CS-2004-03: Florida Tech, Computer Sciences, February 2004

On the Implementation of SwarmLinda*
A Linda System Based on Swarm Intelligence
(Extended Version)

Ahmed Charles[1], Ronaldo Menezes[1] and Robert Tolksdorf[2]

[1] Florida Institute of Technology, Department of Computer Sciences
150 W. University Blvd., Melbourne, FL 32901, USA
acharles@fit.edu
rmenezes@cs.fit.edu
[2] Freie Universität Berlin, Institut für Informatik, AG Netzbasierte Informationssysteme,
Takustr. 9, D-14195 Berlin, Germany
research@robert-tolksdorf.de

Abstract. Natural-forming multi-agent systems (aka Swarms) can grow to enormous sizes and perform seemingly complex tasks without the existence of any centralized control. Their success comes from the fact that agents are simple and the interaction with the environment and neighboring agents is local in nature. In this paper we discuss the implementation of SwarmLinda, a Linda-based system that abstracts Linda concepts in terms of swarm intelligence constructs such as scents and stigmergy. The goal of this implementation is to achieve many characteristics such as scalability, adaptiveness and some level fault tolerance. This paper describes our initial version of SwarmLinda and future steps to improving the implementation.

1 Introduction

Linda [7] is arguably the most important and most successful coordination model ever proposed. Such a statement can be easily verified by looking at the commercial implementations that spawned from Linda's simple idea of generative communication (e.g. TSpaces [20], JavaSpaces [5], GigaSpaces [6]).

In the past 20 years, coordination models, and in particular Linda, have proven to be quite successful in tackling the intricacies of medium-to-large-scale open systems. Yet, Linda systems may not scale well with the number of tuple spaces and processes. One of the reasons for the poor scalability is that the design of these systems still inherits ideas from early Linda systems [7, 8] which were focused on parallel computing.

* This work has been partially funded by the DAAD (Germany) and the NSF (USA) contract numbers D/03/34491 and INT-0337161 respectively.

Swarm-based systems are famous for their organization and efficiency despite their enormous sizes. Their activities are based on simple rules that can be easily implemented as computer programs. Their interaction involves only local communications.

We focus on the use of swarm-based techniques (such as approaches based on ant-colonies [14]) in tuple space systems. Our goal is to tackle the scalability problem studying how to improve the current scenario using techniques adapted from models originating from biological collective organisms. In this paper, we describe the implementation of SwarmLinda [11]. Although SwarmLinda consists of several algorithms, we focus on the two main ones namely tuple distribution and tuple retrieval.

This paper describes SwarmLinda and an initial implementation, developed in Java, succinctly. First in Section 2 we discuss common characteristics of LINDA systems. Then, in Section 3 the two algorithms implemented are described. Next, we describe how the current version of SwarmLinda has been implemented (in Section 4). Last, in Section 5, we discuss the road-map to future versions of the system.

2 Linda Systems

The LINDA coordination model is based on the associative memory communication paradigm. This means that processes cannot communicate directly but rather via shared associative memories, called tuple spaces. Access to these tuple spaces occurs via the mechanism of matching.

LINDA provides processes with primitives enabling them to store and retrieve tuples from tuple spaces. Although the names of the primitives vary slightly in different implementations, the functionalities are normally very similar. Processes use the primitive `out` to store tuples. They retrieve tuples using the primitives `in` and `rd`. These primitives take a template (a definition of a tuple) and use associative matching to retrieve the desired tuple — while `in` removes a matching tuple, `rd` takes a copy of the tuple. Both `in` and `rd` are blocking primitives, that is, if a matching tuple is not found in the tuple space, the process executing the primitive blocks until a matching tuple can be retrieved.

In addition to the primitives above some LINDA implementations provide non-blocking versions of `in` and `rd`, called `inp` and `rdp` respectively. These primitives have the same semantics of their blocking counterparts when a matching tuple can be found in the tuple space. However, they behave significantly different if a matching tuple cannot be found — they fail instead of blocking.

