

Chapter 6

Approaches for Ubiquitous Computing

6.1. Introduction

Ubiquity of communications and services represents the possibility for a user to access services and resources regardless of where he is, when, and from a fixed or mobile terminal [WEI 93]. It is the development of wireless telecommunications, mobile networks and their crossing with fixed communication networks, i.e. proximity and remote networks, enabling us to consider this ubiquity of services and resources today.

The major challenges for ubiquitous implementation are, on one hand, the introduction of automatic adaptation possibilities in processes and communications for dynamic network modifications, of the number and availability of resources and of user behaviors, and the other hand, implementation of mechanisms for the development of services [GAB 00, GAB 06, BAK 07]. In other words, the implementation of new applications in the context of ubiquitous computing requires the development of new approaches having self-organization, self-adaptation and emergence capabilities for mobile ad hoc and sensor network environments.

It should be noted that according to Gaber's classification, interaction paradigms can be classified into three categories: the traditional Client to Server paradigm (CSP) and two alternative paradigms, the Adaptive Services to Client Paradigm (SCP) and the Spontaneous Service Emergence Paradigm (SEP). These alternative paradigms that are more suitable for Ubiquitous and Pervasive Computing respectively require self-organizing, self-adaptive and emergence capabilities to

cope with dynamically changing context environments such as computing contexts and user contexts [GAB 00, GAB 06, BAK 07].

The traditional client/server paradigm is inadequate for ubiquitous computing implementation. More precisely, in this paradigm, the client is the one taking the initiative of requesting an existing service and has the means to locate it. In this chapter, we focus on the implementation of an alternative paradigm, contrary to the traditional client/server paradigm; the service will go to the client instead of the client taking the initiative and requesting a service, or a resource by first knowing its existence and location. In addition, by integrating self-adaptation and self-organization possibilities, this paradigm enables the development of services with the help of a learning mechanism based on resource availability and user behaviors [GAB 00, GAB 06, BAK 03, BAK 05, BAK 07].

This chapter is dedicated to the presentation of this approach and methodologies found in the literature for the implementation of ubiquitous computing. More precisely, we are talking about service discovery and development systems in mobile ad hoc networks, or MANET, and wireless sensor networks. A mobile ad hoc network is a system made up of mobile sites. A site is any mobile object capable of communicating over wireless network. These objects can be personal assistants, laptop computers, cellphones or sensors. There is no centralized management in these networks. More precisely, mobile hosts form, in an ad hoc manner, a network infrastructure. In addition, no assumption or limitation is made on the ad hoc network's size; the network may contain hundreds or thousands of mobile units [FOU 04, PAR 05]. This network lets its users access services whatever their geographical location [FOU 04]. In order for these users to access distributed services in a mobile environment, a services discovery system is required. A services discovery system is based on the definition of an interaction protocol or mechanism between users and available services. In other words, it allows users to locate a service in order to access and use it [BAK 05].

A service is an application or a software component with a specifically defined function. It is directly accessible by users through the use of requests. It can also interact with other services throughout the network with the purpose of creating combined services. More precisely, a service is an application equipped with a well defined interface for accessing and using one or more resources. When there are multiple resources, they can be located in one single node or geographically distributed between several network nodes. We call combined service a service based on distributed network resources. For example, current peer-to-peer (P2P) systems are systems where the service provided is sharing of disk space and file transfers distributed over several network servers.

Service discovery systems which exist in the literature can be classified into three families according to their architectures or their operation modes (see Figure 6.1). The first family groups systems which implement a structural organization for the location of network services. In other words, they are systems using indexing or hash mechanisms. The second family groups systems which are not based on any specific structure or organization. These systems are commonly called unstructured service discovery systems.

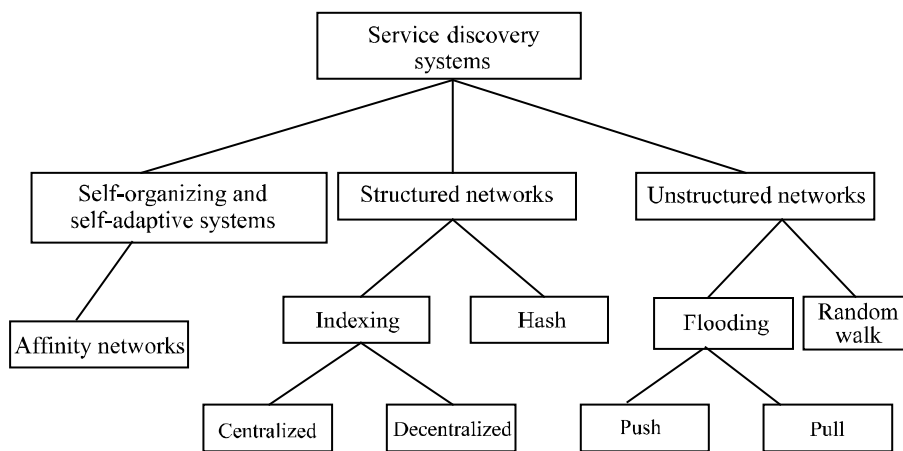


Figure 6.1. Classification of service discovery systems from the point of view of their architectures or their operation modes

The third family groups self-organizing and self-adaptive systems. In this category, we present an approach called AFFINITY, based on the mobile agent paradigm and inspired by the human immune system for services discovery in MANETs. This approach integrates possibilities of reduction/expansion for automatically adapting to an environment where the evolution can be random and which can adapt to combined mobility of network nodes and users. The human immune system presents interesting functional properties such as self-organization, self regulation, emergence and self-adaptation to an environment where evolution is dynamic and random. The analogy with the immune system is made in the following way [GAB 00, GAB 06, BAK 03, BAK 05, BAK 07]: a user request corresponds to an antigen (or a virus) attacking the network. The network then behaves like the immune system to eliminate the attacking antigen, i.e. to respond to the user's request. In other words, the requested resource or service corresponds to the immune response. This represents an alternative approach, contrary to the traditional client/server paradigm; it is the service addressing the client and not the client that

takes the initiative and requests a service or a resource knowing its existence and location.

In this chapter, we present a study of service discovery systems with their architectures and operation modes in ad hoc networks as criteria of comparison. We begin by describing the different structured and unstructured service discovery systems proposed in the literature in sections 6.2 and 6.3 respectively. In section 6.4, we present a comparative study of these systems in a dynamic context. In section 6.5 we present a self-adaptive approach based on the creation of communities for services discovery in mobile ad hoc networks.

6.2. Structured service discovery systems

6.2.1. Systems based on an indexing mechanism

These systems are based on the use of directories (or service lookups) and they can be categorized into two families. The first one groups systems using a centralized indexing mechanism. They are offered for locating services in small-sized networks. The second family groups systems based on a decentralized indexing mechanism. They are offered for locating services in large-scale networks.

In order for servers to register their services in these directories and for users to access them, a mechanism for locating servers storing these services is needed. There are three types of locations [GUR 03]: static location, passive location and active location. In the case of a static location, addresses of servers storing the directories are provided to users and servers by a manual configuration. For passive location, lookup servers periodically broadcast messages announcing their presence. In the case of active location, users and servers are the ones broadcasting messages to locate service lookup servers.

To enable users to know when a service arrives or leaves, leasing and notification mechanisms are used [GUR 03, PER 98]. With the leasing mechanism, the server must indicate the time period during which its service will be valid. More precisely, before the expiration of this delay, it must renew dissemination of its service; otherwise, it is considered as unavailable by the user. The notification mechanism informs users, before the expiration of the service validity period, of certain events, for example, the departure of the service or a change in this service's parameters.

6.2.1.1. *Centralized indexing*

The approach based on the use of the centralized indexing mechanism is the most widely used by service discovery systems in small networks [BET 00, GUR 03, ROB 98]. As an example, the Jini middleware services discovery system developed by Sun Microsystems [GUR 03] uses a directory which maintains information on available network resources. The three types of service lookup location can be used in Jini [MAT 01]. The standard Corba middleware from OMG (Object Management Group) also uses the indexing mechanism [GUR 03] to locate services. CORBA [GUR 03] uses static location of the service lookup server.

In these systems, servers communicate server information on their services to the directory. Location of a service is done in the following manner: the user sends a location request to the service lookup server to get a list of servers whose available services correspond to the request criteria. The user makes her choice and then directly connects to the server with the requested service.

Location by using a single centralized service lookup server is simple and avoids having to broadcast or multicast information on services or requests searching for these services. However, the fact that all information concerning services is concentrated in one place constitutes a major drawback of this model in terms of scalability [MIL 02]. Another problem raised by this approach is its vulnerability to failures and to intermittent connections as this server is a mobile node. In fact, central server failure or disconnection would make it impossible to locate services.

