Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems

Özalp Babaoğlu **Hein Meling** **Alberto Montresor**

Technical Report UBLCS-2001-09

November 2001 (Revised September 2002)

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

Recent Titles from the UBLCS Technical Report Series

2001-2 *Formalization, Analysis and Prototyping of Mobile Code Systems* (Ph.D. Thesis), Mascolo, C., Janaury 2001.

2001-3 *Nature-Inspired Search Techniques for Combinatorial Optimization Problems* (Ph.D. Thesis), Rossi, C., Janaury 2001.

2001-4 *Desktop 3d Interfaces for Internet Users: Efficiency and Usability Issues* (Ph.D. Thesis), Pittarello, F., January 2001.

2001-5 *An Expert System for the Evaluation of EDSS in Multiple Sclerosis*, Gaspari, M., Roveda, G., Scandellari, C., Stecchi, S., February 2001.

2001-6 *Probabilistic Information Flow in a Process Algebra*, Aldini, A., April 2001 (Revised September 2001).

2001-7 *Architecting Software Systems with Process Algebras*, Bernardo, M., Ciancarini, P., Donatiello, L., July 2001.

2001-8 *Non-determinism in Probabilistic Timed Systems with General Distributions*, Aldini, A., Bravetti, M., July 2001.

2001-9 *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*, Babaoglu, O., Meling, H., Montresor, A., November 2001 (Revised September 2002).

2002-1 *A Timed Join Calculus*, Bünzli, D. C., Laneve, C., February 2002.

2002-2 *A Process Algebraic Approach for the Analysis of Probabilistic Non-interference*, Aldini, A., Bravetti, M., Gorrieri, R., March 2002.

2002-3 *Quality of Service and Resources' Optimization in Wireless Networks with Mobile Hosts* (Ph.D Thesis), Bononi, L., March 2002.

2002-4 *Specification and Analysis of Stochastic Real-Time Systems* (Ph.D. Thesis), Bravetti, M., March 2002.

2002-5 *QoS-Adaptive Middleware Services* (Ph.D. Thesis), Ghini, V., March 2002.

2002-6 *Towards a Semantic Web for Formal Mathematics* (Ph.D. Thesis), Schena, I., March 2002.

2002-7 *Revisiting Interactive Markov Chains*, Bravetti, M., June 2002.

2002-8 *User Untraceability in the Next-Generation Internet: a Proposal*, Tortonesi, M., Davoli, R., August 2002.

2002-9 *Towards Adaptive, Resilient and Self-Organizing Peer-to-Peer Systems*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

2002-10 *Towards Self-Organizing, Self-Repairing and Resilient Large-Scale Distributed Systems*, Montresor, A., Babaoglu, O., Meling, H., September 2002.

2002-11 *Messor: Load-Balancing through a Swarm of Autonomous Agents*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems

Özalp Babaoğlu [1] **Hein Meling** [2] **Alberto Montresor** [1]

Technical Report UBLCS-2001-09

November 2001 (Revised September 2002)

Abstract

Recent peer-to-peer (P2P) systems are characterized by decentralized control, large scale and extreme dynamism of their operating environment. As such, they can be seen as instances of complex adaptive systems (CAS) *typically found in biological and social sciences. In this paper we describe Anthill, a framework to support the* design, implementation *and* evaluation *of P2P applications based on ideas such as multi-agent and evolutionary programming borrowed from CAS. An Anthill system consists of a dynamic network of peer nodes; societies of adaptive agents travel through this network, interacting with nodes and cooperating with other agents in order to solve complex problems. Anthill can be used to construct different classes of P2P services that exhibit resilience, adaptation and self-organization properties. We also describe preliminary experiences with Anthill in implementing a file sharing application.*

1. Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna (Italy), Email: {babaoglu,montresor}@CS.UniBO.IT
2. Department of Telematics, Norwegian University of Science and Technology, O.S. Bragstadsplass 2A, N-7491 Trondheim (Norway), Email: meling@item.ntnu.no

1 Introduction

Informally, *peer-to-peer* (P2P) systems are distributed systems based on the concept of resource sharing by direct exchange between *peer* nodes (i.e., nodes having the same role and responsibility). Exchanged resources include content, as in popular P2P file sharing applications [17, 8, 10], and storage capacity or CPU cycles, as in computational and storage grid systems [1, 16, 9].

Distributed computing was intended to be synonymous with P2P computing long before the term was invented, but this initial desire was subverted by the advent of client-server computing popularized by the World Wide Web. The modern use of the term P2P and distributed computing as intended by its pioneers, however, differs in several important aspects. First, P2P applications reach out to harness the outer edges of the Internet and consequently involve scales that were previously unimaginable. Second, P2P by definition, excludes any form of centralized structure, requiring control to be completely decentralized. Finally, and most importantly, the environments in which P2P applications are deployed exhibit extreme dynamism in structure, content and load. The topology of the system typically changes rapidly due to nodes voluntarily coming and going or due to involuntary events such as crashes and partitions. The load in the system may also shift rapidly from one region to another, for example, as certain files become “hot” in a file sharing system; or the computing needs of a node suddenly increase in a grid computing system.

Traditional techniques for building distributed applications are not satisfactory for dealing with the scale and dynamism that characterize modern P2P systems. For example, certain file-sharing applications [8] rely on flooding-style communication, severely limiting their scalability. Other systems require manual intervention for their configuration or tuning as their environment changes. We argue that satisfying the needs of P2P application development requires a paradigm shift that includes adaptation, resilience and self-organization as primary concerns.

