A Complex Systems Approach to Service Discovery

Ricky Robinson and Jadwiga Indulska
School of Information Technology and Electrical Engineering
The University of Queensland
{ricky, jaga}@itee.uq.edu.au

Abstract— **Complex systems are those systems composed of many, often very simple, interacting autonomous entities. Interactions between these entities give rise to behaviour and patterns at the global level that cannot be predicted by examining the behaviour of any single individual component in the system. By this definition, pervasive computing environments are complex systems. This paper develops the idea that complex systems theory can aid the design of service discovery, and that results from the field of complex networks research can be applied to service discovery protocols to improve their scalability and robustness. We describe the influences of complex systems theory on the design of an existing service discovery protocol for pervasive computing environments, and show that the application of complex systems ideas can improve scalability, performance and robustness of service discovery protocols.**

I. Introduction

Pervasive computing environments are complex systems. These are environments where devices are distributed throughout, and the applications executing on each device operate with a higher level of autonomy than traditional applications. This autonomy is achieved through the use of such elements as service discovery and context information. These devices are often simple, with limited capabilities and dedicated to just one or two specialised tasks. It is their interaction with other devices that makes them useful. The composition of these devices, or more precisely the services executing on those devices, enables a broad range of tasks to be completed in the distributed environment. In essence, the whole is greater than the sum of the parts, a key feature of complex systems. Although in current computing environments, both the traditional and pervasive kinds, protocol and application specifications attempt to fully specify the allowable interactions and relationships between services and other entities in the system, other more intriguing and unexpected behaviours may arise from these interactions. In some cases, these emergent properties can contribute in a positive or negative way to the overall operation of the system, by influencing such factors as efficiency and robustness. The important thing to note is that these properties were neither designed nor predicted by the designer of the system.

This paper shows the application of ideas from the field of complex system to service discovery by using our own service discovery protocol, Superstring [8] as an example. Superstring contains elements to support wide-area and dynamic local-area interactions. We apply complex systems ideas to both the local-area and wide-area components of Superstring.

The remainder of this paper is organised as follows. Section II contains a discussion of the related work in the field of service discovery, and points out why these existing protocols do not meet the demands of large scale pervasive computing environments. Section III gives a short introduction to complex systems theory, and introduces some common terms to be used throughout this paper. In Section IV, we show how some of the ideas introduced in Section III have been applied to Superstring and the distributed hashtable algorithm on which it is based. Section V outlines other areas of pervasive computing which may benefit from the application of complex systems science, and concludes the paper.

II. Related Work

Existing service discovery protocols can be classified as commercial and widely available, or as academic research. It is interesting that most of the service discovery protocols developed as part of recent research projects take an entirely different approach than commercial protocols. The research focus in recent times has been on scalability. INS/Twine [2] and Superstring [8], for example, utilise an underlying peer-to-peer distributed hashtable as a basis for a service discovery protocol able to scale to at least metropolitan area networks.

In contrast, the available commercial protocols are focused on providing a discovery mechanism to relatively small networks. Hence, they utilise techniques such as multicast as exemplified by the Simple Service Discovery Protocol component of UPnP [4], centralised directories or resolvers such as that used in Jini [12], or rely on lower level protocols to discover other devices, each of which may be queried for services individually. This approach is used in Salutation [13] and Bluetooth [3]. Their emphasis is not to scale to large numbers of nodes and resources, rather it is to provide a simple discovery mechanism for a small community of devices.

INS/Twine and Superstring utilise the Chord protocol [11] as the underlying key lookup mechanism. The basic Chord algorithm utilises a circular skiplist structure. Each key identifier and node identifier is drawn (by means of hashing the key for key identifiers and hashing the IP address for node identifiers) from the set of m bit identifiers, where m is large enough that the probability of two nodes or keys hashing to the same identifier is negligible. Each key maps to a single Chord node. The mapping is such that the key is stored at the first node whose identifier is equal to or succeeds the key. Every node maintains a structure known as a finger table whose entries map identifiers to IP addresses. The first entry in the finger table contains the ID and IP address of the next node on the

identifier circle. The second entry contains the address of the node distance 2 away. The third contains the address of the node distance 4 away and so on. Thus the table contains m entries, with each successive entry being twice the distance from the current node as the preceeding entry. The routing of a key proceeds by finding the entry in the table whose identifier most closely preceeds the key in the identifier space. The node at that entry is contacted and provides the address of the next hop node by executing the same algorithm. In this fashion, the distance to the target node is reduced by half for each successive contact.