There are a plethora of LINDA implementations that have been proposed to deal with various weaknesses of the original model. Some of the most important include Lime [15], WCL [17], TuCSoN [13] and LogOp [18].

3 SwarmLinda

We have presented SwarmLinda in [19, 11] in condensed form while [12] is an extended description on which we build here. SwarmLinda uses several adaptations of algorithms taken from abstraction of natural multi-agent systems [2, 14].

Over the past few years, new models originating from biology have been studied in the field of computer science [16, 2, 3, 9]. The primary interest lies on using these models as techniques for finding feasible solutions to NP-hard problems.

In these models, actors (ants, termites, etc.) sacrifice individual goals (if any) for the benefit of the collective. They act extremely decentralized. The work is carried out by making purely local decisions and by taking actions that require only few computations. These characteristics alone enable these systems to scale very well. This *swarm intelligence* is an interesting opportunity to rethink scalability of tuple spaces.

The use of such principles as an alternative to data-oriented schemes could simply consist of a system where templates are modeled as ants that search for food (the matching tuples). One can understand the "world" of Linda nodes as a two-dimensional space in which ants search for food, leaving trails to successful matches.

With an ant-based optimization of the trails, shortest tuple-producer/-consumer paths can be found and used to optimize system performance: instead of querying sets of replicas, the "template-ant" goes directly to where it expects a match. Technically, this accounts to a single message interchange between the producing and consuming sites.

The above is just an illustration. A more realistic SwarmLinda should consider a few principles that can be observed in most swarm systems [14]:

Simplicity: Swarm individuals are simple creatures. They do no deep reasoning and implement a small set of simple rules. The execution of these rules leads to the emergence of complex behavior. Active entities in a SwarmLinda should also obey the principle of simplicity and be small in terms of resource usage.

Dynamism: Natural swarms adapt to dynamically changing environments. In open distributed systems, the configuration of running applications and services changes over time. If a tuple is found in a given location it does not necessarily mean that other similar tuples will be found in the same location in the future.

Locality: Swarm individuals observe their direct neighborhood and take decisions based on their local view. As the key to scalability in SwarmLinda, active entities have to perform only local searches and communicate only to direct neighbors.

To fully appreciate these principles, one needs to understand how a SwarmLinda should be organized. Linda systems do not have the idea of ants or food.

The description of a SwarmLinda is based on the following abstractions: *Individuals* are the active entities that are able to observe their neighborhood, to move in the environment, and to change the state of the environment in which they are located; the *environment* is the context in which the individuals work and observe; the *state* is an aspect of the environment that can be observed and changed by individuals.

SwarmLinda is organized as a network of nodes where each node allows connections from multiple processes. The nodes communicate with each other and transfer tuples between themselves. The topology of the network assumed is similar to a peer-to-peer (P2P) network.

In our implementation, we aim at optimizing distribution and retrieval by dynamically determining storage locations for tuples based on that particular tuple's type, and having tuples of the same type stored in clusters. It should be noted that we do *not* want to program the clustering but rather have them emerge from the algorithms implemented through the mechanism self-organization. We decided that a tuple's type should be represented by its template, because searches for tuples are done based on template, thus storing tuples with the same template together would be beneficial.

SwarmLinda describes several algorithms [19, 11] but in this paper and in the implementation of our system we concentrated on two of them: tuple distribution and tuple retrieval.

3.1 Tuple Distribution

Tuple distribution relates primarily to the primitive `out` as this is LINDA's way of allowing a process to store information. A reasonable analogy for the way tuple distribution takes place in SwarmLinda is the way vegetation occurs in nature. Everyone who has traveled have noticed that different regions seem to have concentrations of different kinds of vegetation. Seen from the space, the Earth consists of a set of patches of specific kinds of vegetation.

Analogously, tuple distribution in SwarmLinda attempts to distribute tuples based on their type. So that similar tuples stay closer to each other – in swarm terms this is equivalent to brood sorting in termite or ant colonies. To achieve this abstraction we see the network of SwarmLinda nodes as the terrain in which tuple-ants roam. These ants carry tuples (generated by `out` primitives) and decide at each hop in the network if they should drop the tuple or not. The decision is made stochastically but weighted by the amount of similar tuples around the ant's current location – similarity defined in terms of the tuples' templates.