In this paper, we suggest that *complex adaptive systems* (CAS) commonly used to explain the behavior of certain biological and social systems can be the basis of a programming paradigm for P2P applications. In the CAS framework, a system consists of a large number of relatively simple autonomous computing units, or *agents*. CAS typically exhibit what is called *emergent behavior*: the behavior of the agents, taken individually, may be easily understood, while the behavior of the system as a whole defies simple explanation. In other words, the interactions among agents, in spite of their simplicity, can give rise to richer and more complex patterns than those generated by single agents viewed in isolation. Examples of CAS include multicellular organisms, social insect colonies, and living ecosystems.

From a P2P perspective, CAS offer several attractive properties, including total lack of centralized control. Furthermore, the emergent behavior of CAS is highly adaptive to changing environmental conditions or unforeseen scenarios, is resilient to deviant behavior (failures) and is self-organizing towards desirable global configurations. Most importantly, these global properties are obtained without explicitly “programming” them into the individual agents.

In order to pursue these ideas, we are developing *Anthill*, a novel framework for P2P application development, based on ideas such as multi-agent systems (MAS) and evolutionary programming borrowed from CAS [18, 12]. The goals of Anthill are to provide an environment that simplifies the design and deployment of P2P systems based on these paradigms, and to provide a “testbed” for studying and experimenting with CAS-based P2P systems in order to understand their properties and evaluate their performance.

Anthill uses terminology derived from the ant colony metaphor. An Anthill distributed system is composed of a network of interconnected *nests*. Each nest is a peer entity sharing its computational and storage resources. Nests handle requests originated by local users, by generating one or more *ants* – autonomous agents that travel across the nest network trying to satisfy the request. Ants can observe their environment and perform simple local computations leading to actions based on these observations. The actions of an ant may modify the environment, as well as the ant’s location within the environment. In Anthill, emergent behavior manifests itself as *swarm intelligence* whereby the collection of simple ants of limited individual capabilities achieves “intelligent” collective behavior [2].

The multi-agent paradigm promoted by Anthill is particularly suited for applications that are to be deployed in highly dynamic environments, subject to incomplete and imprecise information [2]. Artificial ant colonies, which mimic real ant colonies that are natural instances of MAS, are known to be capable to solve complex optimization problems including those arising in communication networks [3].

The Anthill API supports P2P application development and deployment through the provision of a set of services offered by nests, such as storage management, communication and topology management, and ant scheduling. Developers can build P2P applications simply by defining the structure of the P2P system and designing appropriate ant algorithms using the Anthill API for solving the application problem. The services provided by Anthill free the developer from considering low-level details such as communication, security and scheduling strategies.

Anthill includes a simulation environment to aid developers analyze and evaluate the behavior of P2P systems prior to deployment. Simulation parameters, such as the structure of the network, the ant algorithms to be deployed, characteristics of the workload presented to the system, are all defined using XML files, providing a flexible configuration mechanism. Unlike other toolkits for MAS simulation [11, 6], Anthill uses a single ant implementation in both the simulation and the runtime environments, thus avoiding the cost of re-implementing ant algorithms before deploying them. This important feature has been obtained through careful design of the Anthill API and by providing distinct implementations for simulation and deployment.

In addition to the adaptation properties derived from its multi-agent structure, Anthill pushes the analogy with natural systems even further by enabling developers to "evolve" ant algorithms to better adapt to certain tasks. This is accomplished through evolutionary computing techniques such as genetic algorithms [12] within the simulation environment. The set of parameters that define the behavior of an ant algorithm are considered its "genetic code" and the system automatically evolves ant populations so that successive generations improve upon an appropriate fitness measure.

In order to test our ideas regarding P2P as CAS, we have used Anthill to build a simple file sharing application called *Gnutant*. There is no doubt that building on top of Anthill has simplified the implementation. But more importantly, the resulting system indeed exhibits adaptiveness with respect to a variety of conditions and continues to improve its performance as time goes on, despite starting from a state of total ignorance. Gnutant itself is of interest as it combines the best characteristics of the two popular file sharing systems, Gnutella and Freenet [8, 10].

2 The Anthill Model

In this section, a description of the Anthill model is provided. The basic elements composing the model are defined, while the details of the prototype implementations of the model are postponed to Section 4.

An Anthill system is composed of a self-organizing overlay network of interconnected *nests*, as illustrated in Figure 1. Each nest is a middleware layer capable of performing computations and hosting resources. Any machine connected to the Internet and running Anthill can act as a nest. The network is characterized by the absence of any fixed structure, as nests come and go and discover each other on top of a communication substrate.

Each nest interacts with local instances of one or more *applications* and provides them with a set of *services*. Applications provide the interface between the user and the P2P network, while services have a distributed nature and are based on the collaboration among nests. An example application may be a file-sharing application, while a service could be a distributed indexing service used by the file-sharing application to locate files.

An applications performs *requests* and listens for *replies* through its local nest. Requests and replies constitute the interface between applications and services. For example, in a scientific document-sharing network, a request would be a query for a particular set of keywords, and the reply would contain a set of URLs to documents containing those keywords.

