Allia: Policy-based Alliance Formation for Agents in Ad hoc Environments

Olga Ratsimor, Dipanjan Chakraborty, Sovrin Tolia, Deepali Khushraj,
Gaurav Gupta, Anugeetha Kunjithapatham, Anupam Joshi, Timothy Finin
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250

May 3, 2002

1 Introduction

Recent years have witnessed tremendous growth in the necessity for pervasive mobile computing. Ad-hoc computing is a key component for this technology. Another field that is undergoing explosive growth is the area of intelligent agents. The merger of mobile computing and agent technologies has created a new class of problems that must be solved in order to make pervasive computing a reality. In this paper, we are proposing a solution to the problem that intelligent agents face in an ad-hoc computing environment, as outlined in the FIPA Call for Technology[5].

Ad-hoc networks are very dynamic and the network topology frequently changes as devices frequently join and leave the network. As a result, connection between devices cannot be guaranteed. The devices that are participating in such a network should be able to easily adjust to such frequent changes. The formation of compounds between agents is really important in ad-hoc networks. This is because it will enable different agents in heterogeneous mobile devices to collaborate and use each other's functionalities. Hence, one on of the meta goals behind formation of compounds is to enable agents to utilize services offered by other agents in an ad hoc environment. We expect mobile devices to range from being extremely resource-poor to being fairly resource-rich. In such cases, agents on resource-poor devices should be able to recognize and take advantage of the resources/services on the more powerful devices found in their vicinity.

An obvious solution for forming compounds is the one in which the resource-rich node maintains information about services and agents present in it's neighborhood. All other peer nodes would utilize the services of this resource-rich node in discovering services. In accordance to FIPA, this resource-rich node would provide the mandatory AMS and DF services, thus acting as an agent platform. There is an overhead associated in "electing" a resource-rich node to host an agent platform. Moreover, a new device moving into this structured compound would have to discover this agent platform and register it's services with it. Another major drawback with this approach is that if the resource-rich node moves out of the network, all other peer nodes would have to reconfigure themselves to the peer-to-peer mode. Highly dynamic environments where nodes keep moving in and out of the network would further aggravate this problem. For enabling service and agent discovery across multiple hops, "leaders" in different compounds have to federate with each other. In the context of ad-hoc networks, this would require those "leaders" to be in direct connectivity range of each other and this might not always be the case.

Our solution introduces a different approach towards handling the problem. We introduce the concept of peer-to-peer caching of services between nodes in an ad-hoc environment. The main goal of platform

formation is to provide an agent better access to services in the vicinity. Our solution describes a policy-based approach of achieving best-effort service discovery in ad-hoc networks.

We envision each participating device in an ad-hoc network will be able to run a lightweight version of the platform components like yellow pages and white pages service. Putting a lightweight platform on extremely resource-constrained devices might overload such devices. It is essential that each device have a set of bare minimum platform components like AMS and DF so that it can host an agent platform and operate even in regions of disconnections or stay independent. An agent in that case does not have to depend on other peer devices to host itself. The Main Container of LEAP [2] that contains the Agent Management System and the Directory Facilitator runs on standard laptops and H3870 iPaqs. Looking at a few types of less resource rich devices, we see that cellphones today have the capability to host advanced runtime environments (like java virtual machine) and the memory capacity of such devices is increasing. For example, most i-mode phones support Java Micro Edition [6], Mobile Information Device Profile [7] or Connected Limited Device Configuration [3] and have capability to store around 300K of data. Some phones have a run time heap of 180K (e.g. Samsung x350 series). However, we are aware that the current version of LEAP on PersonalJava [4] requires around 700K of dynamic memory and cannot run on cell phones. Cell phones can be considered one of the least resource-full devices amongst the plethora of heterogeneous mobile devices existing around us. Considering the curve at which the capabilities of these phones are increasing, it is not unreasonable to assume that cell phones and other devices of tommorow will have enough memory, processing power and energy so as to be able to host mandatory components of an agent platform. In addition to these platform components, the device will be able to host at least one agent. The main purpose for formation of compounds was to enable agents on a device to better utilize services/agents in its vicinity. Our approach concentrates on providing a solution to this issue without imposing the dependence that devices sharing a distributed compound will have on each other. The yellow pages service component registers only the services that are hosted on that local platform. The white pages service component of the platform registers only the agents that are hosted on the local platform. Each device has a policy that reflects the device capabilities, user preferences, application specific settings etc. A policy in a device governs the way it is going to present itself to the other platforms/devices in its vicinity. The policy governs the way in which the device takes advantage of resources/services in other platforms in its vicinity.

