A New Approach to Scalable Linda-systems Based on Swarms

(EXTENDED VERSION)

Ronaldo Menezes
Florida Institute of Technology
Department of Computer Sciences
150 W. University Blvd.
Melbourne, FL 32907, USA
rmenezes@cs.fit.edu

Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik Networkbased
Informationsystems Takustrasse 9
D-14195 Berlin, Germany
research@robert-tolksdorf.de

ABSTRACT

Natural forming multi-agent systems (aka Swarms) have the ability to grow to enormous sizes without requiring any of the agents to oversee the entire system. The success of these systems 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 look at abstractions in the field of swarms and study their applicability in the context of coordination systems. In particular, we focus on the problematic issue of scalability of Linda systems. The purpose of this work is to look at abstractions yielded from observations of swarms and the way they are organized, and demonstrate how these abstractions may be used to implement a scalable tuple distribution mechanism in a Linda system – to be named *SwarmLinda*.

1. INTRODUCTION

In the past 20 years coordination models, and in particular tuple-space-based models such as Linda, have proven to be quite successful in tackling the intricacies of medium-to-large-scale open systems. Currently, the tuple-space model is incorporated in commercial middleware platforms such as Jini [1] and GigaSpaces [10].

Yet, the coordination community is aware that Linda systems may not scale well due to the amount of information transmitted between the entities involved in this model. As the number of actors increases, the communication overhead becomes prohibitive. One of the reasons for the somewhat poor scalability of Linda systems may be the fact that the design of these systems still inherit ideas from early Linda systems [11, 12]. These implementations were focused on parallel computing and not on open large scale computing. When trying to use the communication abstraction advocated by Linda in the context of large scale distributed systems one is faced with a not-so-easy-to-solve scalability issue: communication overhead.

Natural forming multi-agent systems, such as swarms of bees and flies, schools of fish, and packs of wolves, are notorious for their organization (coordination) and also for their ability to grow to enormous sizes – some ant colonies are known to span thousands of miles and to contain millions of ants. Their activities are based on simple rules that can be easily implemented in computer programs. Their interaction translates (in computer science terms) into local communications. Abstractions taken from these areas have been used extensively in areas such as optimization of NP-hard problems [9] and implementation of network routing algorithms [3], and normally yield simple and efficient solutions.

Our work investigates the use of swarm-intelligence techniques and observations in the context of the tuple space systems (in particular the Linda coordination system). Our goal is to look at the scalability problem and study how to improve the current scenario using techniques adapted from models originating from biological collective organisms such as ant colonies and termite molds.

This paper is divided as follows. Section 2 describes the scalability problem in Linda-based systems. In Section 3 the standard approaches for implementing distributed Linda systems are covered and their problems summarized. Section 4 goes over the main concepts of swarms systems and how these should be used in a SwarmLinda. Section 5 shows several algorithms that may be implemented to make a SwarmLinda more scalable and adaptive. Section 6 summarizes the advantages of a SwarmLinda over traditional approaches.

2. SCALABILITY OF LINDA SYSTEMS

Scalability is today the *sine qua non* of efficient distributed systems. It is not uncommon for distributed systems to make use of a large number of active entities. In fact, we can foresee an increase on the number of active entities in the future and consequently an increase of communication in these systems. Given this trend, we can easily say that scalability is one of the main challenges for the future of distributed systems.

Scalability is a well studied topic in the context of tuple-space systems such as Linda. Studies range from theoretical [15] to implementations of tuple-space systems that claim to be scalable [22, 18], to studies on the suitability of these systems to wide-area computing [14] based on Cardelli's claims [5].

Theoretical works tend to formalize the concept of tuple spaces and use the formalism as a way to show how the data should be distributed. For instance, Obreiter and Gräf [15] argue that scalability can be achieved by organizing tuples in servers based on the structure of the tuples. This is generally used in implementations via the mechanism of hash codes associated with tuples (and the use of this hash code to decide the physical location of these tuples). This technique attempts to improve scalability of tuple space systems by improving the tuple matching mechanism (how fast tuples can be found).

In the practical arena, Rowstron did several works attempting to make Linda systems more scalable [19, 18]. His approach is based on the idea of configuring tuple spaces hierarchically and classifying the spaces into local and global. Rowstron views tuple spaces as indivisible entities although some argue that tuples (objects stored in tuple spaces) must be the focus of distribution. One could easily argue that the granularity of Rowstron's systems is too coarse for real-world large scale systems.