When a nest receives a request from the local application, an appropriate service for handling

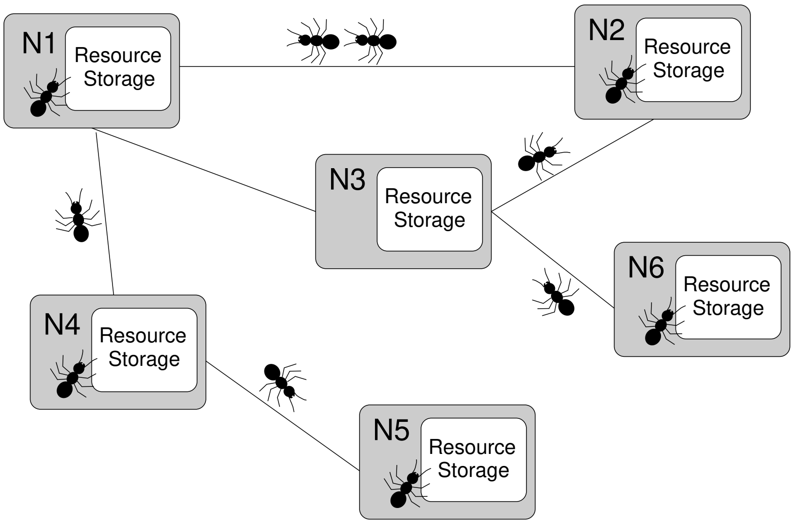

Figure 1. Overview of a nest network.

the request is selected from the set of available services. Services are implemented by *ants*, autonomous agents capable to travel across the nest network. In response to a request, one or more ants are generated and assigned to a particular task. While exploring the network, ants interact with the nests that they visit in order to accomplish their goal.

Anthill does not specify which services a nest should provide, nor impose any particular format on requests and replies. The provision of services and the interpretation of requests are delegated to ants. The set of available services is dynamic, as new services may be installed by the user.

2.1 The Nest

Figure 2 illustrates the architecture of a nest that is composed of three logical modules: ant scheduler, communication layer and resource managers. The *ant scheduler* module multiplexes the nest

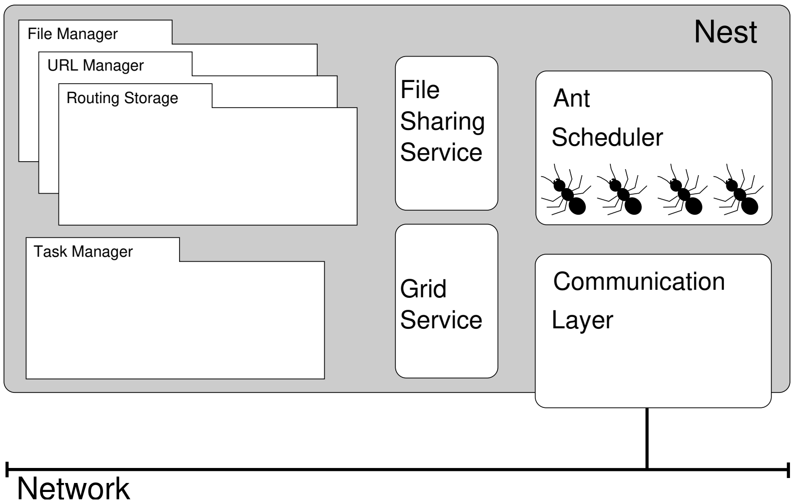

Figure 2. The architecture of a nest.

```
public interface Nest {
  void request(Request request,
     ReplyListener listener);
  void addService(AntFactory factory);
  void addNeighbor(NestId nid);
  void removeNeighbor(NestId nid);
  NestId[] getNeighbors();
}
```

Figure 3. The Nest interface.

computation resource among visiting ants. It is also responsible for enforcing nest security by providing a "sandbox" for ants in order to limit the resources available to ants and prohibit ants from performing potentially dangerous actions (e.g., local file access).

The *communication layer* is responsible for discovery of new nests, for network topology management and for ant movement between nests. In the network, each node has a unique identifier. In order to communicate with a remote node, its identifier must be known. The set of nests known to a node are called *neighbors* of that node. Note that the concept of neighborhood does not involve any distance metrics, since such metrics are application dependent and can more appropriately selected by developers. The collection of neighbor sets defines the nest network that might be highly dynamic. For example, the communication layer may discover a new neighbor, or it may forget about a known nest if it is considered unreachable. Both the discovery and the removal processes may be either mediated by ants, or performed directly by the communication layer. In the former case, ants may report about new remote nodes they visited, or may fail to move to a neighbor because of a communication problem. In the latter case, the exact implementation of discovery and removal depends on the underlying communication substrate, and is discussed in the next section.

Nests offer their resources to visiting ants through one or more *resource managers*. Example resources could be files in a file-sharing system or CPU cycles in a computational GRID, while the respective resource managers could be a disk-based storage manager and a task scheduler. Each resource manager is associated with a set of policies for managing the (inherently limited) resource. For example, a *least-recently-used* (LRU) policy may be used to discard items managed by a file manager when space is needed for new files. Each service installed by a nest is associated with a set of resource manager modules. For example, the nest in Figure 2 provides two distinct services: a file-sharing service based on a distributed index for file retrieval, in which a routing storage is used by ants in making routing decisions, a file manager is used for maintaining shared files and a URL manager is used to maintain the distributed index; and a computational grid application, in which a task manager executes tasks assigned to it.

The details of the Nest interface are shown in Figure 3. It contains the methods that may be invoked by the P2P application to interact with a nest. The main method of this interface is request(), that is used to perform new requests and to register a listener for replies. Furthermore, the interface also provides methods for nest administration, such as addition and removal of neighbors and registration of new services.