In our approach, every node advertises its services to other nodes in its vicinity in accordance with the policy governing that device. These advertisements are broadcasted. On receiving an advertisement, and agent decides based on its policy, whether to cache it or reject it. We introduce the concept of the "alliance" of a node. An *alliance* of a node refers to the set of nodes one or more of whose local services are cached by *it*. Thus, a node explicitly knows the member nodes in its alliance. However, each node does not know the alliances in which they are members. Whenever a node leaves a certain vicinity, and enters a new vicinity, it constructs its own alliance by listening to advertisements and becomes members of new alliances by advertising its services. Its exit from previous alliances is also implicit since nodes governing other alliances detect its absence and removes the node from their alliances.

The local policy of a node dictates the ways in which the node wants to advertise itself to peer nodes in its vicinity. The rate of advertisement is also controlled by the policy. The policy can specify algorithms to allow dynamic adjustments of advertisement rates based on mobility of nodes in a neighborhood. Policies also determine the number of members that a node can have in its alliance (by limiting the number of remote advertisements this node can cache). Alliance of a node can span across multiple hops. The advertisements in that case also are sent across multiple hops.

When an agent needs to discover a certain service, it first looks at its local platform to check whether that service is available. On failure, by looking at its own cache, it checks the members of its own alliance to discover the service. If the service is still unavailable, the source platform tries to broadcast or multicast the request to other alliances in its vicinity. The local policy determines the method (broadcasting or multicasting). Multicasting is used to prevent broadcast storms in the network. We use policy-based multicasting

where the node multicasts the request to other nodes in its vicinity where there is maximum chances of obtaining the service. A node on receiving a service request chooses, based on its policy to either process it or drop this request.

Our flexible approach towards alliance formation does not have the overhead of explicit leader election. Every device in this environment is self-sufficient. However, it utilizes resources/services in the vicinity whenever they are available. Dynamic network topology changes are automatically reflected in the alliances that are formed. Hence, Policy-based approach towards alliance formation buys us the advantages of a compound of agents. Using such policies, more traditional compounds like those utilizing explicitly elected leaders can also be achieved. Our architecture can also take user preferences into consideration. This is a major advantage since mobile users necessarily want to control the ways in which their own resources are utilized.

2 Device Architecture and Platform Components

Every mobile device in an ad-hoc network will run a lightweight platform comprising of the following components:

- *Generic Container*: A generic container that can host agents. The device must be able to run at least one agent.
- *Lightweight yellow pages service*: Services on the local device are registered with this component. This component is the Directory Facilitator (DF) component of the FIPA Agent platform specification.
- *Lightweight white pages service*: Local agents will register with this component. This component is the Agent Management System (AMS) component of the FIPA Agent platform specification.
- *Policy Manager*: It controls the behavior of the platform. Local policies can be used to control various aspects of the platform like advertisement frequency, caching policy etc.
- *Advertisement Manager*: It is responsible for advertising the local services that are hosted on a mobile device. Broadcasting is used to transmit these advertisements in the neighborhood.
- *Cache Manager*: It is responsible for caching service advertisements that have been received from the neighboring devices.
- *Forwarding Manager*: It is responsible for forwarding advertisements and "request for service" messages to other neighbors.

Figure 1 shows the different components in a single mobile device. We describe the new components briefly in the following section.