6.2.1.2. *Decentralized indexing*

This indexing mechanism was mainly proposed to solve scalability issues as well as the problem of failure of the service lookup server using the centralized indexing mechanism in wired networks. As an example, the SSDS (*Secure Service Discovery System*) architecture, developed in the context of the Berkeley Ninja Research Project at University of California [CZE 99], uses service lookup servers structured in a hierarchical way for services discovery [XU 01]. Basic entities of this architecture are SC (Service Client), SP (Service Provider) and DS (Discovery Service). SCs correspond to users discovering and using services. SPs correspond to service providers. DSs are domain servers acting as service lookup. A DS is associated with each domain.

To disseminate information on services, each SP sends its service information to its known domain server by manual configuration (or static location). This domain logs the information and then sends it to its parent server in the hierarchy. The parent server logs this information and in turn sends it to its parent server. This procedure is repeated until service information reaches the central service lookup

server (root of hierarchy). In this system, information on services is compressed using the bloom filter technique [CZE 99].

Location of a service is done in the following way: the user sends its request to its DS. The DS looks in its service lookup if it can locally respond to the request (i.e., the service lookup knows the location of the requested service). If that is the case, it responds to the user's request, otherwise it sends a request to its parent server. This procedure is repeated until the requested service is found.

The SLP (Service Location Protocol) system is proposed by the Internet Engineering Task Force (IETF) [BET 00, GUT 99] for locating services on small-sized networks [PER 98]. The architecture of SLP is made up of three types of entities represented by agents: a service agent (SA) which represents a service, a user agent (UA) acting on behalf of the user and a directory of agents (DA) which centralizes information on services [AZO 02]. UAs discover services offered by SAs and which are logged in DAs. However, no detail is given on how to organize DAs. An extension of SLP is mSLP (Mesh-enhanced Service Location Protocol), which is more scalable because of collaboration between several DAs [ZHA 02] organized into a mesh network. In other words, it is a topology where each DA is connected to all other DAs in the network. In fact, service information is duplicated on DA service lookup servers. However, if the services frequently join or leave the network, information update on these services overloads the network and generates a bottleneck for DAs.

This type of architecture based on hierarchical service lookup servers was proposed to solve the scalability issue in wired networks [LIU 03]. It cannot be suitable for mobile ad hoc networks because the user's connection with the service lookup server and between the server and the other directory servers is difficult to maintain because of node mobility [CHO 05, HAU 05, LIU 03]. It also increases the cost of periodic updates on service information arriving or leaving the network. In addition, the failure or disconnection of a subordinate server or the central server in the hierarchy would make locating services impossible.

To compensate for these problems, several approaches have been proposed for services discovery in mobile ad hoc networks [CHO 05]. These approaches are based on mechanisms of the election of nodes which could assume the role of service lookup. As an example, studies proposed to log service information to nodes called dominator nodes [KOZ 04]. These nodes are considered stable. They are selected according to the frequency of their disconnection. More precisely, the services discovery protocol is made up of two phases: election phase for dominator nodes (BBM (BackBone Management)) and services location phase (DSD (Distributed Service Discovery)). The first phase consists of selecting a set of dominator nodes in the following way: a node with a low disconnection frequency

(NLFF (Normalized Link Failure Frequency)) becomes a dominator node. In other words, this phase consists of eliminating nodes with a NLFF higher than a given threshold $nlff_{th}$. Following this selection phase, each dominator node broadcasts a message to the other dominator nodes. In fact, dominator nodes form an overlay network in the form of a grid (or a mesh).

If it is not dominating, a node is connected to a virtual access point (VAP) to all dominator nodes. It logs information concerning its services with the DA present in its VAP. In order for the other nodes to know the validity of information logged in their cache, a leasing mechanism is used. More precisely, each node must indicate the time period during which its service is valid. Before expiration of this delay, it must renew dissemination of its service with its DA, otherwise it will be considered unavailable by its DA and will be deleted from its cache [KOZ 04].

When a user wants to search for a service, it sends a location request to its VAP and to all neighbor VAPs until the service requested is reached or when a specific search threshold (TTL (Time To Live)) is reached.

Another approach proposes the organization of services in a ring [KLE 03]. In other words, nodes which are physically close and offer similar services are grouped into a ring. Rings form a hierarchical structure. Each ring is represented by an elected node considered as a service access point (SAP) to services offered by nodes in this ring. By using this structure, the location of services becomes simple and efficient. A request is routed throughout the rings until it reaches the service requested. In this approach, services are represented by the DAML-S (DARPA Agent Markup Language Service ontology) language. A similarity function between two services was also proposed. This function returns an integer indicating if two services are similar or not.

An approach was proposed in [SAI 05] where nodes which can play the role of service lookup are elected among all network nodes based on their capacity in terms of resources and constraints of their environment. Each node with the role of service lookup is responsible for storing information on available services in neighbor nodes at H hops. It also exchanges its profile with other directory nodes. In other words, it is the capacity of the node and a summary in condensed form of information on services offered by nodes of its neighborhood by using the bloom filter technique [SAI 05]. Services are represented with a Web Service Description Language (WSDL). When a user wants to locate a service, it sends a location request to its service lookup node. If this node does not have the service or information on the requested service, it broadcasts the request to one or more nodes liable to solve the request based on their profile. Content management for service lookup nodes is based on a policy of cooperative caches enabling these nodes to benefit from information present in the service lookup nodes. This cache policy is efficient in

terms of time of request resolution and message complexity. However, it does not take into account node resources in terms of storage capacity [HAU 05]. In addition, cache maintenance can be difficult because of node mobility.

To resolve the cache maintenance issue, another approach was proposed in [HAU 05]. It is a solution which distributes information on available services in the network by decreasing the distance between a node that has kept a trace of information on a service and a node wanting to locate this service. In order to do this, a distance factor was introduced. This factor controls the maximum distance separating a node with a trace of information about a given service from a node without it [HAU 05]. In addition, a node must be able to find the requested service or information on its location from a closer node. The smaller the distance factor, the shorter the distance separating a node with information on the service of a node not having it. In other words, the smaller the distance factor, the shorter the time for locating a given service.

A method proposed in [LIU 03] is based on a dynamic node election mechanism playing the role of service lookup. More precisely, a network node can play one or more of the three following roles: a user role (or *client*) in the case where it requests a service, a provider role (or RP for *Resource Provider*) in the case where it offers a service or a service lookup role (DA for *Discovery Agent*) if it is elected by the other nodes. In other words, users discover services offered by RPs and are logged in DAs. Each DA is responsible for a set domain in collaboration with the other network DAs. The selection of a node to play the role of a DA is based on an election algorithm in the following way. Each node broadcasts its presence to all network nodes and the node with the smallest identifier will be elected as DA. Let us suppose that M DAs were elected, the initial DA must choose $M-1$ nodes to form the set of DAs indexed by the set $\{2, \dots, M\}$. The initial DA takes index 1.

Following this phase, DA addresses are periodically broadcast in the network. Each node not playing the role of a DA must choose its closest hDA (or *home DA*) to log information on its services. We should note that nodes are mobile; consequently, DAs must update members present in their domain. Each DA periodically broadcasts a message to its neighbors containing its index, message expiration date and a variable representing distance covered by this message. When a non-DA node receives this message from a DA node, it looks at the distance separating it from its DA. If this distance is higher than the current distance from its hDA, it will not rebroadcast the message, otherwise it considers this DA as its hDA and broadcasts the message to its neighbor nodes.

The third phase involves logging service information in DA nodes. Each service is characterized by an attribute known to all network nodes. A node providing a service announces its service to its hDA by sending a logging request. This request

contains the PR node address, attribute of its service α , validity date of this announcement. When the DA receives this message from a PR, it calculates its index β in all indexes $\{1, 2, \dots, M\}$ by using a hash function $H(\beta = H(\alpha))$. It then sends this message to DAs: $DA_{\beta}, DA_{\beta+1}, \dots, DA_{\beta+k-1}$ to log information concerning the attribute service α .

Service location in DAs is done in the following way. A user wanting to locate a service sends a request to its hDA. If it does not have the service or information on locating the requested service, it calculates its index β by using function $H(\alpha)$ to get all qualified DAs: $DA_{\beta}, DA_{\beta+1}, \dots, DA_{\beta+k-1}$. The request is then broadcast to the DA closest to hDA. If this DA cannot respond, it chooses the second DA of this group and the procedure is repeated until the requested service is located. In this way, the user receives a list of PRs and chooses the one providing the service with a given quality of service.

In this approach, DAs are organized into a mesh network. Or in other words, it is a topology where each DA is connected to all other network DAs. This method enables users to locate available services in the network. However, periodic broadcasting of logging requests between DAs in order to maintain the network between DA nodes as well as information on services carries the risk of overloading the network.