2.2 Ants

Ants are generated by nests in response to user requests; each ant tries to satisfy the request for which it has been generated. An ant will move from nest to nest until it fulfills its task, after which (if the task requires this) it may return back to the originating nest. Ants that cannot satisfy their task within a *time-to-live* (TTL) parameter are terminated. When moving, the ant carries its state, that may contain the request, results or other ant specific data. The ant algorithm may be transmitted together with the ant state, if the destination nest does not know it; appropriate code transfer mechanisms are used to avoid to download the same algorithm more than once, and to

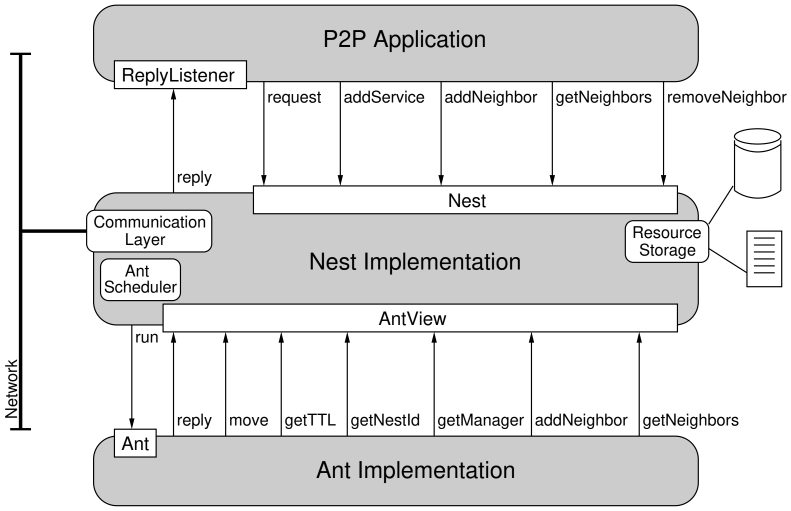

Figure 4. The Anthill interaction model.

update it when a new version of the same algorithm is available.

Ants do not communicate directly with each other; instead, they communicate indirectly by leaving information related to the service they are implementing in the appropriate resource manager found in the visited nests. For example, an ant implementing a distributed lookup service may leave routing information that helps subsequent ants to direct themselves toward the region of the network that more likely contains the searched key. This form of indirect communication, used also by real ants, is known as *stigmergy* [5].

The behavior of an ant is determined by its current state, its interaction with resource managers and its algorithm, that may be non-deterministic. For example, an ant may probabilistically decide not to follow what is believed to be the best route for accomplishing a task, and choose to explore alternative regions of the network. Ants must implement the Ant interface shown in Figure 4. The run() method contains the ant algorithm and is executed at each nest visited during the ant's trip.

The set of actions available to ants are limited to those included in the AntView interface, shown in Figure 4. Among these actions, ants are allowed to move to other nests, access local resource managers, obtain identifiers for the local nest and its neighbors, augment the list of neighbor nests, and finally notify the nest of a reply for a request originated in this nest.

Figure 4 gives a structural overview of the interactions between the various Anthill components. In addition to building a P2P application using the nest interface and the reply listener, the developer must also implement a set of ant algorithms facilitating services needed by that particular P2P application.

3 The Evolutionary Framework

In Anthill, we further exploit the "nature" metaphor by using evolutionary techniques for improving various characteristics of a P2P system. In particular, we make use of genetic algorithms [12] in tuning the ant algorithms used by the P2P system. An Anthill system is the composition of the operating environment and the collective behavior of ants, whose algorithms can be parameterized in various ways. The operating environment describes limiting factors such as disk capacities, connectivity degree, join and leave frequency of nodes, etc. A typical parameter for an ant algorithm is exploration probability, that will allow an ant to either deterministically follow what is reputed to be the best path towards a resource, or non-deterministically select one of the neighbors of the current nest, thus adding some degree of randomness to the exploration.

Using genetic algorithms, we can specify optimization criteria and constraints for the parameters of the operating environment and ant algorithms. A typical optimization criterion could be the minimization of the total path length traversed by ants (thus reducing the imposed network load), constrained by the connectivity degree, leave frequency and some threshold on the percentage of ants that succeed. Such optimization problems involve too many parameters and constraints to be solvable with traditional techniques, thus the need for genetic algorithms. We are currently extending the Anthill simulation environment to include optimization criteria definitions so as to allow automatic selection of parameters based on a fitness function (specific for each application).

In addition to the off-line use of genetic techniques, we are investigating whether genetic techniques can also be applied at run-time. For example, in order to satisfy a request, a nest could launch several ants, each characterized by a different set of parameters, and then rate them using a local fitness criterion. Subsequent requests could be delegated to ants derived genetically from those that were deemed fittest in previous requests. Nests could also "steal" the algorithms and parameters of visiting ants and use them in crossover and mutation techniques for generating new ants. It is interesting to note that the on-line evolutionary selection mechanism itself can be viewed as a P2P system whose task is to tune the ants of the original P2P application.

4 The Anthill Framework

In this section, we discuss the characteristics of the *runtime* and the *simulation* environments. These two distinct implementations of the Anthill model are used for real network deployment and for evaluation purposes, respectively. Unlike other toolkits for multi-agent simulation [11, 6], Anthill uses a single ant implementation in both the simulation and real network environments, thus avoiding the cost of re-implementing ant algorithms when deploying them and promoting the rapid deployment of prototype ant algorithms in the Internet.