INS/Twine generates Chord keys by hashing strands extracted from hierarchical service descriptions and queries. These keys allow INS/Twine to identify nodes to which advertisements and queries should be routed. Superstring also defines a hierarchical service description and query structure. It generates Chord keys from the root component of service descriptions and queries. Again, Chord provides a scalable means to identify the nodes to which Superstring advertisements and queries should be sent. Any improvement to Chord will therefore carry over to Superstring and INS/Twine.

III. An Overview of Complex Systems

Complex systems is a relatively new field of science concerned with the way in which the behaviour of parts of a system give rise to global behaviours. A key aspect that differentiates complex systems from complicated systems is the nature of the interactions between the components in the system. A complicated system, such as a nuclear submarine, consists of many components that interact in a predictable fashion. In complex systems parlance, such a system is known as *linear* due to the predictable sequence of cause and effect. A complicated system (as opposed to a complex one) may consist of many thousands of components, but each component has a static set of other components with which it can interact. On the other hand, complex systems such as societies, economies and, we argue, pervasive computing environments, contain components that often interact in a non-linear fashion. An often quoted example of a non-linear relationship is the way in which the effectiveness of a particular medication changes as the dosage increases. A medicine may be completely ineffective until a threshold dosage is reached. The medication remains effective for dosage levels above this threshold, but may become ineffective or harmful if the dosage is too high.

These complex relationships can give rise to emergent behaviours. Emergent behaviours are those that arise at the global or system level and cannot be predicted from observing the behaviour of the individual components. Emergence is the formalism behind the common saying "the whole is greater than the sum of the parts". Some common emergent behaviours are those of self-organisation such as flocking and swarming in birds and insects. Computer networks such as the Internet organise themselves into scale-free networks. Such emergent patterns are not unique to computer networks. Indeed, social networks, the graphs formed by disease propagation and so forth, have all been found to have the scale-free property. In many cases, when emergent patterns and behaviours are observed, a power-law relationship lies at the heart of the interactions. Among other things, this aspect of complexity science is being used to combat the spread of viri, and also to study social interactions.

Recently, scientists have begun investigating how to improve the performance of various communication protocols armed with the knowledge that much of the time, computer networks have a scale-free structure. For example, researchers are improving the Gnutella peer-to-peer protocol [6], which had previously been shown to scale very poorly [7], by altering the protocol to exploit the scale-free topology of the peers. Anthill [1] is a framework for peer-to-peer computing that makes use of complex systems theory to provide scalability and adaptability. Anthill and Gnutella variants are only two of the attempts to apply complex systems theory to computer network protocols. Complex systems science, in our view, has much to offer the pervasive computing research community and distributed computing in general.

IV. Applying Complex Systems Theory to Service Discovery

A pervasive computing environment is an immense socio-technical system composed of people, institutions, devices, computer networks and protocols. All these elements interact with each other in diverse ways. Often an interaction is precipitated by environmental conditions (or context) surrounding a user or device. These elements and the interactions between them give a sense of the enormous complexity that can arise in a pervasive system.

The following sections present the design of an adaptable service discovery protocol, which uses complex systems theory and ideas from natural complex systems to provide scalability, robustness and flexibility in the face of diverse computing environments and mobility.

A. Overview

The wide variety of computing environments poses a problem to service discovery protocols (though these problems are not unique to service discovery). The differences in network characteristics and device behaviour of a wide-area environment with predominently static nodes as compared to a small ad hoc network of limited capability devices are enormous. The following sections present a service discovery protocol that abstracts away the differences in computing environments using a layered approach. The lower layers are optimised for particular environments, but each of these specialised layers utilises complex systems theory in order to improve its operation. This protocol defines two underlying layers. The first is based on the concept of ant foraging and is intended for use in small, highly dynamic environments. The second utilises an existing protocol overlayed on a distributed hashtable algorithm, to provide scalability to the wide-area [8]. Complex network theory is used to improve the performance of the distributed hashtable algorithm. Above these layers, a single service discovery interface exists to provide a consistent

means for clients and services to access service discovery functions regardless of the nature of the underlying network characteristics. Figure 1 shows the organisation of this layered service discovery protocol.