2.1 Policy Manager

The policy manager is responsible for enforcing policies to control platform behavior. Policies are specified and registered with the Policy Manager. The Manager is then responsible for ensuring that all components of the platform are in compliance with the specified policies. Using this mechanism, policies may be used to

- Restrict platform functionality in respect to device capability
- Specify caching preferences like refresh rate, replacement strategy etc.
- Specifying advertisement preferences like frequency, time-to-live, algorithms etc.

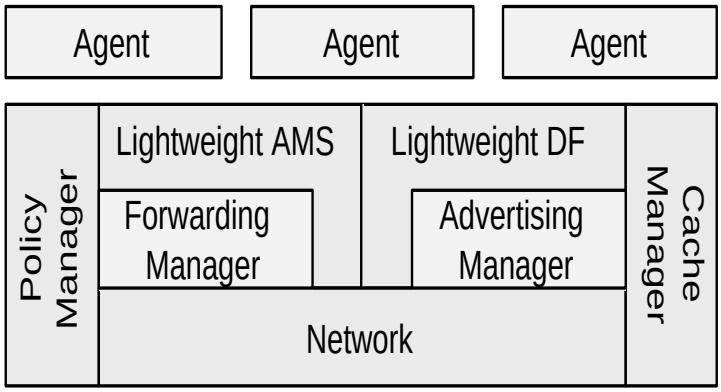

Figure 1: Device Architecture

- Specify priorities among services and how to share available resources among the services
- Specifying advertisement and request message forwarding algorithms
- Specify personal user preferences
- Specify application specific preferences
- Specify security restrictions like access rights and credential verification etc.

Components running on the platform would coordinate all their activities through the Policy Manager. Because of its interactions, this manager can monitor the activities of the different components and notify when policies are being violated.

2.2 Cache Manager

The Cache Manager handles service advertisements from neighboring devices. These advertisements could be services that are running on the neighboring nodes or advertisements that are being forwarded by these neighboring nodes. The Policy Manager coordinates with the Cache Manager to govern various characteristics like entry expiry, replacement strategy, cache size etc.

corresponding to this service. The cache maintains only hints. Due to the dynamic nature of the network topology, it is not feasible to guarantee that services found in the cache are currently reachable. Whenever there is a hit in the cache for a service that is not currently available, the cache entry is immediately deleted.

2.3 Advertising Manager

The Advertising Manager actively broadcasts service descriptions registered to the local DF. The Policy Manager controls the rate of advertisements. Various policies can be employed to adjust the rate of advertisement. We have described them in section 3.2. For example, if the network is fairly static, the advertisement rate can be slowed down. Also policy could be event driven, events here being the arrival or departure of nodes

in an alliance. Advertisements can also be assigned different priorities. The Advertisement Manager also controls the alliance diameter (explained in section 3.2.2).

2.4 Forwarding Manager

The Forwarding manager receives *Service Advertisements* and *Request for Service* messages. It decides based on the local policy whether to drop, or to propagate the advertisement. To prevent broadcast storms, this forwarding mechanism can use multicasting to selectively forward advertisements. For example, this can be used to forward advertisements to more active or resource-rich devices in the network.

3 Architecture Details

The overall design of our system is shown in the following diagram. In this section we describe the discovery, caching, advertising protocols and request routing protocols to be used between compatible mobile devices in ad-hoc networks. We describe the different protocols and principles in the following section:

3.1 Peer-to-Peer Caching

Every mobile device consists of an agent platform. We extend the functionality of a Directory Facilitator in the agent platform in each device with the help of a service cache for remote services. Each mobile device that is ad-hocly connected to different other heterogeneous devices in the network maintains a cache to store service descriptions of remote services. The Cache is controlled by a local Cache Manager. Each Advertisement Manager in a device, after a finite time interval (discussed later) broadcasts a list of its local services to other peer devices around it. To do so, it uses the local Forwarding Manager in the device. These local services correspond to services of local agents that are *registered* to the local Directory Facilitator. The Forwarding Manager receives all messages from the network. It decides based on the local policy whether to process the message or not. For example, if the policy of the local node is such that it does not want to form its own alliance, then it would not accept any advertisement. The Forwarding Manager passes the advertisement to the Cache Manager. A Cache Manager in a remote device on receiving the advertisement, decides based on its policy whether to cache it or not. For example, the policy of the cache manager could be that it will not store more than a certain number of advertisements. This policy is driven by the resource-rich ness of the device. The policy could also be user-driven. The user might not want to store any advertisement. In that case, the size of the cache for such a device would be zero. The idea behind passive caching of advertisements rather than active pulling of service descriptions from neighboring nodes has got a few advantages:

- Efficiency in detecting the change in Environment:
  We follow the *push* paradigm for such service advertisements since it enables us to detect in an efficient manner when the ad-hoc environment has changed. When a device receives a new advertisement from a new device, it automatically knows that there is a new platform in the vicinity and could potentially be included in the alliance. The pull-based approach of explicitly searching for new platforms in the vicinity would have imposed more burden on the mobile device hosting a platform. As we shall see later, the frequency of advertisement could be adjusted based on the relative stability of the alliance and hence the burden imposed on the mobile device would be less.

- Advertisement Collision:
  In a standard pull-based paradigm, a mobile agent platform would have broadcasted a request to all its neighboring platforms asking for a list of local services. The peer platforms would immediately

have sent a list of services to the querying platform. Such a solution has a potential danger of the advertisements from different platforms colliding with each other at the receiver's end.

- Flexibility in handling advertisements:
  The receiver receiving the advertisement has the option to either store or reject the advertisement based on its current policy. The policy could be user-driven or resource-driven. Either way, we guarantee more flexibility of handing advertisements in the system in this way

The Cache Manager is responsible to handle remote advertisements, store remote advertisements of services, expunge the cache and also handle requests (from DF) to match services present in the cache. In this way, a platform implicitly makes its services available to other remote devices around it. The Forwarding Manager, on receiving an advertisement might also decide to forward it to members of its own alliance or broadcast the advertisement to all other nodes in its vicinity. This is determined based on whether the advertisement is meant to traverse multiple hops (determined by the sender node of the advertisement). member. The sender node becomes a member of the alliances of the nodes that cache its advertisement. Figure 2 elucidates the concept of alliance formation in an ad-hoc environment.

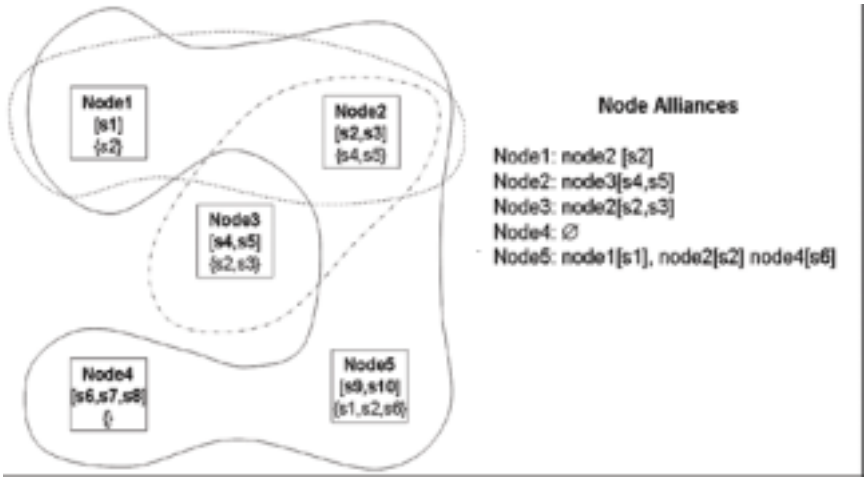

Figure 2: Alliance Formation in Ad-hoc Environment

The cache expunge policy depends on the policy specified by the Policy Manager. Each advertisement contains a lifetime (specified by a time-to-live field in the advertisement structure). When a new advertisement is received by a Cache Manager, the Cache Manager decides to either accept the advertisement or reject the advertisement. The Cache Manager decides to accept an advertisement only when there is sufficient space in the Cache to hold the advertisement or when an old advertisement can be removed out of the cache.