4.1 The Runtime Environment

A prototype of the runtime environment, which is the distributed implementation of the Anthill model, is currently under development. The prototype is written in Java and is based on JXTA [7], which is an open-source P2P project promoted by Sun Microsystems. JXTA aims at establishing a network programming platform for P2P systems by identifying a small set of basic facilities necessary to support P2P applications and providing them as building blocks for higher-level functions. Although we are interested only in the Java version of JXTA, there are other implementations ("bindings") for different programming languages. Interoperability between the different implementations are guaranteed by the use of XML as the low-level formatting systems for messages exchanged between JXTA peers.

The benefits of basing our implementation on JXTA are several. For example, JXTA provides the possibility of using different transport layers for communication, including TCP/IP and HTTP, and is capable of handling firewall and NAT related problems. This spare our implementation from these low-level details. Furthermore, we may exploit the complex security architecture that is being developed for JXTA.

The JXTA middleware is composed of three layers. At the bottom is the *JXTA core*, that deals with low-level functions such as peer establishment, peer discovery, communication management and routing. The *JXTA services* are built on top of the core and deal with higher-level concepts, such as indexing, searching, and file sharing. These services, although useful by themselves, are used by *JXTA applications* to build high-level applications like chat, auction and persistent storage.

The runtime environment of Anthill is designed as a JXTA service and exploits the facilities offered by the JXTA core to provide an infrastructure for the construction of ant-based P2P distributed applications. It implements the `Nest` interface, providing methods for performing generic requests to Anthill applications. This nest implementation includes an ant scheduler capable of multiplexing the Java virtual machine among multiple visiting ants. Using the security model of Java, the execution of ants is confined to a controlled environment ("sandbox") by limiting their interactions with the local nest to those included in the AntView interface. A number of disk- and memory-based resource managers are provided, enabling the ant algorithms to make use of pre-installed classes in the visited nests. Nevertheless, it is also possible for ants to use their own specialized resource storages.

The communication layer is based on some of the fundamental primitives offered by the JXTA core, namely pipes, peer groups and advertisements. *Pipes* are communication channels for sending and receiving messages, and are used in the communication layer to move ants between nests. A *peer group* is a collection of cooperating peers providing a common set of services and speaking the same set of protocols. Peers may participate in several groups at the same time, thus offering several services. In Anthill, there is a general peer group constituted by all peers that are executing the nest service, and several peer groups constituted by the set of nests that accept to execute a particular ant algorithm. Owners of peer nodes are able to decide which kind of services their machines are going to offer, by accepting or rejecting the installation of new ant algorithms. Using features of the Java virtual machine, we are implementing a simple class loader capable of downloading the code of unknown ants from remote sites and cache it on local disks so as to avoid repeated downloads. Finally, *advertisements* are XML structured documents that describe and publish the existence of a resource, such as a peer, a peer group, or a service. Advertisements are used by the JXTA discovery protocol to locate services. In Anthill, they are used to advertise peer groups of nests offering a particular service.

4.2 The Simulation Environment

To evaluate ant algorithms, Anthill includes a simulation environment through which the behavior of a particular ant implementation may be simulated and assessed. Simulating different P2P applications require developing appropriate ant algorithms and a corresponding request generator characterizing user interactions with the application. Each simulation study, called an *experiment*, is specified using XML by defining a collection of component classes and a set of parameters for component initialization. For example, component classes to be specified include the simulated nest network, the request generator to be used, and the ant algorithm to be simulated. Initialization parameters include the duration of the simulation, the network size, failure probability, the number of requests to be generated, and the type and capacity of the resource managers to be used by ants. This flexible configuration mechanism enable developers to build experiments at run-time by assembling a collection of pre-defined and customized component classes, thus simplifying the process of evaluating ant algorithms.

In the current implementation, a P2P network is simulated inside a single Java virtual machine. The network is specified through the total number of nests and the number of neighbors associated with each nest. Initially, the required number of nests are generated, and the set of neighbors for each nest are selected randomly over the set of all nests. The network is dynamic, as new nests may join the network at runtime and existing nests may crash or voluntarily leave the network, based on pre-defined join and leave probabilities. Furthermore, the topology of the network may evolve during simulation, due to ants exploring the network and leaving information about remote nests.

Simulated nests are clearly distinct and simpler than the ones used in the runtime environment. Nevertheless, it is important to note that ant algorithms are totally independent of the nest implementation and continue to work in both environments without any changes. The communication layer is based on local interactions rather than remote communication; ants moving from one nest to another are simply transferred to the scheduling queue of the destination nest. Ant schedulers of simulated nests are controlled by a centralized scheduler, that uses the provided request generator to create requests and in a round-robin fashion invokes the ants run() method. Finally, resource managers are implemented as simple data structures with a maximum capacity and associated replacement policies.

