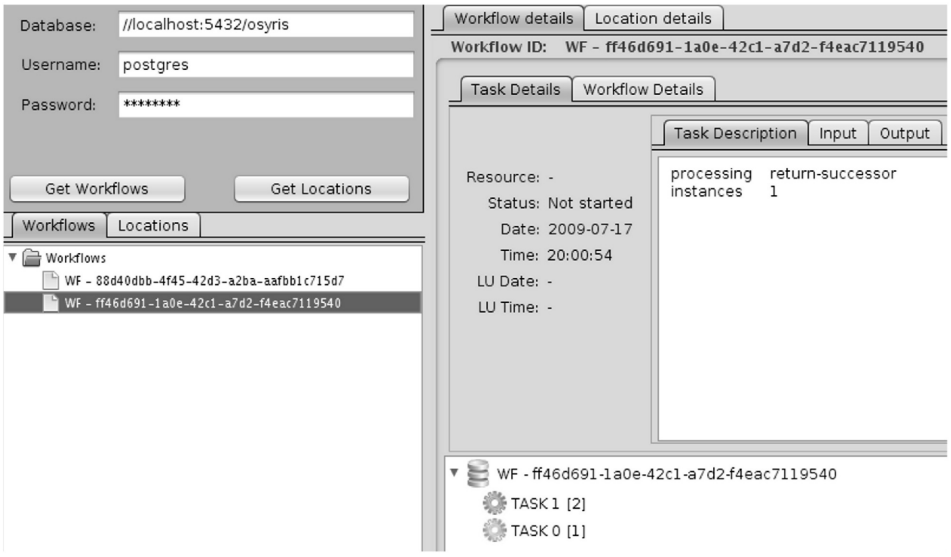


FIG. 4.7. *User interface for the Visual Workflow Manager*

than a single image. In this case the engine replicates the initial workflow rules and tasks up to a number of times equal with that of the input images. As a result we end up with a workflow that is executed several times but with different inputs. The execution is performed in parallel for every input image case and in a single OSyRIS engine instance. Each workflow replica is considered part of the initial workflow request which is considered to be completed only when all replica workflows have finished executing. To handle this issue a special meta-attribute called *isLast* is assigned to each of the workflow replicas final tasks. Only when all the existing final tasks have completed is the workflow considered to be finished.

So GiSHEO also requires additional meta-attributes including *dataset*, *datagroup*, *processing* and *argument-list*. *Dataset* is an identifier that represents the dataset containing the image; *Datagroup* is an identifier for the datagroup where the image *dataset* is stored in; *isLast* is used to mark that the task is the final task of the workflow; *processing* represents the image processing operation assigned to the task and is described using C function prototype syntax; and *argument-list* contains the list of predefined processing arguments.

The following SiLK code presents an example consisting of a workflow which computes the Normalized Difference Vegetation Index (NDVI) [30]. The NDVI is computed by applying the following formula: $NDVI = (NIR - RED)/(NIR + RED)$, where NIR and RED stand for the near-infrared and respectively the red bands of the multi-spectral satellite image. The workflow first extracts the red and near-infrared bands and then uses the output images to compute the NDVI:

```
# Initial activation task
A0:=[o1:output="datasetID","instances"="1"];
# The following tasks belong to the processing workflow
A:=[i2:input, o1:output, "processing"=
 ‘‘image extract-band(band#image)","argument-list"="<band=red>"];
B:=[i2:input, o1:output,"processing"=
 ‘‘image extract-band(band#image)","argument-list"="<band=infrared>"];
C:=[i1:input, i2:input, o1:output, ‘‘datagroup"="datagroupID",
 ‘‘dataset"="datasetID","processing"="image compute-ndvi(image#image)",
 ‘‘isLast"="true"];
# Extract red and infrared bands from the initial image
A0[a=o1] -> A[i2=a], B[i2=a];
# Compute NDVI using red and infrared bands
A[a=o1], B[b=o1] -> C[i1=a#i2=b];
```

6. Conclusions and Future Work. OSyRIS aims to provide a flexible and self adaptive environment in which tasks are linked naturally by ECA rules following a nature inspired paradigm. The platform consists of several components: workflow engine, the SiLK language interpretor, the scheduling module, the automatic workflow generator and the visual workflow designer and administrator. SiLK, a simple workflow language, allows to easily define task semantics or ontologies by using task attributes, meta-attributes, and task relationships through the use of inference rules. It is based on the concepts described in [32, 33].