More specifically, the implementation works as below:

1. Upon the execution of an `out` request by a process, the node to which the process made the request increases the scent for that type of tuple in its own scent table and then determines whether it should store the tuple. The choice depends on the amount of scent for the tuple type the node is currently dealing with – the higher the concentration of scent, the higher the chances the tuple will be stored at the current node.

2. If it decides not to store it, it scans the scents of its neighbors for that type of tuple and determines which neighbor the tuple should be sent to.
3. It then sends the tuple to the chosen node and the steps above are repeated in receiving node.

For the above to work, there should be a guarantee that the tuple will eventually be stored. This is achieved by having a aging mechanism associated with the tuple-ant. At each step, the tuple-ant gets more "tired". The more tired an ant is, the more likely it is to store the tuple even if the scents around the location are not similar to the tuple it is carrying.

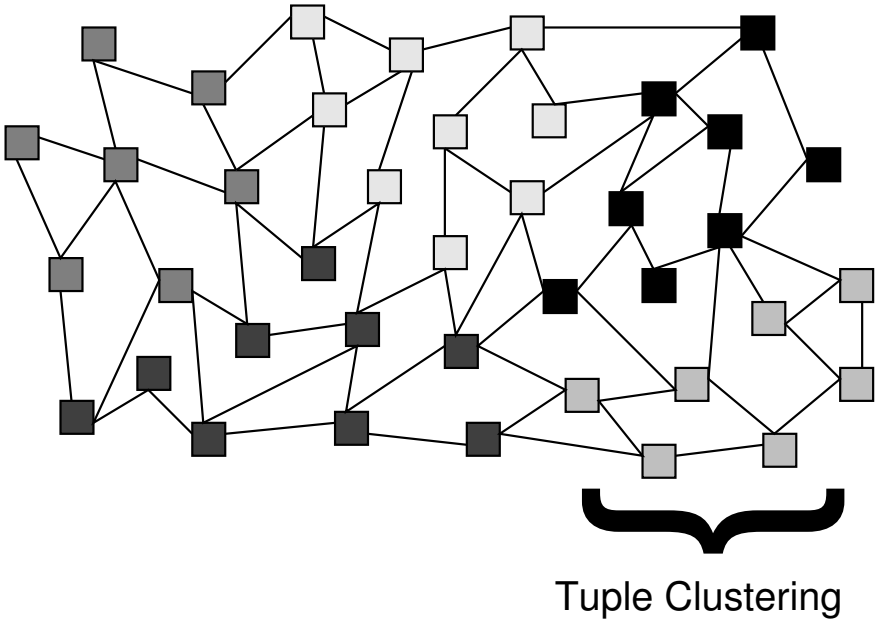

Fig. 1. A P2P network of SwarmLinda nodes storing tuples of five different kinds. Nodes storing a particular type of tuple form clusters.

Ideally one should be able to have some clustering of tuples at the node as depicted in Figure 1. This figure depicts a scenario where five types of tuples are clustered across all nodes. It is worth pointing out that the scenario depicted is only a realistic assumption if the SwarmLinda is augmented with the other algorithms described in [19], which relates to tuple movement *after* they have been stored. This and other algorithms are not discussed here because they have not yet been implemented (see Section 5 for future work discussion).

Also, Figure 1 does not convey the idea that SwarmLinda nodes may contain more than one type of tuple. In fact, this is likely to be the case – clusters do overlap.

3.2 Tuple Retrieval

Tuple retrieval takes place when processes make requests for tuples – made by executing the primitives `in` and `rd`. In swarm terms, requests are seen as ants looking for food in a terrain formed by the network of SwarmLinda nodes. A template-ant carries the request from server to server testing at each step for tuples that match the template.