As an illustration of the communication overhead in existing Linda implementations, consider the problem of retrieving a tuple (based on a given template) from a set of remote servers. A common implementation technique is to ask a set of servers for matching tuples by some multicast. Each server searches, matches and locks the tuples while it offers them to the requestor. This scheme establishes a state "tuple under request" which is global for the set of servers and the requestor. This approach will suffer from severe performance problems if the number of nodes searched or the number of requesting nodes increases by magnitudes.

The drawback of the approaches such as the ones above is that they are all based on the same concept: they are data-oriented. They assume that improved scalability depends solely on the way the data is distributed and not on how search for a tuple is performed. Although better scalability can be achieved using data-oriented techniques, it is a process of dubious reliability in the long run. Additionally, data-oriented techniques do not show tolerance to changes in the environment such as server failures and addition of new servers on-the-fly, to name but a few.

In order to be of use to Linda systems, data-oriented techniques need to be augmented with other concepts to help minimize the communication overhead. This paper looks away from standard techniques and go search for solutions in the field of biological sciences where natural forming multi-agent systems (aka swarms) seem to exhibit the characteristics necessary to build scalable Linda systems.

3. DISTRIBUTING LINDA

There are several aspects that need to be considered when implementing a Linda-like system. Among these, the process of distributed tuple spaces has been widely discussed by the coordination community. The literature on Linda-like systems describes a plethora of approaches for distributing tuple spaces. Several major strategies can be distinguished and evaluated against the requirements of large scale distribution.

- *Centralization* is a simple client-server distribution strategy where one specific server-machine operates the complete tuple space as in TSpaces [21]. It can be accessed by clients that are arbitrarily located somewhere on a network (see Figure 1).

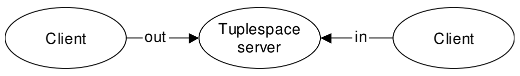

Figure 1: Centralized tuple spaces

The centralized tuple-space server has the advantage of an easy implementation that basically attaches a network interface to an otherwise non-distributed tuple-space. However, it carries all the disadvantages of any centralization of services. The tuple-space server is most likely to become a bottleneck under high load induced from a large number of active clients, it is the single point of failure in the entire system, and it does not make a fair use of the resources available over the network.

The centralized solution is clearly not the best choice for a large scale distributed Linda-like system.

- *Partitioning* of tuple spaces is a strategy in which data with common characteristics is co-located in one of a set of tuple-space servers (see Figure 2). Requests with the same characteristics are then routed towards that machine. While simple partitioning (eg. co-locating tuples with same parity) might lead to unbalanced partitions, a carefully chosen hashing function on tuples can do better [2].

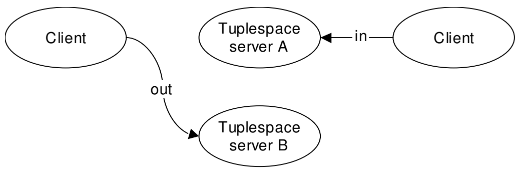

Figure 2: Partitioned tuple spaces

Partitioning has the advantage of providing a distributed management of a tuple space including concurrent execution of operations on the partitions and thus slightly relaxes the problems of centralized solutions. However, for one, it does include a centralization for certain sets of tuple. Also, the partitioning scheme handles reconfigurations very poorly. An automatic adaption of any hashing function and the induced distributed reorganization of the tuple-spaces content will be complex and costly.

The partitioning solution cannot overcome the problems of the centralized approach and cannot be applied to dynamic changes on both of the infrastructure and the applications demand.

- *Full replication* places complete and consistent copies of tuple spaces on several machines at different locations. Any addition of data has to be replicated at all nodes, a search for data can be performed locally on one machine, and any removal of data has to be replicated (see Figure 3).

Full replication distributes the load for data-searches and inherently offers some support for fault-tolerance. However, the communication costs for keeping the replicas consistent

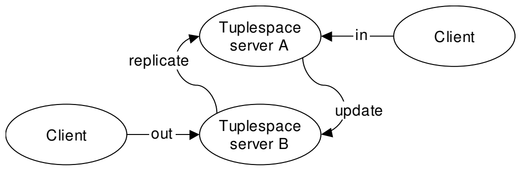

Figure 3: Full replication

on all participating nodes is high and requires locking protocols or equivalents since multiple local searches may lead to the decision to remove the same piece of data [7].