OSyRIS has several particularities that distinguish it from other existing solutions. First, it uses a simple rule mechanism, inspired from chemical reactions, from which users can develop their own custom constructs. Second, it has a self adaptive resource selection and a scheduling mechanism mixed with the support for task semantics and ontology definition. Third, self-adaptiveness is applied also in terms of failure handling and task scheduling.

OSyRIS has been tested on the GiSHEO e-learning environment where it has been successfully used to compose image processing requests with applications in the fields of archaeology and geography. Further comparisons with other approaches are planned in the near future. Additional technical details on the OSyRIS platform can be found at the GiSHEO website [22].

REFERENCES

[1] *Activebpel homepage*, 2009. `http://www.activebpel.org/` (accessed November 18th, 2009).
[2] *Java message service 1.1*, 2009. `http://java.sun.com/products/jms/` (accessed November 19, 2009).
[3] M. J. Adams, *Facilitating Dynamic Flexibility and Exception Handling for Workflows*, PhD thesis, Queensland University of Technology, 2007.
[4] A. Ankolekar, M. Burstein, J. Hobbs, and et al., *DAML-S: Semantic Markup for Web Services*, 2001. `http://www-2.cs.cmu.edu/~softagents/papers/SWWS.pdf` (accessed March 30, 2009).
[5] M. Arbib, *The Handbook of Brain Theory and Neural Networks 2nd Ed.*, MIT Press, 1995.
[6] M. Bali, *Drools JBoss Rules 5.0 Developer's Guide*, Packt Publishing, 2009.
[7] J. Banâtre, P. Fradet, and Y. Radenac, *Programming self-organizing systems with the higher-order chemical language*, International Journal of Unconventional Computing, 3 (2007), pp. 161–177.
[8] J.-P. Banâtre and D. Le Métayer, *Programming by multiset transformation*, Commun. ACM, 36 (1993), pp. 98–111.
[9] W. Banzhaf, P. Nordin, R. Keller, and F. Francone, *Genetic Programming—An Introduction*, Morgan Kaufman Publishers, Inc. San Francisco, California, 1998.