Template-ants look for tuples in the current node and also sense the available scents of the node's neighbors to make a decision. The decision is very simple. If the current node has a matching tuple, the tuple is returned to the requesting process, and if the current node does not have such a tuple, the template-ant uses the information about the scents to make a stochastic decision as to where to hop next.

More specifically, the search for tuples is SwarmLinda occurs as described below:

1. Upon the execution of an `in` or `rd` primitive by a process, the node to which the process is connected (its home node) determines whether a matching tuple exists locally.
2. If none can be found, the scents for that particular tuple type (defined by the template the ant is carrying) is scanned from the neighbors. This information is used to decide the fitness value of each of the neighbors. The neighbor with best fitness value (plus or minus some random factor) is chosen as the next destination for the template-ant.
3. The node chosen and now receiving the request handles it in the same way (starting from step 1).

The life of an template-ant is limited to ensure that it does not seek for tuples that have not yet been produced. After each unsuccessful step, the template-ant may decide to stop searching and take one of the following steps:

1. Sleep for some time and then continue. This is a pure limitation of activity. If the ant has reached an area where no matching tuples have been produced for a long time, the ant will have a hard time to get out of that location.
2. Die and be reborn after some time at the location the search started.
3. Materialize in some other (random) location in the network and continue to search for tuples. This will perhaps lead the ant to find a match but will not lead to an optimal trail from the original location to the tuple's location. However, the marked trail may be used by other template-ants that operate in that region and can help find optimal trails from their origins to tuples.
4. The template-ant simply stops – they become quiescent until a tuple-ant finds it.

Which action is taken depends on the age of the ant. After an ant has slept several times, it then tries a rebirth. After some re-births, it decides to rematerialize elsewhere. Finally it may decide to become quiescent. The last action (4) is SwarmLinda's equivalent to blocking.

4 Implementation

The implementation of SwarmLinda was written in Java. Java is a good choice for these kind of implementations due to its multi-platform nature. Java is also network-ready which makes the implementation of communication between nodes more easily handled. SwarmLinda represents tuples as Extensible Markup Language (XML) documents. In order to handle well XML parsing we have used Apache Xerces XML Parser [1]. The XML document representation for tuples was chosen for it is platform and language independent and because XML is an open standard.

4.1 Network Topology

The implemented network topology can be represented as an undirected acyclic graph, meaning that there is only one path between any two nodes. This basic solution will be changed when the other SwarmLinda algorithms are implemented. The strategy described in the original SwarmLinda paper [11] takes into consideration concepts of self-organization to make neighbor list as dynamic as the system itself.

The SwarmLinda nodes communicated solely via UDP/IP. Although UDP does not guarantee delivery, we opted to use it since it allows for unbalanced neighbor lists. That is, in the end we want the topology to resemble a directed graph so that a node A being a neighbor of B does *not* mean that B is a neighbor of A. Furthermore the abstraction of packets being routed from node to node as it is done in UDP, closely resembles the ant-based model we are following.

The other form of communication used in the implementation is TCP-based connections. These are used to connect SwarmLinda processes to nodes. It is thought that processes need to be given the guarantee that a SwarmLinda node has at least received its request. More importantly, some important characteristics of good LINDA models such as `out`-ordering [4] requires that the ordering of execution of primitives in a process should be the same order these primitives arrive at a node.

One important feature of SwarmLinda is its openness. New nodes can be added dynamically, in fact this is the only way to connect a node; when a node starts, it is either stand-alone or it connects to an existing node in an already formed network.

4.2 Separation of Concerns

The original LINDA model [7] assumed the existence of only one tuple space that was global in nature. Later on, an extension to allow other tuple spaces to be created has been proposed as an improvement to the original model [8]. This extension has been so well received that today it is not uncommon to read authors referring to this extension as *the* standard LINDA.

In SwarmLinda we maintain the concept of multiple tuple spaces but only as an interface to the processes. Internally, multiple tuple spaces are just a differential in the template. That is, the handle of the tuple space (its unique name) is part of the representation of the tuple itself.

SwarmLinda does not impose any distribution mechanism and lets the tuples self-organize. The removal of tuple spaces as physical locations guarantees that processes cannot directly impose a distribution mechanism onto the implementation. Basically, tuples from the same tuple space in SwarmLinda may be physically stored in separate nodes.