3.2 Policy-based Advertising

We introduce the concept of policy-based advertising of services by the Advertisement Manager. The policy may potentially reflect a combination of the user's preferences, resources present in the device, and application-related preferences. Various parameters of the advertising are controlled based on the policy structure of each and every device. Thus, each platform in this environment may follow a different policy in

advertising. However, the advertisements will follow a standard protocol (e.g. FIPA Tell protocol). In this section, we describe the different aspects of the advertisement that may or may not be influenced by the policy.

3.2.1 Advertisement Frequency Control

A very important aspect of advertisement is to decide the frequency at which the advertisements are going to be sent out to the platforms in the vicinity. We enumerate the different possibilities:

1. Constant Frequency:
   Each Cache Manager in a platform sends out a list of the local services that the registered to the Directory Facilitator at a constant time interval. This time interval need not be uniform across all platforms. Each platform might have its own frequency of advertising. The immediate advantage of such a design would be that a platform having inter-advertisement time interval of t would be detected within $t + \delta$ where δ = network delay in message propagation. However, such a design choice has the disadvantage of network overhead if the frequency of advertisement is too high. Moreover, high frequency of advertisement may not be needed in a relatively stable ad-hoc environment. On the other hand, in a highly dynamic ad-hoc environment, a low advertisement frequency might not be sufficient to form an alliance.

2. Variable Frequency:
   Each Cache Manager in a platform may follow a Multiplicative Increase Linear Decrease (MILD) algorithm or a Binary Exponential Backoff (BEB) Mechanism to vary its advertisement frequency so as to adapt the advertisement rate. There will be a threshold value for the inter-advertisement time. In this design, a node will send out advertisements at a high frequency at the start. It will follow the MILD or BEB algorithm to increase the inter-advertisement time (and hence the frequency) based on whether it is receiving any new advertisement or not. If it receives a new advertisement, it knows that there is a new platform in the vicinity and hence goes back to the starting frequency (in case of BEB) to advertise its local services to the newly arrived platform in the alliance. Thus if the environment around a platform changes rapidly, the advertisement frequency would be more. Contrarily, if the environment is relatively stable, the advertisement frequency would be less (following the BEB or MILD) and hence would impose less network overhead. If in either algorithm, the threshold value of the inter-advertisement time is reached, the algorithm starts from the beginning. This type of advertisement policy would be adaptive to the mobility and changes of the ad-hoc environment.

3. Adaptive Mobility-based Advertisement Frequency:
   In this approach, each Cache Manager sends out an advertisement only when it receives a new advertisement. A new advertisement signifies the existence of a new platform in the alliance. Thus, it sends a list of its own local services immediately in order to be included in the alliance of the new node. This type of advertisement policy is highly adaptive to the mobility and change of the environment. The overhead imposed on the network is also adaptive to the change in the environment. In a highly dynamic ad-hoc alliance formation environment, the network load (due to advertisements) would be relatively more than the network load in a relatively stable environment (since there are no new advertisements). However, problem with this policy would be that a new platform might remain undetected for a long time in a relatively stable environment where the advertisement frequency is low. A combination of variable frequency techniques to alternate advertisement frequencies along with this scheme would be able to avoid that problem.

The advertisement scheme to be followed by a platform or an alliance would be handled by the Policy Manager in each platform. This would provide the required flexibility to handle user preferences in using up

the resources of its own mobile device. Figure 3 shows the control flow between the different components in the sender node and receiver node in sending and receiving advertisements.