6.2.2. Systems based on distributed hash

Hash-based systems are service discovery systems mainly dedicated to locating available network files without going through a service lookup server. These systems are based on an approach founded on the use of a distributed hash table [LI 02, LUA 05, MIL 02, RIS 04, SCH 03]. This approach is implemented in the following manner: a hash function is applied to each node's IP address in order to generate its identifier called nodeID. Similarly, the identifier of a file, called fileID, is generated from the file's name or content by using this same function [FEL 03]. Using a hash function makes building an overlay network functioning independently from the underlying physical network possible. In this way, a set of file identifiers corresponds to each network node. For example, nodes receiving files whose identifiers are close to the node identifier. This matching between nodeID and fileID enables routing of requests toward nodes with a nodeID as close as possible to the requested file's fileID [JAN 02, LUA 05, MIL 02].

The system proposed by Plaxton *et al.* [PLA 99] is the first system based on the development of an overlay network in the form of a mesh [LI 02]. File and node identifiers are generated by hash function SHA-1 (*Secure Hash Algorithm version 1*)

[PLA 99]. With this function, you can calculate a value (number) representative of an element (or generally a message).

Other, more elaborate, systems derived from Plaxton routing and locating mechanism were proposed for file transfers in a dynamic network such as Pastry [ROW 01] and Tapestry [ZHA 01]. In order to keep root nodes from indicating that an absent node holds a file, a periodic file information dissemination mechanism is offered by the Pastry and Tapestry system [LI 02]. In fact, a node deletes the pointer to any node which has not disseminated its files in a set delay. Similarly, during the dissemination of a file, if its root node no longer exists, another node replaces it. For example, in Tapestry, several root nodes are defined as responsible for the same file. In this way, failure of a root node would not make it impossible to locate files. However, in a dynamic network in which nodes and connections appear and disappear over time, periodic information dissemination of files can overload the network [GAU 02, JAN 02].

Other file transfer systems exist which are based on the construction of an overlay network such as Content-Addressable Networks [GAU 02, RAT 01], Chord [STO 01] and Viceroy [GAU 02, MAL 02]. This node structure in overlay networks enables the location of a file in an efficient manner, but in return requires a development and maintenance algorithm for this network during frequent arrival or departure of one or more nodes [LIB 02]. In addition, the overlay network between pairs does not reflect the physical network topology at all. In other words, two nodes distant in the overlay network are in reality very close geographically and vice versa. To avoid this loss of efficiency for service discovery in mobile ad hoc networks where the notion of proximity is very important, geographic location must be taken into account during creation and update of the overlay network. More precisely, allocation of identifiers must be based on the network's topology. In addition, these systems do not consider node resources in terms of storage capacity [ROB 04]. An approach using the Chord system [STO 01] was proposed for services discovery in mobile ad hoc networks with a cache adjustment mechanism [ROB 04]. In other words, nodes with a higher storage capacity can much better cache information about the services than other nodes.

6.3. Unstructured service discovery systems

The second family of service discovery systems groups all systems said to be unstructured. In this type of system, no service lookup will centralize information on services. To locate a service, two mechanisms can be used: flooding or random walk [GUR 03, SHU 02].

6.3.1. *Flooding-based mechanism*

Two techniques based on the flooding mechanism can be used for the discovery of services present in the network: push and pull techniques. In the push technique, the user periodically broadcasts his discovery request for services available in the network. In the pull technique, it is the servers who periodically broadcast information on their services, making users aware of available services in the network.

Several services publishing strategies were proposed for service discovery in a mobile ad hoc network using the pull technique: a greedy strategy, incremental, uniform memoryless, with memory and a conservative strategy [HAU 05, LUO 03, LUO 04]. In the first strategy, a node publishes its service to all network nodes. A user wishing to locate a service broadcasts his request to all network nodes. In fact, if all the nodes know all available network services, users will not transmit their requests. In an ad hoc network, the nodes are limited in storage capacity. It is impossible for all nodes to store information on all services available in the network. In addition, nodes can leave or join the network dynamically. Consequently, for nodes to know about the arrival or departure of a node, leasing and notification mechanisms will be used [GUR 03]. However, periodic broadcasting of messages can overload the network [LIU 03].

With the incremental strategy, each node publishes its service to nodes in a set zone at each step and can increment this zone. A user also wanting to locate a service sends a request to nodes present in this zone. He can also extend this zone if the request is not satisfied.

In the uniform memoryless strategy, a node publishes its service to a group of nodes randomly chosen. A user wishing to locate a service sends his request to a group of randomly chosen nodes. With the with memory strategy, the user selects a series of nodes, at each step, among nodes not reached in the previous step.

DEAPspace [NID 01] is a system dedicated to services discovery in mobile ad hoc networks. It is based on the pull technique for services information dissemination. More precisely, each node maintains a service lookup (or a cache) to store information on services. To disseminate services information, each node periodically sends information on its services to all network nodes. Each node must indicate the time period during which the service is valid. Before the expiration of this delay, it must renew dissemination of services information, otherwise it is considered by the other nodes as unavailable and will be deleted from their cache. When a node receives a service dissemination message, it logs information concerning this service in its directory and then broadcasts it to other nodes. This procedure is repeated until information about services reaches all network nodes.

When a node receives its service information dissemination message, it increases its validity time in order to increase the chance of being received by all network nodes. This dissemination technique based on service lookup broadcast decreases the service location time. However, periodic broadcasting of messages increases network resource usage.

Konark [HEL 03] is another system dedicated to service discovery in mobile ad hoc networks by using the push and pull techniques. In other words, each node maintains a service lookup (or a cache) to store information on services. This cache is organized in the form of a tree to categorize services and to facilitate search and dissemination of information on these services. These services are represented by the XML description language, similar to Web Service Description Language (WSDL). When a user wishes to locate a service, he initiates a location message and sends it in multicast. When a node receives this message, it looks in its service lookup to see if it can respond to the request locally (if the service lookup has the location of the requested service). If that is the case, it responds to the request and sends a dissemination message to inform other users requesting the same service of its location. When the user receives the dissemination message, he stores it in his service lookup.

HAID (*Hybrid Adaptive Protocol for Integrated Discovery*) is a system proposed in [CHA 05] based on the use of the push and pull techniques where information on services is disseminated to the nodes in a given zone. This zone is determined by node neighbors at a single hop and it can vary dynamically according to requests of users requiring a given service. All the nodes of each region store information on services offered by nodes of this region. Each node periodically sends a dissemination message of service information on services to node neighbors at one hop. This message contains the following information: type of service, service provider ID, identifier of the node receiving this message and region of dissemination. This region corresponds to the lifespan of message propagation, initially set at 1. When a node receives this message, it searches in its directory. If the disseminated service exists, it updates information concerning this service, otherwise it adds it to its service lookup.

A user wanting to locate a service sends a location message with a time to live (TTL). When a node receives this message, if it does not have information on the requested service, it increases the TTL value and broadcasts the message. Otherwise, it notifies the requesting node by sending a response containing provider identifier and the number of hops to reach it. The message follows the opposite route until it reaches the user. If the TTL value is equal to zero, the request is not rebroadcast in the network.

Another system using the push and pull techniques for services discovery in a mobile ad hoc network was proposed in [MOH 04]. Each node periodically broadcasts information on its services to other network nodes. Each user wanting to search for a service which has not yet received a dissemination message concerning this service broadcasts his request in the network. To limit periodic broadcasting of dissemination messages, each server listens to its carrier to determine the number of nodes having disseminated their services or request a service location. If this number is lower than a given threshold, the server sends a service dissemination message, otherwise it waits for a given period of time. After the time has expired, it listens to its carrier once more and repeats the same procedure.

When a node receives a request, if it has information concerning this service, it responds to the user by broadcasting the response. The response broadcast enables other users requiring the same service to know of its location. When a user wants to locate a service, he listens to his carrier to determine the number of nodes having been disseminated or to search for a service and proceeds in the same manner as a node wanting to disseminate its service.

Gnutella [AND 01, JAN 02] is an unstructured system made up of a group of nodes, also called peers, for file transfer between users. The Gnutella system eliminates the need for a service lookup server to index user files [FEL 03]. A node joins the network by connecting to another node already connected. It then starts getting to know other network nodes by receiving requests or responses from them. The request resolution principle in the Gnutella system is accomplished in the following way: a user wanting to locate a file sends a request to neighbors. If the neighbors cannot respond to the request, they send the request to their own neighbors. If a node has the file involved, it notifies the user by sending a response. Otherwise, the search is stopped when a given search threshold (a given depth) has been reached. More precisely, requests have a limited hops to live (HTL) time [LUA 05]. This value is expressed in terms of maximum number of nodes to reach so as to avoid circulating in the network indefinitely. It is often set to 7 to start, and is decremented by each neighbor [FEL 03]. If the HTL value is equal to zero, the request is no longer rebroadcast in the network.