Full replication is not a good choice for large scale systems, as both adding and removing data involves operating on all replicas which certainly does not scale with the number of participating nodes.

- *Intermediate replication* has been proposed in the early of Linda [6]. The schema bases on a grid of nodes that is formed by logical intersecting "busses". Each node is part of exactly one *outbus* and one *inbus*. Emitted data is replicated on all nodes of the outbus, whereas searches are performed on the inbus. As the inbus intersects all outbusses, it provides a complete view of the tuple space (see Figure 4). By introducing simulated nodes, the number of nodes can change dynamically while retaining the virtual grid [20].

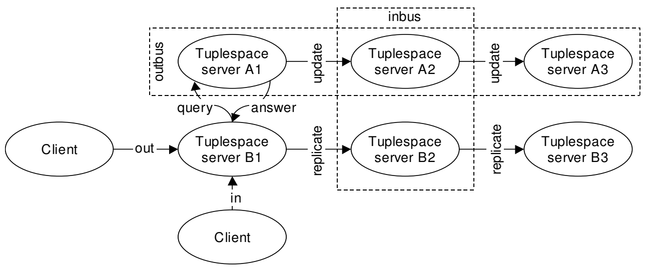

Figure 4: Intermediate replication

The intermediate replication schema allows for as many concurrent replications and searches for data as there are out- and inbusses respectively. The removal of some data, however, requires a consistent update on all nodes on the respective outbus.

Intermediate replication is only a partial solution for large scale Linda-like systems. While introducing more concurrency, it still carries scalability problems of keeping replicas consistent within an outbus. While it has the potential for dynamic reconfigurations of the numbers of in- and outbusses, the two "dimensions" used seem to be insufficient to scale.

While none of these approaches seems to be usable (without modification) in the implementation of a large scale distributed Linda-like system, the intermediate replication schema seems to offer the most flexibility. In fact, it is the most general schema, as it conceptually captures the centralized approach – a single outbus of size 1 – and the fully replication case – a single outbus of size n for n participating nodes.

Schema	Problems
Centralized	Centralization of load and failure, not scalable
Partitioning	Repartitioning hard
Full replication	Huge overhead for replicated addition and removal of data, not scalable
Intermediate replication	Overhead for coordinating replication and locking, reconfiguration hard

Table 1: Problems with known schemas

Table 1 summarizes the main drawbacks of standard distribution approaches. None of the approaches described here scale well and therefore should not be used in an implementation of a large scale Linda. Long-term solutions for this problem are long due and need to be explored. Thus SwarmLinda.

4. CONCEPTS OF A SWARMLINDA

Over the past years, new models originating in biology, such as swarms, ant colonies and termite molds, have been studied in the field of computer science [17, 4, 8, 13]. The current interest lies on using these abstractions as a technique (a meta-heuristic) for finding feasible solutions to NP-hard problems.

In these models, actors (workers) sacrifice individual goals (if any) for the benefit of the society. At the same time, these actors (birds, fish, ants, termites) act extremely decentralized without any explicit global or central control. The work is carried out by making purely local decisions and by taking actions that require only very small computations. These characteristics alone allow for the systems to scale to very large sizes – ant colonies can involve millions of individual ants and still perform highly complex tasks such as building an ant hill. This so-called *swarm intelligence* provides an interesting opportunity to rethink scalability of coordination media like tuple spaces.

We apply these principles as a radical new approach to implementing a large scale distributed coordination platform, which we call SwarmLinda. An example of such principles being used 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 servers as a two-dimensional space in which ants search for food, leaving trails to successful matches.

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

The nature of an ant-based optimization ensures that changes in the structure of a distributed application in terms of tuple-generation and -consumption are dynamically discovered. Secondly, when searching for food, the "template-ants" make only local decisions, like moving to some next server, examining the local tuple-store, etc. There is no global state that will be established by the mechanism.

4.1 Principles

A SwarmLinda has to consider a few principles that can be observed in most naturally occurring swarm systems [16]. These principles guarantee the success of the collective activity performed by the swarm and also allow for these natural occurring systems to become very large in size:

Simplicity: Natural swarm individuals are simple creatures that perform simple tasks. They do no deep reasoning and implement only a small set of simple rules. The execution of these rules in a society leads to the emergence of complex behavior. Active entities in a SwarmLinda should also obey the principle of low complexity. They should be “small” in terms of resource usage.