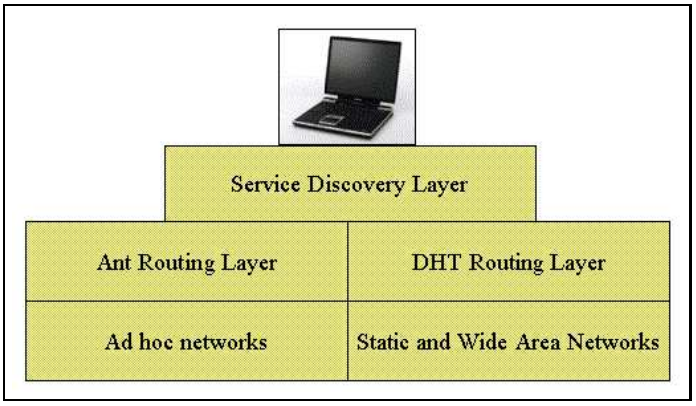

Fig. 1. A high level view of the service discovery architecture.

B. A Biological Model for Small Dynamic Networks

Many biological systems exhibit properties of self-organisation and emergence, and can be classified as complex systems. Such systems are inherently adaptive, but there is no centralised control. There is no authority determining when or how the system should adapt. Rather, the behaviour exhibited by the biological system as a whole emerges from the behaviours of the individuals within the system. Clearly such systems have parallels with pervasive computing environments, which likewise lack a central authority and must adapt to prevailing conditions.

The service discovery protocol described here is based on the process used by ants to discover food resources. In a process known as stigmergy, ants leave behind pheromone trails from food resources to the nest. This allows other ants to follow the trail. The service discovery protocol envisaged here also uses stigmergy. However, in this protocol (see algorithm below), pheromone trails can be differentiated by the type of resource they lead to. When a service is deployed it sends its service description to nearby nodes, thereby slightly increasing the chance of its discovery by a matching query.

procedure $receive(message)$
if $isQuery(message)$ **then**
  if $match(message)$ **then**
    $respond(matchingServiceRecords)$
  else if $expired(message)$ **then**
    $respond(unresolved)$
  else
    $decrementTTL(message)$
    $forwardRandom(message)$
  end if
else
  $store(message)$
  $forwardNextHop(message)$
end if

The above algorithm is executed at each node. Nodes receive messages. If the message is a query then it is checked against stored query results that have previously traversed through this node on their way back to a querying client. If there is a match, then the query terminates and the matching service record is returned. If there is no match and the query has reached its hop limit, then the query is unresolved and a message to that effect is sent in response. Otherwise, the query is forwarded to a randomly chosen neighbour. If the message is a positive query response, then the response is stored at this node and then forwarded in the direction of the querying client. Stored records eventually timeout unless they are reinforced by further queries. This simplified version of the algorithm ignores issues such as routing loops and query backtracking (if a loop or dead end is detected, the query should backtrack to the first node that has other neighbours to which the query can be sent).

If a service (or more correctly, queries for a particular kind of service) is popular, then the pheromone trails become reinforced. If a service is not popular, then the pheromone trail leading to it (which was created as the result of a matching query) will dissipate. This means that an already popular service will have more chance of being discovered than a less popular one because there are more pheromone trails leading to it, and thus likely that a query will come across an existing pheromone trail. In complex systems parlance, it can be said that the network *evolves* to favour popular services because resolution times for popular services will be faster than for unpopular services. The protocol adapts to node failures by reforming pheromone trails as time goes by, thereby providing some robustness. In the event of many simultaneous node failures resolution times, even for popular services, may be long, but they regain their former levels as more queries are issued.

Not only do ideas from complex systems - such as emergence, self-organisation and adaptation - aid the design of protocols for pervasive computing environments, but complex systems analysis techniques may also prove useful. Visualisaton is a powerful analysis technique used by many complex systems theoreticians. Whilst other methods of analysis may also be utilised, visualisation often provides quick insights into potential problems and unexpected behaviour. A visualisation of interacting agents created by Spector and Klein [10] provides a striking example of the way in which unexpected, seemingly intelligent, behaviour can arise from simple interactions. Whilst we do not elaborate on complex systems analysis techniques in this paper, we provide an example where visualisation can aid the design of a protocol such as that described above.

The above protocol based on ant foraging displays many advantageous behaviours such as biasing popular services and adapting to topology changes. However, a closer examination reveals a potential problem, and one that would become immediately obvious with a relevant visualisation. The creation of pheromone trails has the effect of *hiding* some matching services. The problem is that if multiple matching

services exist, then the one to have been discovered first has a much higher chance of being discovered first by future queries since the pheromone trail means it becomes more widely known. This has the unwanted effect of burdening some service instances while other service instances remain largely unutilised. A simple visualisation loaded with an example community of nodes (devices) and a simulation of the service discovery protocol will highlight this problem almost immediately. Ants themselves, provide a solution to this problem. A small proportion of worker ants in a colony will not follow established pheromone trails, electing instead to discover other food sources. We add this behaviour to our protocol by randomly deciding not to follow an established pheromone trail for a small percentage of queries.