The main drawback of the push technique is the important number of messages transmitted for locating a file which overloads the network [FRA 02, LI 02]. In addition, requests can fail simply by expiration of their TTL before covering the whole network [DEV 03]. In other words, using HTL does not guarantee locating a service available in the network. To solve this problem, an approach using a location mechanism based on the use of a dynamic HTL (expanding ring or dynamic TTL setting) was proposed [LI 02, RIS 04]. More precisely, a request is transmitted with a small HTL. If no service is found, a new request is transmitted with a larger HTL and so on until the service requested is found or until HTL values flooding the

network are reached. This approach guarantees the location of an available network service but does not make scaling possible because of the high number of messages used for locating a service [LI 02].

6.3.2. Random walk-based mechanism

To avoid network overload caused by the flooding mechanism with HTL, a method proposed in [LV 02] consists of using a location algorithm based on parallel random walk, or K-random walk in an unstructured system. The objective of this approach is to locate a file without using HTL. The principle of a single random walk is that a node wanting to locate a file sends a message executing a random type walk, or in other words its route is not predetermined. When the requested service is located, it comes back to the initiating node. However, this approach increases service location time [RIS 04]. The use of the k-random walk can decrease file location time. In this case, the node wishing to locate a file sends k messages which work in parallel. In [GKA 04, LV 02], the authors illustrate that location based on the use of parallel random walk in an unstructured network supports scalability compared to using a flooding mechanism with HTL, but the cost in terms of necessary time for request resolutions can be high [LI 02, LV 02]. The solution consisting of duplicating network files, as was proposed in [COH 02], is not appropriate since it raises the issue of their update in a dynamic network.

6.4. Comparison between structured and unstructured systems

In structured systems based on the centralized indexing mechanism, request resolution relies on the use of a directory (or service lookup) that groups information on services available in a small-sized network. However, the fact that all information concerning services is centralized in a single place is a major flaw in this model in terms of scalability. In addition, failure of the server storing services information would make it impossible to locate these services. The use of a decentralized indexing mechanism can solve the scalability problem. However, the cost of information updates for services frequently entering or leaving the network becomes high [CHO 05].

Structured systems based on the use of the hash mechanism were initially proposed for file transfer and access and disk space sharing. They require the development of an overlay network for locating available files in the network [FRA 02, MIL 02, SCH 03]. However, when one or more nodes simultaneously enter or leave an overlay network, a maintenance algorithm becomes necessary [LIB 02]. In addition, these systems use a mechanism of regular dissemination of file information. Consequently, the high number of messages generated for overlay

network maintenance and periodic services information dissemination can overload the network.

Unstructured systems do not use an approach based on using service lookups. Two techniques based on a flooding mechanism can be used for service discovery in the network: push and pull techniques. In the push technique, the user periodically broadcasts its discovery request for services available in the network. With the pull technique, it is the servers who periodically broadcast information on their services enabling users to know what services are available in the network. The advantage of this type of system is that they require no structure between network nodes. Therefore, the nodes can freely join or leave the network without any coordination with other nodes. Service location is done through a flooding technique with TTL or HTL. In other words, requests have a limited lifespan in terms of hops so that they do not circulate in the network indefinitely. However, requests can fail simply by expiration of their time before covering the whole network. To solve this problem, an approach was proposed based on the use of dynamic HTL [LI 02, RIS 04]. This approach guarantees the location of a service available in the network but does not enable scalability because of the high number of messages used for locating a service. To avoid network overload by messages, a method based on the use of parallel random walks for request resolution was proposed [LI 02, LV 02]. However, cost in terms of time required for request resolution can be high. The solution consisting of duplicating information concerning services in the network, as was proposed in [COH 02], is not appropriate since it raises the problem of their update in a dynamic network.

A self-organizing and self-adaptive method, called AFFINITY [BAK 03, BAK 05], was proposed for service discovery in dynamic networks. Self-organization is implemented by the creation of server communities (i.e. nodes able to offer services). A community is a network of affinities connecting nodes together that are capable of providing a service together. Self-adaptation is implemented by the automatic adjustment of affinities in relation to evolution and dynamic modifications of the number and availability of resources and services, and user requests. More precisely, affinity networks self-organize and dynamically adapt to the type and frequency of user requests through a mechanism of learning and reinforcement of node affinity connections. In other words, the mechanism makes it possible to determine corrections required for adapting to new network conditions and future user requests.

6.5. Self-organizing and self-adaptive approach

Contrary to services discovery methods proposed in the literature, this approach is implemented through an adaptive middleware inspired by the human immune

system [GAB00, GAB06, BAK 03, BAK 05, BAK07]. The analogy with the immune system is done in the following way. A user request is considered as an antigen attacking the network. The middleware reacts by developing an immune response to eliminate the attacking antigen i.e. to satisfy the user's request [GAB 00]. The immune system presents a set of operational principles, such as self regulation, self-organization, cooperation and adaptive memory [SOM 97, WAT 99]. These characteristics will enable the middleware to develop decentralized and adaptive solutions in a dynamic and uncertain environment.

We should remind the reader that the operation principle of the immune system is the principle of selection by cloning. This is put in place by idiotypic networks and by adaptive memory. In this approach, the analogy is: the principle of selection by cloning is developed by server communities representing idiotypic networks. Adaptive memory is represented by the request response reinforcement mechanism.

The services discovery approach proposed is made up of two processes which can work in parallel. The first one concerns the development of server communities (or affinity networks) and the second one involves user request resolution.

We begin this section by presenting the mechanism of server community development. Then, we will present request resolution mechanisms.

6.5.1. Server community development approach

Jerne [STE 94, WAT 99] introduced the idea that the immune system's B-cells communicate together and create affinity networks (or idiotypic networks) even in the absence of antigens. More precisely, B-cells are not isolated cells, but are connected by chains of stimulation/deletion in which a B-cell is considered an antigen for another B-cell. Therefore, we can consider that an idiotypic network can develop itself in two ways: a proactive way in the absence of an antigen and a reactive way by the introduction and stimulation of this antigen.

By analogy with the immune system, the creation of a community can be done in a proactive or reactive way. The proactive creation is initiated by servers without the presence of user transmitted requests. The requests transmitted correspond to the antibody of the immune system. Creation can be done in a reactive way during a user request resolution.

The process of proactive creation is a community detection process enabling servers (in the sense of nodes able to offer resources) to mutually discover each other with the goal of creating relations of affinity. To implement this detection process, we adopt a technique based on mobile agents executing parallel random

walks [BAA 03, BRO 89] for discovery of services present in the network. Note that a service is made up of a group of resources. A node having a part of these resources cannot provide this service individually, but, because of the process of dissemination through mobile agents, nodes with resources necessary to implement the service, will mutually discover each other and will be able to provide this service together. More precisely, a community of these nodes is created to represent the service.

Generally, a community emerges from the cooperation between network nodes through mobile agents to represent a service available in the network by creating affinity connections. Three types of agents participating in the community development process are:

- mobile agents, called *aAgents*, representing immune system antibodies. These agents enable implementation of the community detection process;
- resource agents, called *BAgents*, representing resources available in the network. These agents correspond to B-cells in the immune system;
- server agents or nodes, called *SAgents*, representing network nodes. These agents correspond to T-cells in the immune system.

6.5.1.1. *SAgent server agent*

An *SAgent* is associated with each network node. It can belong to one or more communities and has two important roles. The first role concerns node resource management. In particular, when a node enters the network, *SAgent* creates *BAgents* associated with its resources. When a node resource must be deleted, the *SAgent* informs *BAgent* representing this resource. The second role of *SAgent* concerns the reception of mobile antibody *aAgents* from a proactive creation. The *SAgent* activates appropriate *BAgents* for the implementation of affinity connections. More precisely, when *SAgent* receives the activation message from an *aAgent* to develop a service, it communicates back the list of required resources to build the service if they exist. It also communicates back the list of its immediate neighbors to enable it to select a following node to visit.

6.5.1.2. *BAgent resource agent*

Proactive creation is initiated by nodes without the presence of requests transmitted by users. *BAgents*, created by *SAgents* to represent resources, are responsible for this process through two roles. Their first role involves the creation of mobile antibody agents to stimulate corresponding immune responses to the creation of node communities (i.e., the server). These communities represent initial services predefined for proactive creation. More precisely, when a *BAgent* is created and initiated by *SAgent*, it in turn creates a group of antibody *aAgents* for the proactive creation of services belonging to the group of initial services. The second role of *BAgent* concerns the development of affinity connections between its

resource and other resources discovered by *aAgents*. In other words, when *BAgent* receives a message from *aAgent* to create an affinity connection with another *BAgent*, it creates this affinity connection in its local table of connections with the affinity value, which is a real and positive number initially chosen in a random way. This value will then be determined based on the number of requests that covered this connection in search of the service, as we will see in section 6.5.2. We should note that an affinity connection is deleted when its affinity value becomes lower than or equal to 0. The value corresponding to an affinity connection can decrease and go over 0 when the community to which it belongs is no longer used by user requests.