The simulation proceeds by executing the sequence of generated requests on the nest network and by monitoring performance parameters such as the number of request initiated, satisfied, ant moves performed, network generated traffic, etc. The simulation environment enables pro-

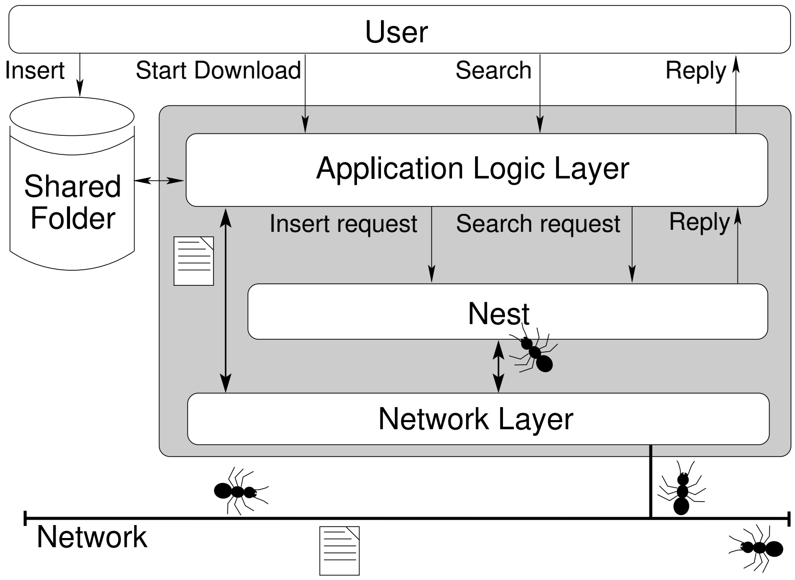

Figure 5. Gnutant Application Overview.

grammers to evaluate several different experiments and obtain average figures for the collected statistics. Monitoring network traffic is not performed at the packet level, but rather at the ant level, measuring the number of ants sent between various nests in the system.

In addition to this single-machine simulator, we are also investigating how we can distribute the simulated nest network on a cluster of Java virtual machines, to remedy problems with scaling the simulation environment to millions of nodes. The scalability problems of the simulation environment are due to the relatively large memory footprint required by nest implementation.

5 File Sharing in Gnutant

In this section, we present our preliminary experience in using Anthill to build a file-sharing application called *Gnutant*. In order to facilitate file searches, Gnutant builds a distributed file index scattered across the nest network, whose task is to store URLs for shared files, together with routing information needed to navigate through the index. The index is constructed at runtime by Gnutant ants, that travel through the network collecting information about new and existing files and insert this information in the index.

In Gnutant, each file is associated with some *meta-data* comprising a set of textual *keywords* and a unique *file identifier*. The keywords are used by Gnutant to organize the distributed index for routing, and may be provided by the user who inserted the file, or obtained automatically from the filename. The file identifier is composed of the file size and a cryptographic digest computed over the file content, and enables comparison of files for equality. Thus, different URLs for replicas of the same file will have the same file identifier. We exploit this property in order to provide faster file downloads, by requesting disjoint fragments of the file from multiple locations.

Figure 5 gives an overview of the Gnutant application. Users interact with Gnutant by copying files for sharing into a local folder, by issuing search queries and listening for replies, and by selecting files for download from remote sites. When the *application logic layer* detects a new file in the shared folder, an *insert request* is issued to the local nest in order to advertise the presence of the file to other nests in the network. Search queries presented by users are issued as *search requests*. Upon receiving a request, the nest will generate the appropriate ants to handle it. An insertion request for a file contains the file identifier, a URL and the collection of keywords, while a search request simply specifies a collection of keywords. We say that a file "satisfies a search request" if its set of associated keywords contains *all* keywords included in the search request.

As shown in Figure 2, each nest includes three resource managers: a *file storage* for managing files in the shared folder; a *URL storage* containing URLs to files; and a *routing storage* that ants may access or modify in order to make routing decisions or improve the routing of future ants, respectively. The URL and routing storages in the network constitute the distributed file index.

5.1 The Gnutant Ant Algorithms

In this section, we briefly present the ant algorithms used to implement Gnutant. Additional details can be found in a companion paper [13]. Gnutant ants are generated in response to user requests, and travel across the network trying to satisfy them. Three distinct types of ant algorithms are used, each of them specialized in a different task. The *InsertAnt* type is specialized in advertising the existence of files by insertion of URLs into the distributed index, and is used when there is a new file available in the shared folder, either because the user placed it there or after a download. The *SearchAnt* type is specialized in file searches, and is generated in response to user queries. It exploits the information left in routing storages by other ants, trying to determine the shortest path to files matching the user query. Upon reaching its TTL, the ant will return to the originator nest backtracking its path. During the return trip, the ant will update both the distributed index and the routing storages to reflect its findings. Finally, the *ReplyAnt* type is used to reduce the response times of searches. A ReplyAnt is generated at each nest where a SearchAnt locates a file. The ReplyAnt returns immediately to the originator nest, while the SearchAnt may continue its exploration to find other files satisfying the query.

To advertise a file, an InsertAnt is generated for each keyword associated with the inserted file. Similarly, a SearchAnt is generated for each of the keywords contained in a search request. Each of these ants carries also the entire query string. Together, the ants try to satisfy the given request concurrently, exploring different regions of the nest network, since each of them will be routed independently based on its associated keyword.

5.1.1 Gnutant Routing

InsertAnt and GnutantAnt make routing decisions using the specialized routing storage provided with Gnutant, by selecting the next nest to visit in their network exploration. The routing storage is based on the concept of *hashed keyword routing*, that is similar to the routing technique used in Freenet [10]. Routing storages associate the hash value of a keyword with a set of nests that are believed to store URLs for files associated with the corresponding textual keyword. When visiting a nest, ants inspect the routing storages using their associated keyword. If an exact match is found, the ant selects a nest from the set corresponding to the matching hashed keyword; otherwise, a nest associated with the "closest" hashed keyword is selected.

The hash value of a keyword is computed using the Secure Hash Algorithm (SHA) to obtain a 160 bit value. This mapping from the textual string space to the bit string space enables us to compare hashed keywords to determine their closeness. Furthermore, hashing the keywords also helps disperse the load evenly on the routing storages due to the uniformity property of SHA. Basing routing storages on the raw textual keywords would result in highly unbalanced load since keywords tend to be highly clustered in textual string space.