Dynamism: Natural swarms operate in a dynamically changing environment and are able to adapt to it. In an open distributed system, the configuration of running applications and services changes over time. If a tuple is found in a given location it doesn’t necessarily mean that other similar tuples will exist in the same location in the future. Therefore, a search heuristic for tuples should not be fixed; instead it should dynamically adapt to changes.

Locality: Natural swarm individuals observe their direct neighborhood and take decisions based on this local view. As the key to scalability in SwarmLinda, an active entity has to perform only local searches and inquire only to direct neighbors.

4.2 Abstractions

One needs to understand how a SwarmLinda should be organized in order to respect the principles just described. Standard Linda systems do not have the idea of ants or food. A SwarmLinda needs to abstract these concepts in the Linda world.

In the later part of this paper, we will base the description of a SwarmLinda 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.

Environment is the context in which the individuals work and which they observe.

State is a characteristic of the environment that can be observed and changed by individuals.

There is not necessarily a fixed mapping between the abstractions above and the concepts common to Linda implementations. Although the environment and the state are somewhat fixed, the individuals may represent different Linda entities depending on the algorithm in question. For instance, when searching for a tuple the individuals are the active templates that move on the grid of servers looking for tuples. On the other hand, when trying to be adaptive, individuals are tuples that move from location to location based on a semi-random decision process.

5. ALGORITHMS FOR A SWARMLINDA

Having described the general concepts of a SwarmLinda, we can now focus on bringing it to life by defining the concrete environment and its state, and the individuals and the rules they apply. Next we describe some algorithms that seem useful. These are taken from abstraction of natural multi-agent systems [4, 16], especially those formed by ants.

These algorithms make the core of SwarmLinda – they are the abstraction of swarm intelligence in the context of Linda. In general, these algorithms are fairly independent and should be able to be implemented without requiring the other algorithms to exist in the system.

5.1 Searching for Tuples

This algorithm is normally used by ants to find food. Ants look for food in the proximity of the ant hill. Once found, the food is brought to the ant hill and a trail is left so that other ants can know where the food can be found. The ants know the way back to the ant hill because they have a short memory of the last few steps they took and also because the ant hill has a distinctive scent that can be tracked by the ants. In a tuple space context, one could view tuples as food. The locations where the tuples are stored can be seen as the terrain while the templates are seen as ants that wander in the locations in search of tuples. The ant hill is the process that executed the operation.

The active individuals are the template-ants, the environment consists of tuple-space servers whose state is composed by the tuples stored and “scent” of different kinds of template that indicate a likelihood that matches for a template is available at a location. The scents are volatile and disappear slowly over time.

The tuple-searching ant should follow the following rules:

1. The first step is to spread the scent of process in the server it is connected to and this server’s neighborhood. This distinctive scent will be tracked by the ants on their way back to the ant hill.
2. Check for a matching tuple at the current location. If a match is found, return to the origin location and leave scent for the template matched at each step. They are able to find their way back using their short memory and tracking the distinctive scent of the process (as described above). If no match is found, check the direct neighborhood.
3. If there are no scents around the current location that fit to the template, randomly choose a direction in the grid of servers to look for a tuple.
4. If there is a scent that indicates a direction for next step (matching scent), move one step towards that scent and start over. As pointed out before, we want to guarantee adaptability in SwarmLinda. Additionally, we also want to maintain the non-determinism when searching for tuples. One way guarantee these properties is by adding a small random factor in a range of $[-\xi, \xi]$ to each scent. This enables new paths (not necessarily the path of the strongest scent) to be discovered.
5. The activity of the ant is limited to ensure that it does not seek for tuples that have not yet been produced. After each unsuccessful step without a match, the ant stops its search with a probability of γ. This factor is 0 in the beginning and increased by some Γ with each unsuccessful step. Γ itself also increases over time. When the ant decides to stop searching, it takes one of three actions:

- 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. The sleeping would allow sometime for the system to change and maybe get to a state where tuples can be found in that location.
- Die and be reborn after some time at its original location – where the search started.
- Materialize in some other random location and continue to search for tuples. This will perhaps lead the ant to a find a match but will not lead to an optimal trail from the original location to the tuple found at such a distant place. However, the trail from the randomly chosen location to the tuple is marked for the other template-ants that operate in that region and can help find optimal trails from their origins to tuples. In sum, this method might be acceptable as a last resort.