The ant foraging algorithm has several benefits over existing protocols for small dynamic environments. Existing protocols either use an immobile central registry, or they require queries to be sent to all devices until the required service is found. The pheromone trail in the ant foraging algorithm allows all nodes to be mobile because there is no central repository and because pheromone trails will heal over time. Furthermore the pheromone trail allows the community to grow to many more than one hundred devices since service descriptions can be propagated throughout the network.

This section has shown the way complex systems theory can inform the design of a service discovery protocol for pervasive computing environments. The next section details an adjunct to a distributed hash table protocol inspired by ideas from the field of complex networks.

C. A Deterministic Model for the Wide Area

Another challenge is to find a protocol that will scale to much larger networks than the ant foraging protocol described above and that will provide the guarantee of no false negative responses. Peer-to-peer hashtable protocols such as Chord [11] and Pastry [9] offer a way in which to guarantee, in the absence of node failures, the success of key lookups in a distributed environment. INS/Twine [2] and previous work by the authors [8] are built upon these distributed hashtables to create a reliable service discovery mechanism. But by building on top of a deterministic hashtable structure negates any benefits of the scale-free nature of the underlying network upon which the distributed hashtable is built. That is, capable, highly connected nodes are forced to play the same role as less capable and less connected nodes. An obvious question to ask, in light of this discussion of complex and scale-free networks is, can the distributed hashtable be made scale-free? Doing so has the important benefit of reducing the average path length from one node to any other node in the peer-to-peer hashtable. In turn, any service discovery protocol overlaying such a scale-free hashtable would achieve lower query latency. The remainder of this section is devoted to detailing the mechanism by which Chord can be influenced toward a scale-free topology.

First of all, note that *any* type of caching in Chord will result in a reduced average path length from one Chord node to another. As such, a simple modification to the algorithm and the structures used to support the algorithm can yield a more scalable protocol in terms of the average number of nodes that need to be contacted during lookup. The simplest addition to the Chord algorithm is to cache the addresses of nodes which are discovered during a lookup. The finger table then stores as many identifiers and IP addresses for each interval as it is capable of doing. During lookup, the node can then choose to contact either the node with the closest ID to the target node, or it can measure response times, and choose to contact the node with the fastest response. This elementary improvement, or one very similar to it, is suggested in the Chord paper itself.

Complex network theory suggests a way in which this cache can be used to transform the Chord protocol to tend toward a scale-free set of interactions, thereby reducing the average number of nodes contacted during each lookup further than the above simple caching mechanism could do. In a heterogeneous network, such as one might find in a pervasive computing environment, one will find all manner of devices, some of which are capable of storing large amounts of data and are connected to high bandwidth links. Other devices will be resource poor, and are connected to other devices only through low bandwidth links. If a caching function is introduced to Chord, the size of the cache will vary with the capability of the Chord node. Nodes with large caches will generally have information about nodes closer to the target than would be the case in the original Chord algorithm. This is true because complex network theory holds that highly connected nodes, or hubs as they are known, can usually reach every other node in the network in a small number of hops. In the context of our hashtable protocol, it means that it is likely that highly connected nodes can more than halve the distance to the target on each hop, since it is probable they know of nodes closer to the target than was possible under the original Chord protocol. The new algorithm maintains correctness since each step brings the querying node at least twice as close to the target node as the previous step.

Capable nodes cache the ID and IP address of each node contacted during their own lookup operations. This requires no additional message overhead. Less capable nodes operate similarly, but only cache the IDs and IP addresses of the most highly connected nodes within an interval. The size of the cache is completely adjustable for each node. In our simulation, nodes are assigned a random *capability* that represents a space metric. In our testing, the maximum capability that can be assigned to a node is ten percent of the total number of nodes in the simulation. The minimum capability is one.

Our results indicate that this simple modification of the original Chord algorithm results in a significant lowering of the average path length between nodes. In fact, as more and more lookups are performed, the average path length between nodes shortens. Figure 2 summarizes the results obtained from our simulation of the modified Chord algorithm. For each test, a million lookups were performed, and the average path length calculated. For example, in a network of 512 nodes, the average path length in the original Chord algorithm is