6.5.1.3. Mobile *aAgent*

An *aAgent* is created by a *BAgent* for the development of a community representing a given service. The role of mobile *aAgent* is to create a community of nodes representing a given service. This agent corresponds to an immune system antibody. It moves randomly between network servers and activates *SAgent* encountered in order to stimulate the creation of affinity connections between appropriate *BAgents*. More precisely, when this agent is created, it sends an activation message to *SAgent* requiring a message containing the list of resources corresponding to the service and the list of the network's direct neighbors. When it receives this message, in the case where the server does not have the resources requested by the service, it moves to a randomly chosen neighboring server. Otherwise, it activates each *BAgent* corresponding to resources required for the creation of the service by sending an activation message to it requesting a message containing affinity connections from activated *Bagents* corresponding to the service to create.

In the case where activated *BAgents* do not yet have affinity connections involved in the creation of service community, the agent randomly chooses a *BAgent* among them and sends to it the message of creation of affinity connection. The agent then moves to a neighbor node randomly chosen from the list of direct neighbors. If connections corresponding to the service to create exist but do not yet include a link between *BAgents* from the current server and the last server visited, then it sends a development message to the last *BAgent* that has participated in creating the community representing the service. Then, it chooses a connection with the highest affinity value and moves to the corresponding server. The *aAgent* is eliminated when its lifespan has expired.

In the natural immune system, an antibody generated by a B-cell is considered an antigen for other B-cells. In other words, an antibody is an antigen which does not correspond to a request coming from a user but is proactively initiated by a *BAgent*. It is important to note that the mobile *aAgent* can clone itself during its random

walk. The self regulation approach proposed in [AMI 02, BAK 05] is used to regulate *aAgent* population size in the network.

By analogy with the immune system, a community of servers represents an idiotypic network, server resources correspond to B-cells and a user request for a service corresponds to an antigen. In the natural immune system, when an antigen attacks the human body, B-cells activated by this antigen are cloned and produce antibodies until the antigen is eliminated; this is the “primary” response. Thus, the presence of these B-cells is reinforced by a cloning mechanism and when the same antigen attacks the body once more, the immune system reacts more quickly to this new exposition; this is the “secondary” response. On the other hand, B-cells whose presence is not reinforced by antigens wind up being eliminated [BAL 00]; this is the apoptosis. In other words, B-cells not reinforced by exposition to the same antigen are gradually eliminated and corresponding idiotypic networks disappear.

Similarly, in this approach, when a community exists in the network but is not used, it must be undone (i.e. affinity connections must be deleted). In addition, mobile *aAgents* creating this community must also be eliminated from the network. Otherwise, these agents will continue to cover the network and build communities to represent their service. As we will see in the following section concerning request resolution, the resolution approach is based on the reinforcement of affinity connections from a community. This reinforcement is done by mobile agents representing requests when this community corresponds to the response requested. Otherwise, affinity connections are weakened. More precisely, values of affinity connections from a community of servers progressively decrease, according to an equation that will be described in the following section, as request agents not requesting the represented service cover servers in this community.

6.5.2. Request resolution

Request resolution must be executed from a greedy resolution strategy based on the selection of the best neighbor within a community with affinity connection reinforcement. More precisely, a mobile agent representing a request (or the antigen), once created, initiates a random walk in the network until it reaches a node that is able to provide one or more resources requested to make up the service. If the node encountered is the entry point to a community corresponding to the service requested, the agent executes a greedy run within this community according to affinity connection values. It uses a connection when its affinity value is higher. In addition, during its route, it reinforces the value of selected affinity connections and decreases those from non selected connections [BAK 03, BAK 05]. We will call this reinforcement process *local reinforcement* of connections. When the agent ends its route within the community and constitutes the list of servers providing the service,

it proceeds to the reinforcement of the selected route. More precisely, it uses the opposite route to the entry point by once more reinforcing the chosen links. We will call this second reinforcement process *global reinforcement*.

The objective of the local reinforcement process is to allow servers to adapt their affinity connections in pairs according to available services in the network and based on user requests. This step is vital in this self-organizing approach since it makes it possible for the system to adapt to dynamic network modifications, to the number and availability of resources and to user requests.

The objective of the global reinforcement process is to enable the selection, throughout the community, of an appropriate route for the user request. In other words, remember that a route within a community corresponds to a list of nodes liable to provide the service together. Since there are several possibilities of choosing a route, the global reinforcement process makes it possible for one to emerge based on user requests. This step is also vital in this self-organizing approach since it enables the emergence of adaptive solutions to requests transmitted by users.

In a more formal way, request resolution is implemented by using affinity connections created during the community development step. A community is an affinity network which can be represented by a direct weighted graph $G(S,L)$, where S is the group of nodes and L is the group of logical connections representing the affinity relations between these nodes. A connection between node i and node j is characterized by weight $m_{ij}^{(s)}$ representing the affinity between these two nodes within the community representing service s .

6.5.2.1. Local reinforcement mechanism

Affinity $m_{ij}^{(s)}$ between node i resource (i.e. a *BAgent* of i) is a resource of server j (i.e., a *BAgent* of j), in relation to a service s , is adjusted at step $k+1$ in the following way:

$$m_{ij}^{(s)}(k+1) = m_{ij}^{(s)}(k) + \Delta m_{ij}^{(s)}(k) = m_{ij}^{(s)}(k) + \mu(\text{satLocal}_{ij}^{(s)} - f(m_{ij}^{(s)}(k))) \quad [6.1]$$

f being logistic function $\frac{1}{1+e^{-m_{ij}}}$, and μ being the constant of the proportionality

included between 0 and 1. $\text{satLocal}_{ij}^{(s)}$ is local satisfaction between *BAgent* i and *BAgent* j in relation to request s . This value weighs 1 for positive satisfaction or in other words, the agent request arriving in server i selects server j among neighbors

of i providing the following requested resource. $satLocal_{ij}^{(s)}$ weighs 0 for negative satisfaction.

The use of the logistic function enables affinity value $m_{ij}^{(s)}$, when close to 0, to quickly increase when $satLocal_{ij}^{(s)}$ is equal to 1 and to quickly decrease when $satLocal_{ij}^{(s)}$ is equal to 0.

To illustrate this reinforcement mechanism, we consider the following example in which a request agent searches for a service s made up of three resources r_1 , r_2 and r_3 (i.e., $s = (r_1, r_2, r_3)$). Let us presume that the request agent arrived at node s_1 using resource r_1 (see Figure 6.2). The request agent thus considers that s_1 is the entry point in a community providing requested service s and has affinity relations with its neighbors s_2 , s_3 and s_4 . These neighbors provide resources r_2 , r_3 and r_4 respectively. Let's further presume that initially m_{14} weighs 0.4, m_{13} weighs 0.3 and m_{12} weighs 0.2. Node s_4 does not belong to the community providing the requested service. The request agent cannot choose s_4 and consequently $satLocal_{14}^{(s)}$ is equal to 0. On the other hand, both s_2 and s_3 nodes can satisfy the request. Consequently, $satLocal_{12}^{(s)}$ and $satLocal_{13}^{(s)}$ are equal to 1. Between these last two, the request agent selects the affinity connection with the highest value, i.e. connection m_{13} . Values of these connections are then adjusted based on equation 1 and consequently m_{14} is decreased and becomes -0.1987 , m_{12} is increased and becomes 0.6502 and m_{13} is increased and becomes 0.7256.

The request agent, having chosen node s_3 as the next step in its route for request resolution, moves into this node and continues the resolution process in the same way. The request agent then uses the connection between s_3 and s_6 for the selection of the last resource r_2 and locally reinforces the value of affinity m_{36} which becomes equal to 1.0318.

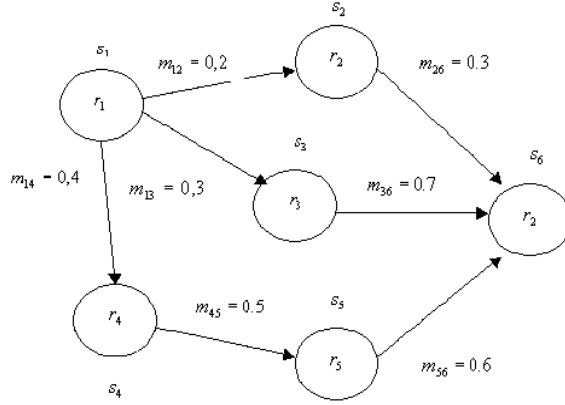


Figure 6.2. In this example, the request agent searches for a service made up of three resources r_1 , r_2 and r_3 . Node s_1 has affinity connections with its neighbors s_2 , s_3 and s_4 . Only s_2 and s_3 belong to the requested service community $s = (r_1, r_2, r_3)$. When the request agent chooses connections m_{13} and m_{36} , affinity values become $m_{12} = 0.6502$, $m_{13} = 0.7256$, $m_{14} = -0.1987$ and $m_{36} = 1.0318$

We should note that, despite the fact that it has resource r_1 , if server s_1 was not already part of a community providing service $s = (r_1, r_2, r_3)$, the proactive development process is launched as described in section 6.5.1.