4.3 Architecture

SwarmLinda is comprised of three main components, a Process API/Manager, a Node Manager and an Information Manager. These are written as components and take advantage of the benefits of component-based design such as uncoupling.

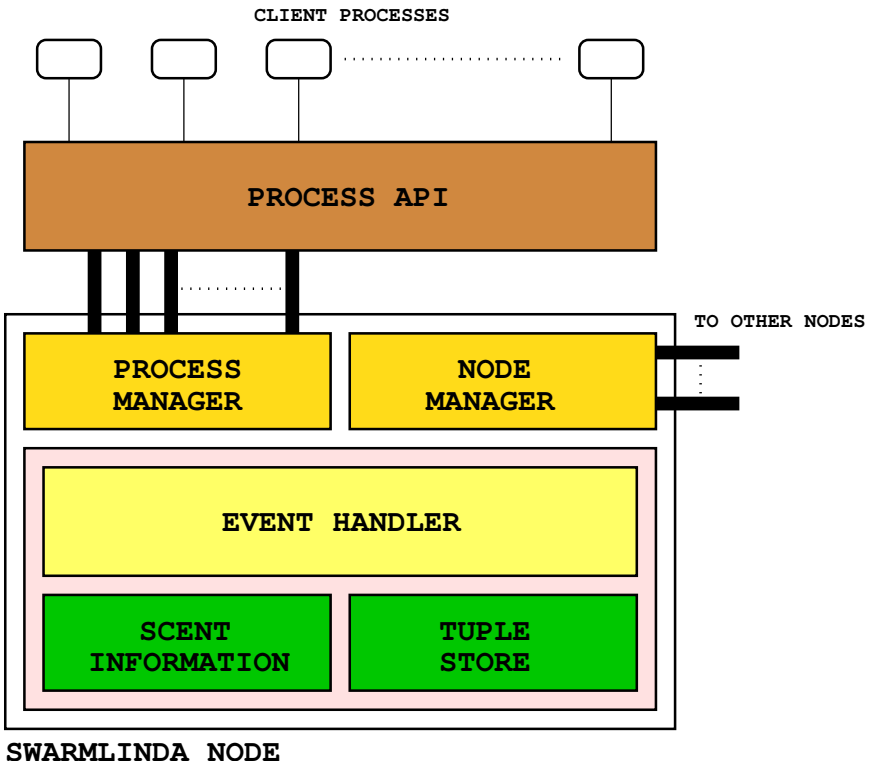

Fig. 2. SwarmLinda Architecture

Figure 2 shows the architecture of the current implementation. The parts are described below:

Process Manager: The Process API is used by processes that wish to communicate via the SwarmLinda System. Each node has a Process Manager which formats information, and passes it on to the Event Handler. It also provides an interface by which messages can be sent to the client processes.

Node Manager: The Node Manager is similar to the Process Manager, in that it deals with other nodes (peers) in the network. It is primarily concerned with the routing of information between the nodes.

Information Manager: The real interesting part of the program is the Information Manager, because this is where all decisions concerning tuple management are made. It has three sub-components, an Event Handler, a Scent Information component and a Tuple Storage Space. The Tuple Store is different from a tuple space, in that a tuple space is an abstraction used by a process to implement separation of concerns and a Tuple Store is a data structure used to store any tuple on a SwarmLinda node.

Both the Process and Node Managers are fairly trivial and have straightforward implementations. The reason for doing it this way is that it allows for a consistent interface for both process and node communications. This makes the Information Manager less complex because it only has to communicate with one interface.

All requests are interpreted and handled by the Event Handler which in turn decides whether the other sub-components in the Information manager need to be involved.

The Scent Information component is responsible for maintaining the neighbor list and the scent table for each neighbor as well as the scent table for the node itself. Neighbors' scents are maintained locally in each node to optimize decision making, so that instead of requesting scents from your neighbors before you decide, you make your choice based on previous scents. After a decision has been made, a request is sent to all neighboring node to send their updated scent for that tuple type. Updates are sent only if needed.