The notion of closeness between hashed keywords is fundamental to Gnutant's routing scheme. It allows nests to become biased toward a certain portion of the hashed keyword space. If a nest is listed in a routing storage under a particular keyword, it will tend to receive more requests for keywords similar to it. Moreover, nests become specialized in storing URLs of files having similar hashed keywords, since forwarding a request will result in the nest itself gaining a URL for the requested file. This clustering property will improve the search performance over time as the routing storages evolve their knowledge, enabling ants to quickly find the relevant region in the nest network.

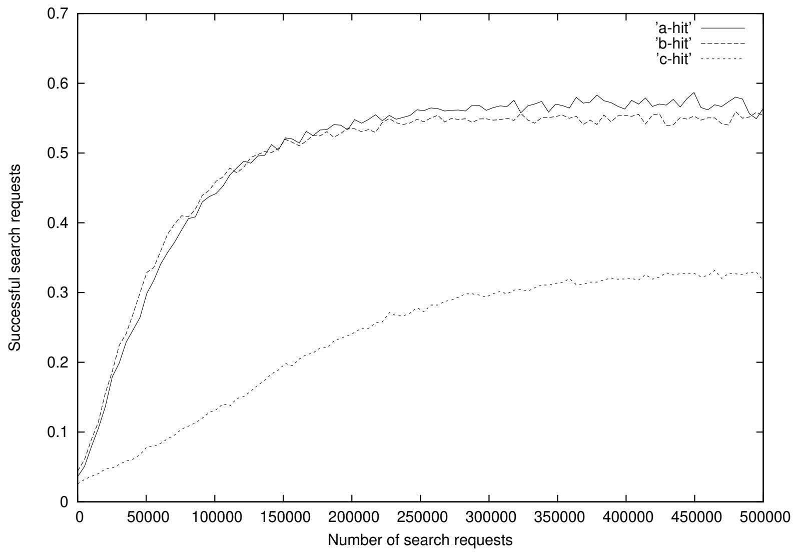

Figure 6. Search success rate.

5.2 Preliminary Simulation Results

In this section we present an evaluation of preliminary results for the Gnutant application obtained using the Anthill simulation environment. In order to render our simulation more realistic, we have collected a set of 10,000 query strings by monitoring the Gnutella network. The obtained query strings were also used as the names for 10,000 files, all of which were inserted into the nest network *a priori* to running the simulation. Thus potentially, all of the queries could have been satisfied. Furthermore, the routing storages were initialized with randomly generated SHA keys, causing the ants to move randomly in the beginning. The simulation was run on a static 2,000-node nest network with a fan out degree of 6 and 10.

After the insertion phase, 500,000 search requests were issued and statistics for the behavior of the system was collected. Search requests for the simulation was generated using a geometric distribution for selecting queries from the set of 10,000 Gnutella query strings. This distribution enable us to bias the search requests towards a certain portion of the available documents, i.e., the popular search requests are selected more frequently. The TTL parameter for the search ants was fixed at 10 hops.

The capacity of the routing, file and URL storages were set to 16, 16, and 64 entries, respectively. All resource storages use the LRU replacement policy. The number of search hits was sampled every 50th request, and the simulation was repeated ten times in order to obtain average values.

The simulation results are shown in Figures 6 and 7. Figure 6 shows the success rate for search requests, when 90% of the search requests correspond to 10% (upper curves) or 50% (lower curve) of the available documents. The fan out degree for curves a and c are 6, while curve b has degree 10. Figure 7 shows the number of hops necessary for the first reply to a successful search request. As expected, both figures confirm that the performance of the system improves over time, as the total number of performed requests increase and the content of the distributed index evolves. Furthermore, we can see from the figures that the system converges towards a 55% (33%) success rate for searches and approximately 2.3 (2.8) hops for the average search depth.

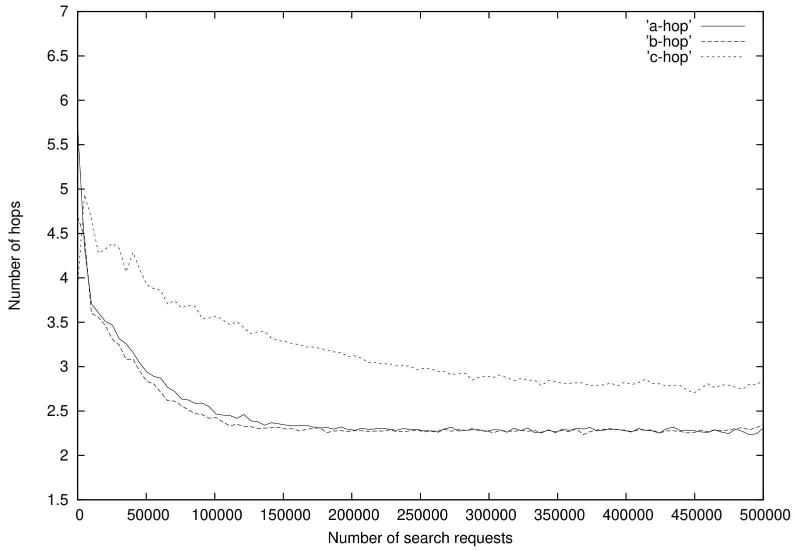

Figure 7. Number of hops until first reply.