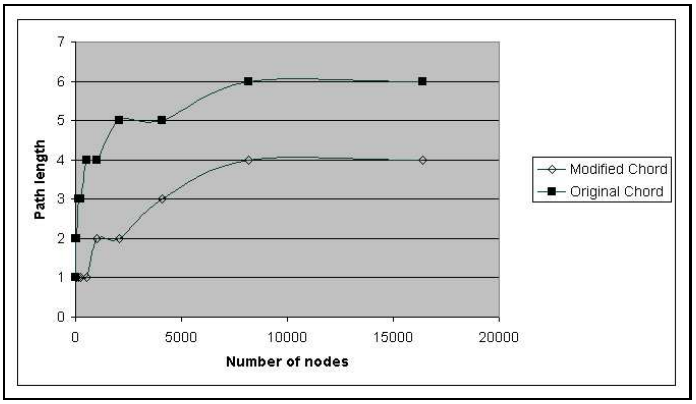

Fig. 2. Average path lengths of the original and modified Chord algorithms

approximately four. In the modified algorithm, after a million lookups, the average path length is one. In larger networks, where our simulation shows the hop length has been reduced to two thirds of its original length, the hop length is reduced even further if more than a million queries are issued since knowledge of highly connected nodes will propagate further through the network. The choice of caching algorithm does have an effect on the average path length. To determine this, the caching algorithm was modified to cache the *least connected nodes*. In a network of 512 nodes, the average path length is two. Therefore, the cache replacement algorithm has a bearing on the average path length in the network. Furthermore, we do not consider a least-recently used purging policy since node popularity changes dynamically when new nodes join the network and when new keys are added to the distributed hashtable. The capability of a node, in terms of its capacity to store pointers to other nodes, is a static property, and therefore a better choice on which to base a cache purging policy. Superstring and Twine can benefit from the performance improvement gained from the above modification to Chord. Superstring, because it is targeted at those situations in which ad hoc communities of devices are linked by more permanent infrastructure, can especially benefit from this scalability gain, since the core infrastructure is not mobile. That is, it is very likely that nodes in the core infrastructure will make millions of lookups during their lifetimes, and thus benefit from these modifications.

V. Conclusion

Our foray into the application of complex systems techniques to service discovery has suggested many other areas of pervasive computing where complex systems theory could be applied.

For example, our previous work on context [5] indicates that the relationships between context elements, such as location, user preferences and device capability are not simple or linear. Rather, they are interdependent and can be affected by actions of the user. Therefore, context is a primary example of a necessary pervasive computing component that meets the definition of a complex system. Treating it as such may provide new insights into creating a scalable context management system, and might yield useful emergent context elements that go unnoticed using existing techniques. Current approaches to context management tend to focus on small subsets of context information (primarily location information), and this is inadequate for the pervasive environments of tomorrow.

Applications for complex systems theory are being found in fields as wide-ranging as sociology, biotechnology and immunology. In this paper, we show a novel application of complex systems theory to service discovery. In particular, this paper shows that the theory of complex networks can be applied to create a scalable distributed hashtable algorithm, which can benefit any service discovery protocols overlayed on them.

Pervasive computing environments *are* complex systems. They abound in intricate relationships that give rise to complex behaviours. This paper highlights how the pervasive computing research community could benefit by embracing complex systems approaches as a way to designing more robust and scalable components for pervasive computing environments.

References

[1] Özalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002.

[2] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *Pervasive 2002 - International Conference on Pervasive Computing*, number 2414 in LNCS, pages 195–210. Springer-Verlag, August 2002.

[3] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.

[4] Yaron Y. Goland, Ting Cai, Ye Gu, and Shivaun Albright. Simple Service Discovery Protocol/1.0. IETF draft specification, October 1999.

[5] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In *The 4th International Conference on Mobile Data Management*, 2003.

[6] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernado A. Huberman. Search in power-law networks. *Physical Review E*, 64, 2001.

[7] Jordon Ritter. Why Gnutella Can't Scale. No, Really. Technical report, Darkridge Security Solutions, Available from: http://www.darkridge.com/ jpr5/doc/gnutella.html, 2001.

[8] Ricky Robinson and Jadwiga Indulska. Superstring: A scalable service discovery protocol for the wide-area pervasive environment. In *The 11th IEEE International Conference on Networks*, 2003.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, number 2218 in LNCS, pages 329–350, Heidelberg, Germany, November 2001.

[10] Lee Spector and Jon Klein. Evolutionary Dynamics Discovered via Visualization in the BREVE Simulation Environment. In *ALife VIII: Workshop Proceedings*, 2002.

[11] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*. MIT Laboratory for Computer Science, August 2001.

[12] Inc Sun Microsystems. Jini Technology Core Platform Specification v1.2. Technical report, Sun Microsystems, Inc, 2001.

[13] The Salutation Consortium. Salutation architecture specification (part 1) v2.0c. Specification, The Salutation Consortium, June 1999.