The Tuple Store is the data structure used to store tuples on a particular node. It is also responsible for the pattern matching of tuples to templates when retrieving tuples.

To better understand the flow of information in a SwarmLinda node, let us describe the procedure for the execution of an `out`:

1. Process calls the API's `out` function and the Process API formats and transmits request to the node.
2. The Process Manager at the node receives the request and transfers it to the Event Handler.
3. The Event Handler asks the Scent Information component whether the tuple should be stored here (its comfort level).

4. If the answer is yes, then it uses the Tuple Store to store the tuple, otherwise, it asks the Scent Information component which other node the tuple should be sent to – based on their scent level.
5. The same procedure is executed on the next nodes, with the exception of the request being received by the Node Manager and not the Process Manager.

4.4 SwarmLinda Process API

The Process API is implemented in Java, but does not use any of the native serialization provided by the Java API, thus it becomes easier to have nodes that are not Java-based. There are three main abstractions in the Process API, namely, a Tuple, a Template and a Handle.

Tuple: A Tuple is implemented as a XML document, containing tags to indicate Fields and their order encalsulated in a Java object. A Field can be any well-formed XML string, allowing complex data to be stored without adding complexity to the SwarmLinda System. Once a tuple is created, fields are added one at a time it the desired order. There are functions to automatically handle deletion and addition of native types such as integers and floating-point numbers in a tuple.

Template: A Template extends the Tuple class, and therefore its representation in XML is similar. The difference is that Templates have Formals and these are represented by the tag `Formal`, and the text contained within is the string representation of the type. This allows the application developer the freedom to take advantage of of different object types.

Handle: A Handle is an object that represents a Tuple Space from the process' perspective. This is used to implement the separation of concerns concept.

Figures 3 and 4 depict two simple examples of processes implemented in SwarmLinda. The example is trivial but demonstrates some of the syntax of our system:

```
SwarmLindaAPI sla=new
   SwarmLindaAPI(‘‘cs.fit.edu’’, 5555);

Tuple t=new Tuple();
t.add(‘‘This is a string’’);
t.add(7);
t.addXML(‘‘<NewType>A new type</NewType>’’);
sla.out(t);

sla.close();
```

Fig. 3. Process stores a tuple in the global tuple space.

Figure 3 is an example of a simple producer of tuples. The process first connects to a node at `cs.fit.edu` that is listening at port `5555`. Once the connection is established, the process creates a tuple using its native string and integer functions and also demonstrates how you would create your own type. The effects of executing this code would be to store a tuple with of the form `[‘‘this is a string’’,7,A new type]`. Last, the process terminates the connection to the SwarmLinda node.

```
SwarmLindaAPI sla=new
   SwarmLindaAPI(‘‘cs.fit.edu’’, 5555);

Template t=new Template();
t.add(‘‘This is a string’’);
t.addFormal(‘‘int’’);
t.addFormal(‘‘NewType’’);
Tuple retreived=sla.in(t);

sla.close();
```

Fig. 4. A process removes a tuple from the global tuple space.

Figure 4 is an example of a simple consumer. It shows how the API allows Template creation and the various ways in which pattern matching can occur. This template matches the tuple stored by the process in Figure 3. Note that the third entry is a user-defined type and that there is no difference between that and specifying a native type, like `int`. In fact, the native types are just additional function overloads that provides a convenience to the developer.

5 Conclusion

SwarmLinda has been well specified but its implementation is still in its infancy. For the next release we will concentrate on implementing the abstraction of multiple tuples spaces. Most importantly, in order for SwarmLinda to demonstrate its capability as a self-organized system we need to deal with the evaporation of scents. Currently scents can become stronger but not weaker. Swarm-based algorithms assume both positive (already implemented) and negative feedback on the scents.