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 rebirths, it decides to rematerialize elsewhere.

The result of the above is the emergence of application specific paths between tuple producers and consumers. Given that scents are volatile and become less strong with time, the paths found can dynamically adapt to changes in the system – when consumers or producers join, leave or move within the system.

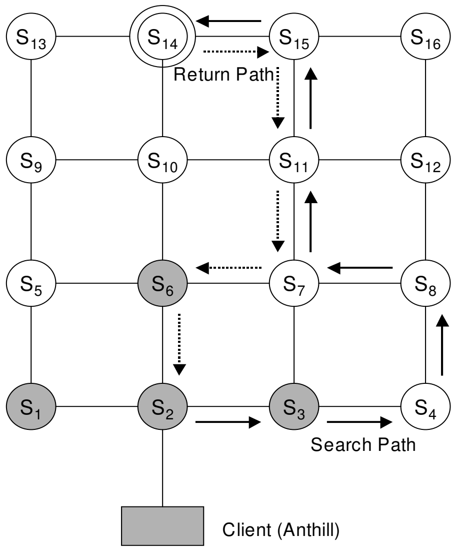

Figure 5: Searching for a tuple

Figure 5 shows an example of the search algorithm. A client connected to server S_2 spreads its scent on servers S_1, S_2, S_3, S_6. Later, a template-ant goes searching for a tuple. For the purposes of this example let us assume the template-ant can remember the last three steps it took. The template-ant wanders in search for a tuple making a decision at each server. After a few steps the template-ant finds a matching tuple in server S_{14} – the path it took until it found the tuple was [S_2, S_3, S_4, S_8, S_7, S_{11}, S_{15}, S_{14}]. After the tuple is found the template-ant uses its short memory to return to the ant hill. The first three steps returning are taken based on what is in memory: [S_{15}, S_{11}, S_7]. After this the template-ant tries to track the scent of the ant-hill. In S_7 the template-ant is influenced by such a scent and moves to S_6, S_2, and finally back to the client. Observe that the return path is not necessarily the path it took to find the tuple. In the end, the return path [S_2, S_6, S_7, S_{11}, S_{15}, S_{14}] was marked with the scent for that particular template.

Compare this approach with one standard mechanism to find distributed tuples: hashing. Tuples are normally searched based on a hash function that takes the template as the input and generate a location where the tuple can be found as the output. Hashing is definitely fast but unfortunately not very adaptive. The determinism that exist in hash functions forces tuples with the same template to *always* be placed in the same location no matter the size of the system, thus causing bottleneck problems if tuples matching such template are in demand in the system.

Although in a SwarmLinda tuples matching the same template would *tend* to stay together, this is not necessarily true in all cases. If such tuples are being produced in locations that are far enough from each other the tuples will remain separate and create clusters across all the system. This minimizes the creation of bottlenecks when tuples of a certain template are required by several processes. As the search will start from various locations, tuples will tend to be retrieved from the closest cluster from the source of the search.

Another problem with hashing approaches is that they are not fault tolerant – if a tuple is being hashed to a location, that location is expected to be working. Hashing may be made fault-tolerant (based on conflict resolutions techniques) but its implementation is normally cumbersome, its use in practice may be very expensive, and its effectiveness doubtful. For instance, one could use a re-hash technique to attempt to search for a tuple elsewhere when the first hash function failed. This is very expensive because it doubles the number of messages in the system for each tuple being searched. From the point of view of swarm techniques, failures are just another change in the environment. This would be like ants trying to search for food in a food supply that was suddenly destroyed. Surely this is not a problem and the ants will only starve if food cannot be found.

5.2 Distribution Mechanism

Another area of SwarmLinda where abstraction from natural multiagent systems can be used is in the distribution of tuples amongst the servers. Historically, tuples have been distributed using various *static* mechanisms (as described in Section 3). In SwarmLinda the partitioning of the tuple space would be dynamic using the concept of brood sorting used by ants.

Ants are able to sort different kinds of things they keep in the ant hill such as food, larvae, eggs, etc. In an ant hill these are normally sorted by type. That is, they are grouped by type in one place and kept separate from other things. More importantly, ants do this process in spite of the amount of each type, thus being very scalable.

The individuals that operate here are tuple-ants in contrast to the algorithm of the previous section. The environment is unchanged – it remains as a network of servers. Their state is the set of tuples stored thus far.