[10] T. D. Braun, H. J. Siegel, and N. Beck, *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*, Journal of Parallel and Distributed Computing, 61 (2001), pp. 801–837.
[11] A. Cârstea, G. Macariu, and M. Frîncu, *Description and execution of patterns for symbolic computations*, in Procs. SYNASC09, Timisoara, Romania, September 2009.
[12] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, *Heuristics for scheduling parameter sweep applications in grid environments*, in Procs. 9th Heterogeneous Computing Workshop (HCW), Cancun, Mexico, May 2000, pp. 349–363.
[13] A. Charfi and M. Mezini, *Ao4bpel: An aspect-oriented extension to bpel*, World Wide Web, 10 (2007), pp. 309–344.
[14] S. Chun, V. Atluri, and N. Adam, *Domain knowledge-based automatic workflow generation*, in Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA'02), London, UK, 2002, Springer-Verlag, pp. 81–92.
[15] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, *Programming scientific and distributed workflow with triana services: Research articles*, Concurr. Comput. : Pract. Exper., 18 (2006), pp. 1021–1037.
[16] E. Deelman, G. Singh, M. Su, and et al., *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, Scientific Programming, 13 (2005), pp. 219–237.
[17] J. Fernández, R. Hoffmann, and A. Valencia, *ihop web services.*, Nucleic acids research, 35 (2007), pp. 21–26.
[18] C. Forgy, *Rete: a fast algorithm for the many pattern/many object pattern match problem*, Expert systems: a software methodology for modern applications, (1990), pp. 324–341.
[19] M. Frîncu, S. Panica, M. Neagul, and D. Petcu, *Gisheo: On demand grid service-based platform for earth observation data processing*, in Procs. of HiperGRID'09, Politehnica Press, 2009, pp. 415–422.
[20] M. E. Frîncu, *Distributed scheduling policy in service oriented environments*, in Procs. of the 11th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing SYNASC'09, IEEE Conference Publishing Series, IEEE Press, 2009.
[21] ———, *Dynamic scheduling algorithm for heterogeneous environments with regular task input from multiple requests*, in Procs. of the 4th Int. Conf. in Grid and Pervasive Computing GPC'09, vol. 5529 of Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 199–210.
[22] GiSHEO, *On demand grid services for high education and training in earth observation*, 2009. http://gisheo.info.uvt.ro (accessed April 22, 2009).
[23] M. Greenwood, *Xscufl language reference*, 2004. `http://www.mygrid.org.uk/wiki/Mygrid/WorkFlow#XScufl_workflow_definitions` (accessed March 30, 2009).
[24] M. Hiel, H. Weigand, and W. van den Heuvel, *An adaptive service-oriented architecture*, in Enterprise Interoperability III, ed. K. Mertens, and R. Ruggaber, and K. Popplewell, and X. Xu, Springer, 2008, pp. 197–208.
[25] N. Imyanitov, *Is this reaction a substitution, oxidation-reduction, or transfer?*, Journal of Chemical Education, 70 (1993), pp. 14–16.
[26] S. Lu, A. Bernstein, and P. Lewis, *Automatic workflow verification and generation*, Theoretical Computer Science, 353 (2006), pp. 71–92.
[27] G. Macariu, A. Carstea, and M. Frincu, *Service-oriented symbolic computing with symgrid*, Scalable Computing: Practice and Experience, 9 (2008), pp. 11–25. http://www.scpe.org/vols/vol09/no2/SCPE_9_2_04.pdf (accessed April 16, 2009).
[28] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund, *Dynamic mapping of a class of independent tasks onto heterogeneous computing systems*, Journal of Parallel and Distributed Computing, 59 (1999), pp. 107–131.
[29] R. Muller, G. Greiner, and E. Rahm, *Agent work: a workflow system supporting rule-based workflow adaptation*, Data Knowl. Eng., 51 (2004), pp. 223–256.
[30] R. B. Myneni, F. Hall, P. Sellers, and A. Marshak, *The interpretation of spectral vegetation indexes*, IEEE Transactions on Geoscience and Remote Sensing, 33 (1995), pp. 481–486.
[31] C. Nagl, F. Rosenberg, and S. Dustdar, *Vidre–a distributed service-oriented business rule engine based on ruleml*, in Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06), Washington, DC, USA, 2006, IEEE Computer Society, pp. 35–44.
[32] Z. Németh, C. Pérez, and T. Priol, *Workflow enactment based on a chemical metaphor*, in SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods, Washington, DC, USA, 2005, IEEE Computer Society, pp. 127–136.
[33] Z. Németh, C. Pérez, and T. Priol, *Distributed workflow coordination: molecules and reactions*, Parallel and Distributed Processing Symposium, International, 0 (2006), pp. 260–268.
[34] T. Oinn, M. Greenwood, M. Addis, and et al., *Taverna: lessons in creating a workflow environment for the life sciences: Research articles*, Concurrency and Computation: Practice and Experience, 18 (2006), pp. 1067–1100.
[35] G. Paun, *Computing with membranes*, Journal of Computer and System Sciences, 61 (2009), pp. 108–143.
[36] D. Petcu, D. Zaharie, M. Neagul, S. Panica, M. Frîncu, D. Gorgan, T. Stefanut, and V. Bacu, *Remote sensed image processing on grids for training in earth observation*, in Image Processing, V. Kordic, ed., InTech, Vienna, 2009, in print.
[37] J. Schiff, *Cellular Automata: A Discrete View of the World*, Wiley and Sons, Inc., 2008.
[38] B. Snyder, D. Bosanac, and R. Davies, *ActiveMQ in Action (1st ed.)*, Manning Publications, 2010.
[39] D. Thain, T. Tannenbaum, and M. Livny, *Distributed computing in practice: the condor experience.*, Concurrency - Practice and Experience, 17 (2005), pp. 323–356.
[40] W. M. P. van der Aalst and A. H. M. ter Hofstede, *Yawl: yet another workflow language*, Information Systems, 30 (2005), pp. 245—275.
[41] H. Weigand, W. van den Heuvel, and M. Hiel, *Rule-based service composition and service-oriented business rule management*, Proceedings of the International Workshop on Regulations Modelling and Deployment (ReMoD'08), (2008).
[42] P. Wolniewicz, N. Meyer, M. Stroinski, M. Stuempert, H. Kornmayer, M. Polak, and H. Gjermundrod, *Accessing grid computing resources with g-eclipse platform*, Computational Methods in Science and Technologie, 13 (2007), pp. 131–141.

[43] WSBPEL, *Web services business process execution language version 2.0*, 2007. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf` (accessed March 30, 2009).
[44] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau, *Automating daml-s web services composition using shop2*, in International Semantic Web Conference, 2003, pp. 195–210.
[45] F. Xhafa and A. Abraham, *Computational models and heuristic methods for grid scheduling problems*, Future Generation Computer Systems, (2009).
[46] C. Zandron, C. Ferretti, and G. Mauri, *Using membrane features in p systems*, Romanian Journal of Information Science and Technology, 4 (2001), pp. 241—257.

Edited by: Viorel Negru, Adina Magda Florea
Received: February 27, 2010
Accepted: March 31, 2010

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.