6.5.2.2. Global reinforcement mechanism

When the agent ends its route within the community and makes up the list of nodes providing the requested service, it then proceeds to global reinforcement of route selected φ (i.e., affinity connections between selected nodes). More precisely, it uses route φ in the opposite direction until it reaches the entry point by once more reinforcing the chosen connections. Value variation of affinity connections is calculated as follows:

$$\Delta m_{ij}^{(s)}(k) = \mu(\text{satGlobal}_{\varphi}^{(s)} - f(m_{ij}^{(s)}(k))) \quad [6.2]$$

$\text{satGlobal}_{\varphi}^{(s)}$ being the global satisfaction in relation to the resolution of request s . This value weighs 1 for a positive satisfaction or in other words, the request agent has found all requested resources, or 0 for negative satisfaction. That is the case when delay (TTL) granted in the request agent expired before it has been able to find all requested resources.

To illustrate this global reinforcement mechanism, we will take the previous example in which a request agent searches for a service s made up of three resources r_1 , r_2 and r_3 . Once resource r_1 is found in node s_1 , resource r_3 in node s_3 , and remaining resource r_2 in node s_6 , the request agent uses the route made up of selected links in the opposite direction by once more reinforcing values based on equation [6.2.] Values m_{36} , m_{13} are incremented and become equal to 1.2945 and 1.0518 respectively (see Figure 6.3).

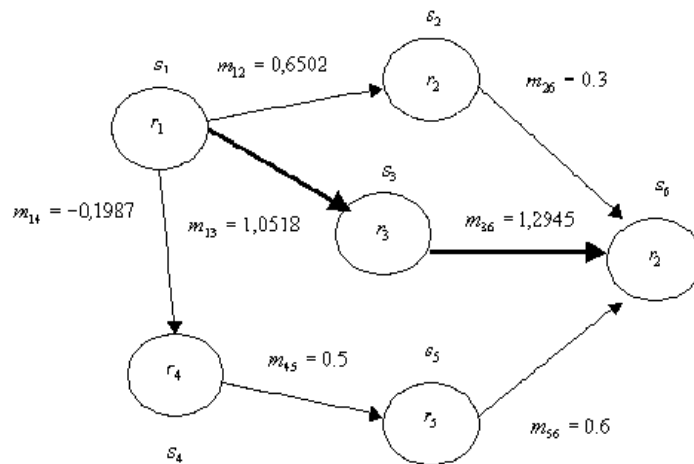


Figure 6.3. In this example, when the request agent finds all requested resources r_1 , r_2 and r_3 , it uses the opposite route and reinforces the values of affinity connections a second time. In this way, the values of selected links become $m_{13} = 1.0518$ and $m_{36} = 1.2945$

6.5.2.3. Types of agents

In the process of creating affinity networks between nodes, we have used all three types of agents $aAgent$, $BAgent$ and $SAgent$. For request resolution, the following two types of agents are added:

- user agents, called $UAgent$, representing users and initiating the request resolution process;
- mobile agents, called $AAgent$, for the implementation of the resolution by triggering an immune response. These agents correspond to the immune system's antigens.

6.5.2.3.1. User agent (UAgent)

A *UAgent* is associated with each user connected to the network. The *UAgent* has two roles. The first one involves receiving user requests. In other words, for each request transmitted by the user, *UAgent* creates an *AAgent* associated with it to initiate the resolution process. The second role of *UAgent* involves receiving responses from these *AAgents*. More precisely, when *UAgent* receives a request from the user, it creates an *AAgent* for its resolution and waits for the result containing the list of nodes providing a part of or all resources requested. Once it receives this message, the result is then sent to the user. Then, *UAgent* sends a termination message to *AAgent*.

6.5.2.3.2. Request agent (AAgent)

This agent is created by *UAgent* for the resolution of a request. The role of this mobile *AAgent* is to resolve a user's request. Once created by a *UAgent*, this agent initiates a random walk in the network in search of nodes liable to provide one or more resources making up the service. More precisely, when this agent reaches a node, it transmits an activation message to *SAgent* requesting a message containing the node's and its neighbors' list of resources.

In the case where the node encountered has no resource to satisfy the request, *AAgent* continues its random walk by choosing the next node to randomly visit among the current node's neighbors.

In the case where the node encountered is the entry point of a community corresponding to the requested service, *AAgent* executes a greedy route within this community based on affinity connection values. In other words, *AAgent* uses the connection with the highest affinity value to continue its route. In addition, during this route, *AAgent* reinforces the value of selected connections and decreases the value of non selected connections. When *AAgent* ends its route within the community and makes up the list of nodes providing the service, it sends to *UAgent* the response containing the list of resources found. Then, it proceeds to a second reinforcement of the selected route. More precisely, *AAgent* uses the opposite route until it reaches the entry point by once more reinforcing the chosen links.

In the case where the node encountered has one of more resources liable to satisfy the request and that this node is not the entry point of a community representing the requested service (i.e., none of its *BAgents* with resources has affinity connections), it stimulates reactive creation by sending a stimulation message to a *BAgent* randomly chosen among selected *BAgents*. In order to do this, the *BAgent* will create a mobile *aAgent* as presented in section 6.5.1. *AAgent* then sends a response to *UAgent* to indicate that the service is not represented.

6.5.2.3.3. Server agent (SAgent)

We might remember that an *SAgent* represents a network node. The roles of this agent in the community development process were described in section 6.5.1. It also has a role in the resolution process consisting of communicating to *AAgent* the list of its resources (or *BAgents*). More precisely, when it receives an activation message from an *AAgent* for a request resolution, it responds by sending a message containing the following information: a list of its resources, its identifier and the list of its direct neighbors.

6.5.2.3.4. Resource agent (BAgent)

We should also remember that *BAgents* represent resources offered by network nodes. In the resolution process, *BAgents* play two roles. The first role involves value adjustment of affinity connections with other *BAgents* within a community. The second role, activated by an *AAgent*, involves reactively creating node communities. In other words, *BAgent* can initiate mobile antibody *aAgents* for the creation of a community as presented in section 6.5.1.

It should be noted that the reinforcement or attenuation of affinity connections between nodes is based on value modification of these connections by using equations [6.1] and [6.2]. This reinforcement of affinity connections is at the root of the emergence of the most widely used routes and their quick selection during future request resolution. In fact, it is important to note that, because of this learning mechanism, when a service is often requested, the reactive development of new communities and the presence of those already existing are reinforced. Consequently, the encounter between a request searching for a popular service, and communities representing it will be done much faster.

It is also important to note that this affinity connection reinforcement constitutes the self-organizing memory of this self-adaptive approach. This also corresponds, by analogy, to the adaptive memory of the immune system enabling the development of faster *secondary* immune responses to antigens previously encountered.

6.6. Simulation results

The simulations presented below have been developed by Network Simulator (NS2) [NET]. NS2 is a simulator for generating a network and simulating communications between its nodes. To accomplish our simulations, we have generated a network of 100 nodes. Each node has one resource. There are 10 types of resources randomly distributed among the nodes.

The first simulation consists of comparing request resolution time using two strategies. In the first strategy, there are no affinity connections between servers, and agents execute random walks in the network for request resolution. In the second strategy, the creation of communities is done but the reinforcement mechanism of affinity connection values is not applied. In this case, request agents execute a random route within these communities. Requests are randomly generated from all 10 resources.

The simulation result presented in Figure 6.4 shows that the creation of communities, with a request resolution based on a random route within these communities, improves request resolution time compared to a strategy of resolution based only on random walk in the network and with no community creation.

The objective of the second simulation is to compare request resolution time when the value reinforcement mechanism of affinity connections is applied (resp. is not applied). When the reinforcement mechanism for affinity connection values is not applied, the request agent executes a random route within the community instead of a greedy route with reinforcement. In addition, affinity connections are not modified during both routes. More precisely, affinity connections between servers are not weighed by adjustable values.

The simulation result presented in Figure 6.5 shows that the use of the reinforcement mechanism improves request resolution time compared to a resolution strategy only based on random walk within created communities.