A SwarmLinda implementation may use brood sorting as below in the process of tuple distribution. One could see tuples being grouped based on their template which will lead to the formation of clusters of tuples. In this process, tuples are the food and the ant is the active process representing the out primitive:

1. Upon the execution of an out primitive, start visiting the servers.
2. Observe the kind of tuples (the template they match) the servers are storing. Each out process should have a limited memory so it doesn't remember the information about the entire "grid" of servers but only the last few – this guarantees that the decision of a process is based on local information.
3. Store the tuple in the server if nearby servers store tuples matching the same template. Again this decision also considers a small random factor $[-\xi, \xi]$.
4. If nearby servers do not contain similar tuples, randomly choose (using the random factor) whether to drop or continue to carry the tuple to the next step (to another server).

In order to guarantee that the steps above work well, certain condition must be satisfied. The out process should be able to store the tuple eventually. For each time the process decides *not* to store the tuple, the random factor will tend to ξ. This increases the chance of storing the tuple in the next step. Also the likelihood to store the tuple is also calculated stochastically based on the kinds of objects in memory – if most of the objects in memory are of the same kind as the one being carried out the likelihood to store the tuple becomes very high.

The power of this approach shows when it is compared with the partitioning scheme as described in Section 3. Similar to the case for searching tuples, partitioning is based primarily in the use of a hash function to decide where the tuple should be placed. This standard technique is far from being able to cope with failures and changes in the application behavior. Failures in certain servers may be fatal to the system while changes in application behavior may force changes in the actual hash function being used.

The approach described in this subsection is able to improve the availability of the system without having to count on costly techniques such a replication of data. In the ant-based approach described above, there are no assumptions about the behavior of applications, there are no pre-defined distribution schema, there are no special scenarios implemented to deal with failures in the server.

Scalability is also improved. Similar to the case of server failures, hashing techniques are not very simple to modify to consider servers added on-the-fly. In SwarmLinda, servers can be added at any point and the ants will make sure that these servers are explored and used in storing tuples.

Another fact that improves scalability is the limitation in the memory of the ant-tuples. This forces them to make decision based on local information *only*. In practice, this may account for having the out-process having to communicate to servers only in its neighborhood. The use of local communication in the entire scheme improves the scalability of the system.

5.3 Dealing with Openness

Openness is known to be one of the main challenges in distributed systems. In this arena, the ability of a system to deal with changes can be a great asset. For instance, in open Linda systems the need for tuples of specific formats (templates) can change greatly overtime.

Swarm systems are very adaptive and can rapidly respond to changes in the environment. Ants for instance, can move the ant hill to another location if they find themselves in danger.

To enable a SwarmLinda to show such a behavior for collections of similar tuples, we again use tuple-ants as the individuals. The environment is again a terrain of servers that has scents as the state.

Ants find the ant hill based on a particular scent unique to the ant hill. This scent is what keeps the ants together in the ant hill. For a SwarmLinda, we want tuples matching the same template to be kept together (as described in Section 5.2) but we do not want them to be fixed to a given location. Instead, we want them to dynamically adapt to changes.

There is a function $Sc : T \to S$ on templates and tuples. There is a relation $C : S \times S$ on scent that defines similarity of scent. When template te and tuple tu match, then $Sc(te), Sc(tu) \in C$.

1. A new tuple-ant that carries a tuple tu emits $Sc(tu)$ at its origin. A new template-ant that carries a template te emits $Sc(te)$ at its origin.
2. Template-ants remain at that position and never move.
3. Tuple-ants sense their environment for a scent similar – as given by C – to $Sc(tu)$. If there is such, then other template- or tuple-ants are around.
4. Based on the strength of the detected scent plus the small random factor $[-\xi, \xi]$, the tuple-ant decides to move into that direction or to stay where it is.

The above causes tuples to stay closer to where other similar tuples are needed or are being produced (based on the number of in and out primitives executed) even if this consists of migrating from one server to another. This would also have an effect on the distribution mechanism explained in Section 5.2. When a tuple is being stored the scent left by previous in and out primitives should also be considered when deciding to drop the tuple in the current server or to keep "walking" through the terrain of servers searching for a good place to drop the tuple.

5.4 Balancing tuple- and template movement

In the preceding algorithms, we always identified either the tuple-ants or the template-ants as individuals that move and perform a continued search. In this section we describe an intermediate approach where ants can be both tuples and templates. Every tuple- and template ant decides after its birth whether it goes out to other servers seeking matches or stays at its origin until it is found by some other ant.