6 Related Work

Since the original file sharing applications [17, 8, 10], a flurry of recent P2P projects in different application domains have come to being. These include persistent storage services [16, 9], distributed lookup services [15, 4], and distributed computing applications [1]. The importance of the P2P distributed computing model was recently recognized by industry, leading to several standardization and infrastructure efforts, including JXTA and the Peer-to-Peer Working Group [14, 7]. Anthill differs from these industrial initiatives because its main goal is to support the scientific investigation of the properties of P2P systems, by providing a simulation testbed for prototyping and tuning their P2P algorithms. On the other hand, we are exploiting the rich facilities offered by JXTA [7] as a basis for the runtime implementation of Anthill.

Anthill's simulation environment can, to some extent, be compared with agent simulators such as Swarm [11] and MASS [6]. Swarm is a general purpose software package for simulating distributed artificial worlds. It provides a general architecture for problems that arise in a wide variety of disciplines and is particularly suitable for problems involving a large number of autonomous entities "living" in an environment. MASS is an agent simulator developed with the aim of accurately measuring the influence of different multi-agent coordination strategies in an unpredictable environment. Anthill differs from these systems, as they are only focused on simulation, and do not support deployment in a real network environment.

Gnutant can be compared with existing file-sharing systems. In Gnutella [8], queries are text strings transmitted through broadcasting: each node receiving a query forwards it to all its neighbors. Being based on broadcasting, Gnutella is prone to serious scalability problems, and to avoid an exponential growth in the number of messages exchanged, strict limits are imposed on the TTL of messages and the number of neighbors known to each node. Unfortunately, these limits restrict the reach of a Gnutella query and thus the number of matching replies. Gnutant inherits the free search capability of Gnutella, without relying on inefficient broadcasting techniques.

In Freenet [10], each file is associated with a key obtained by hashing the file name. Search requests contain a single key, representing the desired file. Requests are not broadcast; instead, they are routed through the network using information gathered by previous requests. Freenet routing is based on the closeness between keys: if a node is unable to satisfy a request locally, it is forwarded to the node that is believed to store files whose keys are closest to the requested key. The main limitation of Freenet is that queries are limited to files with well-known names. Gnutant adopts a routing technique similar to that of Freenet, but adds the possibility of performing free search queries by associating files with keywords.

Despite the fact that Gnutant was designed for file sharing, the underlying mechanism could be used to implement generic distributed lookup services like CAN [15] and Chord [4]. Our lookup services would be probabilistic, since we cannot guarantee that existing name-value associations will be returned when performing a lookup. Using an ant-based implementation of a distributed lookup service provides ad hoc replication of the key-value associations, rather than uniform replication as provided by Chord and CAN. The drawback of uniform replication in highly dynamic P2P systems is that it requires a fairly complex rearrangement of replicas. Using ad hoc replication has the advantage that popular lookups will return quickly, while the drawback is that some (unpopular) index data may be lost due to lack of storage space at peers.

7 Conclusions

This paper presented Anthill, a framework supporting a new approach for building P2P applications based on the MAS paradigm, in which societies of autonomous agents cooperate in order to accomplish tasks assigned to them.

The Anthill project is in its early development stages. So far, we have implemented prototypes of the simulation and runtime environments, and we have used them to develop Gnutant, a set of ant algorithms for a file-sharing application. The simulation environment, with its flexible configuration mechanism, has been very valuable in supporting the design of Gnutant. Once tuned, Gnutant has been deployed without modification in the runtime environment, by simply substituting simulated resources managers with “real” implementations.

Work is under way to improve the simulation environment by augmenting it with an interface for the graphical visualization of the properties of the simulated ant algorithms. We also plan to use Anthill to evaluate properties of several existing P2P algorithms, such as persistent storage services [16, 9] and distributed lookup services [15, 4]. We are implementing ants that mimic the behavior of Freenet, for the purpose of comparison with Gnutant and studying how the reliability, availability and performance of hash-based routing may be improved. Finally, we plan to exploit evolutionary programming techniques to improve the performance of the resulting algorithms.

References

[1] D. Anderson. SETI@home. In A. Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 5. O'Reilly, Mar. 2001.

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

[3] G. D. Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[4] F. Dabek et al. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[5] P. Grasse. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. *Insectes Sociaux*, 6:41–81, 1959.

[6] B. Horling, V. Lesser, and R. Regis. Multi-Agent System Simulation Framework. In *Proc. of the 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland, Aug. 2000.

[7] Project JXTA. http://www.jxta.org.

[8] G. Kan. Gnutella. In A. Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly, Mar. 2001.

[9] J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proc. of the 9th International Conference on Architectural support for Programming Languages and Operating Systems*, Cambridge, MA, Nov. 2000.

[10] A. Langley. Freenet. In A. Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly, Mar. 2001.

[11] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations. Technical report, Swarm Development Group, June 1996. http://www.swarm.org.

[12] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Apr. 1998.

[13] A. Montresor, Ö. Babaoğlu, and H. Meling. Gnutant: Free-Text Searching in Peer-to-Peer Systems. Technical Report UBLCS-02-07, Dept. of Computer Science, University of Bologna, May 2002.

[14] Peer-to-Peer Working Group. http://www.p2pwg.org.

[15] S. Ratnasamy et al. A Scalable Content-Addressable Network. In *Proc. of the ACM SIGCOMM'01*, San Diego, CA, 2001.

[16] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, Canada, Nov. 2001.

[17] C. Shirky. Listening to Napster. In A. Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 2. O'Reilly, Mar. 2001.

[18] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.