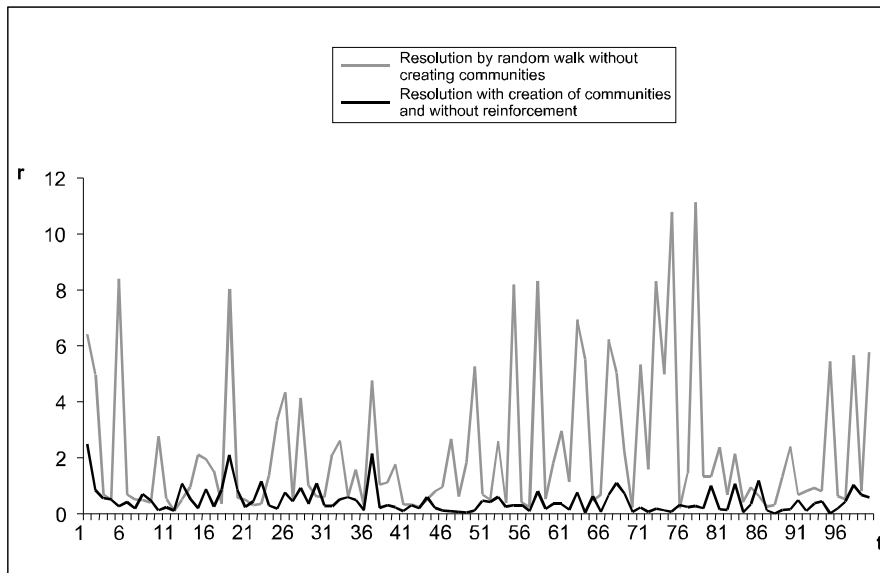


Figure 6.4. Comparison in terms of resolution time (r), with or without creation of server communities

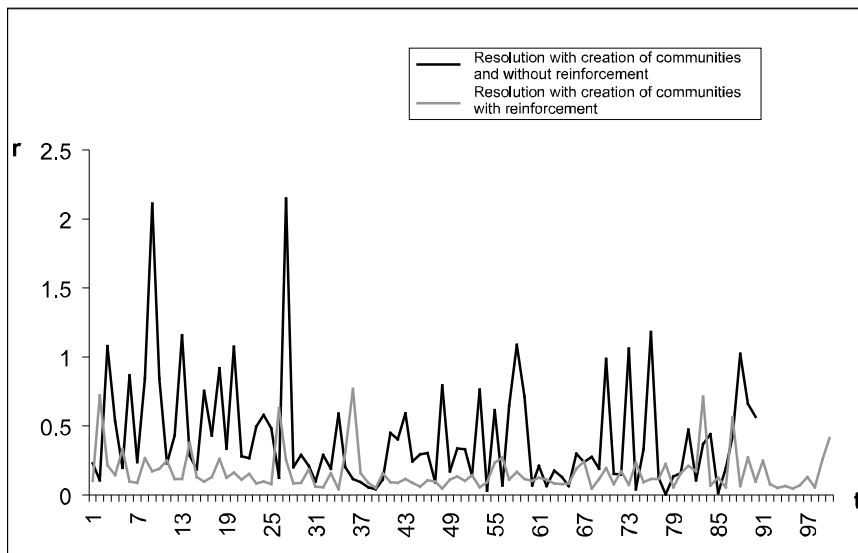


Figure 6.5. Comparison in terms of resolution time (r), with and without reinforcement of affinity connections

6.7. Conclusion

In this chapter, we presented the different systems of services discovery in the literature for implementing ubiquitous computing. They are classified into three families. One family groups systems based on structural organization such as centralized or decentralized indexing and hash. The second family groups systems not based on a structural organization. They use push and pull mechanisms or random walk. The third family groups self-organizing and self-adapting systems for the discovery of services available in a dynamically and randomly evolving network.

More precisely, the approach presented in this chapter is a self-adaptive methodology for a network in which the addition or subtraction of resources and servers, and the variation of the type or frequency of requests is done randomly and in an unpredictable way. Self-organization is implemented by the autonomous creation of server communities (i.e. nodes) to represent services available in the network. These communities can be created in a proactive or reactive manner. Servers belonging to one community are linked by affinity connections. Self-adaptation is implemented on one hand by the maintenance of these links in relation to the evolution and dynamic modifications of the number and availability of resources and services, and on the other hand by the adjustment of these links' values in relation to user requests. This adjustment by reinforcement of affinity connections constitutes self-organizing memory of this self-adaptive approach made up of the experience acquired during past request resolutions, enabling the middleware to quickly solve future user requests.

6.8. Bibliography

- [AMI 02] AMIN K.A., MIKLER A.R., "Dynamic Agent Population in Agent-based Distance Vector Routing", *Second International Workshop on Intelligent Systems Design and Applications*, Atlanta, GA, USA, August 2002.
- [AND 01] ANDERSON K., Analysis of the Traffic on the Gnutella Network, CSE222 Final Project, <http://www.cs.ucsd.edu/classes/wi01/cse222/projects/reports/p2p-2.pdf>, 2001.
- [AZO 02] AZONDEKON V., Service Selection and Association Facilitation in a Network, M.Sc.A. Thesis, Génie Electrique and Génie Informatique Department, University of Sherbrooke, Sherbrooke, January 2002.
- [BAA 03] BAALA H., FLAUZAC O., GABER J., BUI M., EL-GHAZAWI T., "A Self-Stabilizing Distributed Algorithm for Spanning Tree Construction in Wireless Ad Hoc Networks", *Journal of Parallel and Distributed Computing*, vol. 63, no. 1, p. 97-104, January 2003.

- [BAK 03] BAKHOUYA M., GABER J., Self-Adaptive and Self-Organising System Inspired by the Immune System for Ubiquitous Computing, Internal research report, Université de Technologies de Belfort-Montbéliard (UTBM), October 2003.
- [BAK 05] BAKHOUYA M., Approche auto-adaptative à base d'agents mobiles et inspirée du système immunitaire de l'Homme pour la découverte de services dans les réseaux à grande échelle, PhD thesis, University of Technologies de Belfort-Montbéliard, Belfort, 2005.
- [BAK 07] BAKHOUYA M., GABER J., Ubiquitous and Pervasive Application Design. Encyclopedia of Mobile Computing & Commerce, Eds. D. Taniar, Idea Group Pub, ISBN: 978-1-59904-002-8, Feb. 2007.
- [BAL 00] BALLEP P., Intérêts mutuels des systèmes multi-agents et de l'immunologie, applications à l'immunologie, l'hématologie et au traitement d'images, PhD thesis, Université de Bretagne Occidentale, Brest, January 28 2000.
- [BET 00] BETTSTETTER C., RENNER C., "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School*, September 2000.
- [BRO 89] BRODER A.Z., KARLIN A.R., RAGHAVAN P., UPFAL E., "Trading Space for Time in Undirected s-t Connectivity", *ACM Symposium on Theory of Computing*, p. 543-549, Seattle, WA, 1989.
- [CHA 05] CHANG-SEOK OH Y.-B.K., ROH Y.-S., "An Integrated Approach for Efficient Routing and Service Discovery in Mobile Ad Hoc Networks", *IEEE Consumer Communications and Networking Conference (CCNC 2005)*, Las Vegas, NV, 2005.
- [CHO 05] CHO C., LEE D., Survey of Service Discovery Architectures for Mobile Ad Hoc Networks, <http://www.cise.ufl.edu/class/cen5531fa05/files/ccho-dlee.pdf>, 2005.
- [COH 02] COHEN E., SHENKER S., "Replication Strategies in Unstructured Peer-to-Peer Networks", *ACM SIGCOMM'02 Conference*, vol. 32, no. 4, p. 177-190, 2002.
- [CZE 99] CZERWINSKI S., ZHAO B., HODES T., JOSEPH A., KATZ R., "An Architecture for a Secure Service Discovery Service", *Proceeding of ACM MobiCom'99*, Seattle, WA, 1999.
- [DEV 03] DEVERGE J.-F., Systèmes distribués de partage de données, Mémoire de DEA, IFSIC, Université de Rennes, Rennes, 2003.
- [FEL 03] FELTIN G., DOYEN G., FESTOR O., "Les protocoles Peer-to-Peer, leur utilisation et leur détection", *Cinquième Journées Réseaux JRES'2003*, Lille, <http://2003.jres.org/actes/paper.70.pdf>, 2003
- [FOU 04] FOUIAL O., Découverte et fourniture de services adaptatifs dans les environnements mobiles, PhD thesis, Ecole Nationale Supérieure des Télécommunications, 2004.
- [FRA 02] FRANÇOIS P., "Vers de nouveaux modèles peer-to-peer", *Mini-Workshop Systèmes Coopératifs*, <http://www.info.fundp.ac.be/ven/CISma/FILES/2003-pierre-FRANCOIS.pdf>, 2002.