Consider an application where one node consumes a lot of tuples that are generated on other nodes. If trails from the producers to the consumer are found – and these can be found by programming a tuple-ant with the algorithm from Section 5.1 – it makes no sense to have the consumer start template-ants that seek the consumers. Based on the system history (of scents) it is known where a consumer is and what the path is, so tuple-ants should be started there while the template-ants at the consumer should remain stationary and basically wait for a matching tuple-ant to appear. But if the former consumer starts to be a producer after the calculation of some results, it might become reasonable to start template-ants from the former producers to reach out for the result.

Our algorithm should lead to a dynamic balance between active and passive ants that takes into account the current producer/consumer configuration in the system.

For the algorithm, the individuals are tuple- and template-ants. The environment is still the terrain of servers. The state at each location includes two scents: One scent indicates whether the location is visited successfully by other ants – it is an attraction – or not – it is an outsider. Successful means that the visiting ant found a match at this location. We call this the visitor scent. The second scent, the producer-consumer scent ranges over $[-\phi, \phi]$. Positive values indicate that the matches that took place were such that a visiting template-ant retrieved a tuple from that location – showing that the location is a producer of information. If the scent is negative, it indicates that visiting tuple-ants were matched with a template at that location – the location is a consumer of tuples.

Tuple- and template-ants follow the algorithms from Section 5.1 to find matching templates resp. tuples. If an tuple-ant finds a match, it neutralizes a bit of producer-consumer scent at the location. When a template-ant finds a matching tuple, it adds a bit of this scent at the location. Both kinds of ants leave a bit of visitor scent in the case of success.

When a new ant is born, it will either be a tuple- or a template-ant depending on the kind of operation requested. A new tuple-ant emits a bit of producer-consumer scent at the location of its birth, a template-ant neutralizes some.

These ants can behave in two different ways: Either they are active and move around following the search algorithms as described above, or they are passive and remain at their origin to be found by others.

The further fate of a new ant depends on the current state of the location where they are born. This state distinguishes producing and consuming locations and whether the location is attractive for visitors. The following table shows how new ants behave based on these two characteristics:

	Producer	Consumer
Attraction	Passive tuple-ant Passive/active template-ant	Active/passive tuple-ant Passive template-ant
Outsider	Active tuple-ant Passive template-ant	Passive tuple-ant Active template-ant

If a producer is visited by many ants, there is no need to send out tuple-ants. Template-ants can be passive too as many visitors satisfy them, but also active to establish a global balance. The ratio passive/active could be adjusted.

For an attractive consumer, template-ants can remain passive. Tuple-ants might be active with the same argument on a global balance.

If a producer is not visited by many other ants, it will send out its tuples-ants to find matches. Its template-ants can remain passive as there are not many.

If a consumer is not visited by many other ants, it will send out active template-ants to find matches and generate passive tuple-ants to attract other active template-ants to improve the chance of becoming an attraction.

The algorithm can be compared to the intermediate replication scheme described in Section 3. There, an in leads to a broadcast of the template on the inbus and to a search for a matching tuple. An out leads to the replication of the tuple on the outbus where the local lists of waiting tuples are then searched. This seems to resemble the idea of having tuple-ants and template-ants go out and seek for matches. However, the in- and outbusses in intermediate replication are usually very static. In the SwarmLinda algorithm the balance between tuple- and template-ants is highly dynamic and adapts to the current behavior of the running applications.

6. EVALUATION

The algorithms described in Section 5 demonstrate the power of swarm abstractions in the context of Linda systems. They have the potential to make a SwarmLinda more efficient than standard Linda implementations. In particular, SwarmLinda appears to be more efficient dealing with (at least) the following issues:

Scalability: The decisions taken by ants are based on local observations like sensing the direct environment or asking the currently visited server for a match. State changes are also purely local like changing the scent at the current or directly surrounding locations. There are no global states like locking at a set of servers. With that, the activities of the algorithms are independent of the number of active ants. This increases the scalability of the resulting SwarmLinda with respect to the number of tuples produced and requested and the number of nodes involved in the system.

Adaptability: The decisions of ants are taken based on the current state of the system. This state is changed based on decisions of individuals in the system by leaving fresh scent at locations. The influence of prior decisions becomes smaller over time with the evaporation of old scent. The state therefore reflect the current configuration of the system in terms of where and what kind of tuples are produced and consumed. The behavior of the ants dynamically adapts to this configurations. A SwarmLinda is therefore adaptive while traditional Linda-systems behave based on static design and initialization decisions.