Further improvements would include having the ants get tired and therefore stop at a location as described in Section 3. On distribution, this would just mean storing the tuple, but on retrieval, the template would stay at a particular node for some time. Various other techniques have been proposed in the original paper and will be considered in future releases. These techniques include the restart of a template-ant after a period of time in sleep mode, or have the template-ant

warp to a random node on the network and restart there, or even just stay there and wait for a tuple to find it.

It is the last option that intrigues us the most, because it is possible to have two scents instead of just one. One scent would represent the location of a tuple, and the other the location of a template. The appropriate scent would therefore be updated on the moving of tuples and templates, and the decision on where tuples or templates would be slightly different: for tuples, the scent of a template would be considered more heavily that the scent of a tuple and the opposite for templates.

Another idea is to have tuples reorganize themselves based on how comfortable they are in their surroundings. This comfort level would be based on the number of similar tuples located on that server. If the tuple decided that it was not comfortable at that server, it would again be transferred to another server based on the tuple distribution method.

In this paper we have described the project developed for a course on Swarm Intelligence given at the Florida Institute of Technology. The implementation of this first system was done primarily by the first author of this paper. The implementation as well as other articles related to the SwarmLinda project can be found at the project website [10].

Acknowledgements

Our thanks to Andy Tinkham for his assistance with the development of SwarmLinda, and to anyone we may have shown this to for their insight and enthusiasm.

References

1. Apache Software Foundation. The Apache XML Project. `http://xml.apache.org/`. Xerces2 Java Parser 2.6.0 Release.
2. E. Bonebeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford Press, 1999.
3. M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.
4. A. Douglas, A. Wood, and A. Rowstron. LINDA Implementation Revisited. In P. Nixon, editor, *Proc. of the 18th World Occam and Transputer User Group*, pages 125–138. IOS Press, April 1995.
5. E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns and Practice*. Addison-Wesley, Reading, MA, USA, 1999. The Jini Technology Series.
6. G. T. Ltd. Gigaspaces platform. White Paper, February 2002. `http://www.gigaspaces.com`.
7. D. Gelernter. Generative Communication in Linda. *ACM Trans. Prog. Lang. Syst.*, 7(1):80–112, 1985.
8. D. Gelernter. Multiple tuple spaces in Linda. In E. Odijk, M. Rem, and J.-C. Syre, editors, *PARLE '89, Vol. II: Parallel Languages*, LNCS 366, pages 20–27, 1989.
9. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

10. R. Menezes and R. Tolksdorf. Swarmlinda website. http://cs.fit.edu/~rmenezes/SwarmLinda.
11. R. Menezes and R. Tolksdorf. A new approach to scalable linda-systems based on swarms. In *Proceedings of ACM SAC 2003*, pages 375–379, 2003.
12. R. Menezes and R. Tolksdorf. A new approach to scalable linda-systems based on swarms (extended version). Technical Report CS-2003-04, Florida Institute of Technology, Department of Computer Sciences, 2003. http://www.cs.fit.edu/~tr/cs-2003-04.pdf.
13. A. Omicini and F. Zambonelli. TuCSoN: a coordination model for mobile information agents. In D. G. Schwartz, M. Divitini, and T. Brasethvik, editors, *1st International Workshop on Innovative Internet Information Systems (IIIS'98)*, pages 177–187, Pisa, Italy, 8-9 June 1998. IDI – NTNU, Trondheim (Norway).
14. H. Parunak. "go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.
15. G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering*, pages 368–377, 1999.
16. M. Resnick. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.
17. A. Rowstron. WCL: A co-ordination language for geographically distributed agents. *World Wide Web*, 1(3):167–179, 1998.
18. J. Snyder and R. Menezes. Using Logical Operators as an Extended Coordination Mechanism in Linda. In F. Arbab and C. Talcott, editors, *Proceedings of the 5th International Conference, COORDINATION 2002*, number 2315 in Lecture Notes in Computer Science, pages 317–331, York, UK, April 2002. Springer-Verlag.
19. R. Tolksdorf and R. Menezes. Using swarm intelligence in linda systems. In A. Omicini, P. Petta, and J. Pitt, editors, *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World ESAW'03*, 2003. To appear.
20. P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.