- [GAB 00] GABER J., New Paradigms for Ubiquitous and Pervasive Computing, White paper, Université de Technologies de Belfort-Montbéliard (UTBM), September 2000.
- [GAB 06] GABER J., New Paradigms for Ubiquitous and Pervasive Applications, Proceeding of First Workshop on Software Engineering Challenges for Ubiquitous Computing, Lancaster, UK, 2006.
- [GAU 02] GAURON P., Topologies dynamiques pour les systèmes pair-à-pair, Training report from DEA Informatique distribuée, Université Paris-Sud-Orsay, Paris, 2002.
- [GKA 04] GKANTSIDIS C., MIHAIL M., SABERI A., “Random Walks in Peer-to-Peer Networks”, *INFOCOM*, <http://www.ieee-infocom.org/2004/Papers>, 2004.
- [GUR 03] GURGEN L., Découverte de données dans les réseaux mobiles, DEA training, ENSIMAG, Institut National Polytechnique de Grenoble, Grenoble, June 13 2003.
- [GUT 99] GUTTMAN E., PERKINS C., VEIZADES J., DAY M., “Service Location Protocol, Version 2”, *RFC*, no. 2608, June 1999.
- [HAU 05] HAUSPIE M., Contributions à l’étude des gestionnaires de services distribués dans les réseaux ad hoc, PhD thesis, Université des Sciences et Technologies de Lille, Lille, 2005.
- [HEL 03] HELAL S., DESAI N., VERMA V., LEE C., “Konark – A Service Discovery and Delivery Protocol for Ad-hoc Networks”, *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, New Orleans, 2003.
- [JAN 02] JAN M., Systèmes pair-à-pair de gestion de données à grande échelle: localisation et routage, DEA d’Informatique de l’IFSIC training report, Université de Rennes I, Rennes, 2002.
- [KLE 03] KLEIN M., KONIG-RIES B., OBREITER P., “Service Rings – A Semantic Overlay for Service Discovery in Ad Hoc Networks”, *14th International Workshop on Database and Expert Systems Applications (DEXA’03)*, p. 180, Prague, 2003.
- [KOZ 04] KOZAT U.C., TASSIULAS L., “Service Discovery in Mobile Ad Hoc Networks: An Overall Perspective on Architectural Choices and Network Layer Support Issues”, *Ad Hoc Networks*, no. 2, p. 23-44, Elsevier, Paris, 2004.
- [LI 02] LI C.W.B., Peer-to-Peer Overlay Networks: A Survey, Department of Computer Science, <http://comp.uark.edu/~cgwang/Papers/TR-P2P.pdf>, 2002.
- [LIB 02] LIBEN-NOWELL D., BALAKRISHNAN H., KARGER D., “Analysis of the Evolution of Peer-to-Peer Systems”, *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, p. 233-242, ACM Press, Monterey, CA, 2002.
- [LIU 03] LIU J.C., SOHRABY K., ZHANG Q., LI B., ZHU W., “Resource Discovery in Mobile Ad Hoc Networks”, *The Handbook of Ad Hoc Wireless Networks*, p. 431-441, 2003.
- [LUA 05] LUA E.K., CROWCROFT J., PIAS M., SHARMA R., LIM S., “A Survey and Comparison of Peer-to-Peer Overlay Network Schemes”, *IEEE Communications Survey and Tutorial*, vol. 7, no. 2, 2005.

- [LUO 03] LUO H., "Performance Evaluation of Service Discovery Strategies in Ad Hoc Networks", *Master of Science, School of Computer Science*, Carleton University, Ottawa, Ontario, 2003.
- [LUO 04] LUO H., BARBEAU M., "Performance Evaluation of Service Discovery Strategies in Ad Hoc Networks", *Second Annual Conference on Communication Networks and Services Research (CNSR'04)*, p. 61-68, Fredericton, 2004.
- [LV 02] LV Q., CAO P., COHEN E., LI K., SHENKER S., "Search and Replication in Unstructured Peer-to-Peer Networks", *ACM Sigmetrics 2002*, <http://citeseer.ist.psu.edu/lv02search.html>.
- [MAL 02] MALKHI D., NAOR M., RATAJCZAK D., "Viceroy: A Scalable and Dynamic Emulation of the Butterfly", *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, Monterey, CA, 2002.
- [MAT 01] MATA J.G., "Comparison of Bandwidth Usage: Service Location Protocol and Jini", *Master of Computer Science, School of Computer Science*, Carleton University, Ottawa, Ontario, February 2001.
- [MIL 02] MILOJICIC D.S., KALOGERAKI V., LUKOSE R., NAGARAJA K., PRUYNE J., RICHARD B., ROLLINS S., XU Z., *Peer-to-Peer Computing*, Research report no. HPL-2002-57, HP Labs, March 2002.
- [MOH 04] MOHAN U., ALMEROTH K.C., BELDING-ROYER E.M., "Scalable Service Discovery in Mobile Ad Hoc Networks", *NETWORKING 2004*, p. 137-149, 2004.
- [NID 01] NIDD M., "Service Discovery in DEAPspace", *IEEE Personal Communications*, no. 39-45, 2001.
- [NET] Network simulator NS-2, available on Information Sciences Institute's website: <http://www.isi.edu/nsnam/ns>.
- [PAR 05] PAROUX G., "Une plate-forme pour les échanges P2P sur les réseaux mobiles ad hoc", *MANifestation des JEunes Chercheurs STIC*, Rennes, France, 2005.
- [PER 98] PERKINS C., "Service Location Protocol", *ACTS Mobile Networking Summit/MMITS Software Radio Workshop*, Rhodes, Greece, June 1998.
- [PLA 99] PLAXTON C.G., RAJARAMAN R., RICHA A.W., "Accessing Nearby Copies of Replicated Objects in a Distributed Environment", *Theory of Computing Systems*, no. 32, p. 241-280, 1999.
- [RAT 01] RATNASAMY S., FRANCIS P., HANDLEY M., KARP R., SHENKER S., "A Scalable Content-Addressable Network", *Proc. ACM SIGCOMM'01*, San Diego, CA, 2001.
- [RIS 04] RISSON J., MOORS T., *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*, UNSW-EE-P2P-1-1 Technical Report, University of New South Wales, Sydney, Australia, September 2004.

- [ROB 00] ROBERT M., Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing, Urbana UIUCDCS-R-99-2132, Department of Computer Science University of Illinois Urbana-Champaign, citeseer.nj.nec.com/mcgrath00discovery.html, March 25 2000.
- [ROB 04] ROBINSON R., INDULSKA J., “A Complex Systems Approach to Service Discovery”, *DEXA Workshops 2004*, p. 657-661, 2004.
- [ROW 01] ROWSTRON A., DRUSCHEL P., Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, *Reading notes in Computer Science, N± 2218*, 2001.
- [SAI 05] SAILHAN F., ISSARNY V., “Scalable Service Discovery for MANET”, *Proceedings of the 3rd IEEE Int’l Conf. on Pervasive Computing and Communications*, Kauai Island, HI, 2005.
- [SCH 03] SCHÜRMAN S., Distributed Hashtables, reference no. 227747, 2003.
- [SHU 02] SHUKLA A., “Issues in Auto Service Discovery”, *A Study for the Course in Advanced Computer Networks (CS625)*, Doctor Dheeraj Sanghi, Instructor 2002.
- [SOM 97] SOMAYAJI A., HOFMEYR S., FORREST S., “Principles of a Computer Immune System”, *Proceedings of the Second New Security Paradigms Workshop*, p. 75-82, Little Compton, RI, 1997.
- [STE 94] STEWART J., “Un système cognitif sans neurones: les capacités d’adaptation, d’apprentissage et de mémoire du système immunitaire”, *Intellectica*, vol. 1, no. 18, p. 15-43, 1994.
- [STO 01] STOICAY I., MORRISZ R., LIBEN-NOWELLZ D., KARGERZ D.R., KAASHOEKZ M.F., DABEKZ F., BALAKRISHNANZ H., Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, <http://www.pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>, 2001.
- [WAT 99] WATANABE Y., ISHIGURO A., UCHKAWA Y., “Decentralized Behavior Arbitration Mechanism for Autonomous Mobile Robot Using Immune System”, in D. Dasgupta (ed.), *Artificial Immune Systems and Their Applications*, p. 186-208, Springer-Verlag, New York, 1999.
- [WEI 93] WEISER M., Ubiquitous Computing, www.ubiquitous.com/hypertext/weiser, March 1993.
- [XU 01] XU D., NAHRSTEDT K., WICHADAKUL D., “QoS-Aware Discovery of Wide-Area Distributed Services”, *First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, Brisbane, Australia, May 2001.
- [ZHA 01] ZHAO B.Y., KUBIATOWICZ J.D., JOSEPH A.D., Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, UC Berkeley, UCB/CSD-01-1141, 2001.
- [ZHA 02] ZHAO W., SCHULZRINNE H., GUTTMAN E., “mSLP-Mesh-enhanced Service Location Protocol”, *ICCCN 2000*, Internet draft draft-zhao-slp-da-interaction-07.txt, 2002.