Fault-Tolerance: The influence of wrong decisions of ants are only temporary. Changes in scent based on wrong decisions also vanish over time with the evaporation. In a traditional Linda-system, precautions have to take care of fault-detection and correction and often even the coordination language is extended. A SwarmLinda is fault-tolerant via its inherent adaptability mechanism.

Additionally, the availability of the system is also improved. Failures in server nodes are treated by the ants as changes in the systems configuration. Based on the adaptability characteristic described above, ants should be able to deal with such failures in a very elegant way.

Load Balancing: Adaption to the current configuration of tuple production and consumption does not lead to bottlenecks in the system. As shown, the load in terms of activity of ants can also be dynamically adjusted by simple local decisions.

A SwarmLinda can load balance itself by adapting the activity performed by ants.

7. CONCLUSION

This paper describes approaches for the implementation of a SwarmLinda. The paper demonstrates that organized behavior in SwarmLinda can be implemented based on a few simple rules that mimic naturally forming multi-agents systems.

We claim that the use of swarm abstractions in an implementation of SwarmLinda would not only be simple but will improve its scalability and adaptability.

The algorithms presented here provide an excellent alternative to the standard approaches used in various Linda implementations. The algorithms follow three principles that can be observed in any swarm-based systems: simplicity, dynamism and locality.

We are currently working on a design that will include the algorithms described in Section 5. As a first step we are primarily focusing on the algorithms related to the tuple distribution mechanism and the one for searching tuples.

8. REFERENCES

[1] Ken Arnold, Ann Wollrath, Bryan O'Sullivan, Robert Scheifler, and Jim Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.

[2] Robert Bjornson. *Linda on Distributed Memory Multiprocessors*. PhD thesis, Yale University Department of Computer Science, 1992. Technical Report 931.

[3] Eric Bonabeau, Florian Henaux, Sylvain Guérin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. Routing in Telecommunications Networks with Ant-Like Agents. In Sahin Albayrak and Francisco J. Garijo, editors, *Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication Applications (IATA-98)*, volume 1437 of *LNAI*, pages 60–71, Berlin, July 4–7 1998. Springer.

[4] E. Bonebeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford Press, 1999.

[5] Luca Cardelli. Wide Area Computation. *Lecture Notes in Computer Science*, 1644:10pp, 1999.

[6] Nicholas Carriero and David Gelernter. The S/Net's Linda Kernel. *ACM Transactions on Computer Systems*, 4(2):110–129, 1986.

[7] A. Corradi, L. Leonardi, and F. Zambonelli. Strategies and Protocols for Highly Parallel Linda Servers. *Software: Practice and Experience*, 28(14), 1998.

[8] 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.

[9] Marco Dorigo and Gianni Di Caro. Ant Colony Optimization: A New Meta-Heuristic. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477. IEEE Press, 6-9 July 1999.

[10] G. T. Ltd. GigaSpaces platform. White Paper, February 2002.

[11] David Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[12] David 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.

[13] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[14] Ronaldo Menezes, Robert Tolksdorf, and Alan Wood. Coordination and Scalability. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 12, pages 299–319. Springer Verlag, 2001. ISBN 3540416137.

[15] P. Obreiter and G. Gräf. Towards scalability in tuple spaces. In *Proceedings of the 2002 Symposium on Applied Computing*, pages 344–350. ACM, March 2002.

[16] H.V.D. Parunak. "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.

[17] Mitchel Resnick. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.

[18] A. Rowstron. WCL: A co-ordination language for geographically distributed agents. *World Wide Web*, 1(3):167–179, 1998.

[19] A. Rowstron and A. Wood. Bonita: a set of tuple space primitives for distributed coordination. In *Proc. HICSS30, Sw Track*, pages 379–388, Hawaii, 1997. IEEE Computer Society Press.

[20] Robert Tolksdorf. Laura - A Service-Based Coordination Language. *Science of Computer Programming, Special issue on Coordination Models, Languages, and Applications*, 1998.

[21] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[22] F. Zambonelli, A. Corradi, and L. Leonardi. A Scalable Tuple Space Model for Structured Parallel Programming. In *Proc. of the 1995 2nd Int'l Conf. on Programming Models for Massively Parallel Computers*, October 1995.