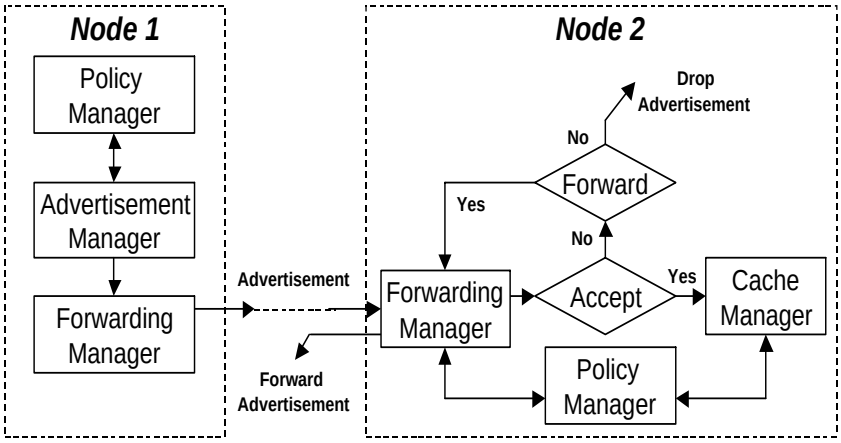

Figure 3: Advertisement Flow between Nodes

3.2.2 Alliance Diameter Control

Each advertisement contains a *hop-count* field that would decide the number of hops to which the advertisement may be propagated by receiving platforms. This hop-count field determines the *diameter* of the Alliance of the receiving platform/node. This is because, any platform within the diameter would have the potential to cache the local services of the Directory Facilitator of the source platform. In a highly dynamic environment, a small diameter of an alliance is more desirable. This is because, if the diameter is large, stability of a platform in an alliance would be less. Hence, the information of services in the cache would become stale quickly. For implementation purposes, we will be limiting the hop-count to one. Hence, a platform will only advertise its local services to its one-hop neighbors and an alliance diameter would be one.

A platform on receiving an advertisement will perform the following activities:

1. Accept the advertisement and store it in the cache if the local Policy Manager allows that
2. Decrement the hop-count field of the advertisement
3. If after decrementing the hop-count field, the value of the field is zero, then discard the advertisement else
   Forward the advertisement to its one-hop neighbors if the Policy Manager allows that.

Due to resource consumptions and other user preferences, the policy in a platform might decide not to forward any advertisement message to its one hop neighbors. In this case, the advertisement would be silently discarded by the platform. When a platform decides to forward an advertisement, it sends the advertisement to the Forwarding Manager. The Forwarding Manager manages the re-broadcasting of the advertisement its one-hop neighbors.

3.2.3 Advertisement Structure

There are two types of advertisement structures in our design, viz. the Service Advertisement and the Platform Advertisement structures. The Service Advertisement structure is used to send out a new advertisement describing all the services that are registered to the local Directory Facilitator of a platform. This type of service advertisement is sent out only when a node receives a new advertisement or somehow detects that there is a new platform in the vicinity that may potentially be included in the alliance. This is determined by the Policy Manager of the particular platform. The advertisement primarily contains the following information:

- Service Description(s)
- Agent Identifiers
- Platform Identifier
- Network Address: Network Address refers to the actual hardware interface address through which the device can be accessed in an ad-hoc network. For example, in a Bluetooth Network, this would refer to the 48-bit Bluetooth Address.
- Accessory Information: Accessory information include a time-to-live field that is used to convey information about how long the platform is going to be in the vicinity, resources of the local platform (e.g. cache size) or some important policy information (e.g. I don't forward any advertisements or requests) that might help the receiver node to decide whether to forward any requests/advertisements to this node later on.

The platform Advertisement structure is used as a *heartbeat* message to notify the members of an alliance about the presence of a certain platform in its vicinity. Clearly, it is not required for a platform to advertise all its services (and hence send a large volume of data) every subsequent time interval in the event that the platform is relatively stable in the environment. It only needs to send the whole information (using Service Advertisement) when a new platform (device) is in the vicinity. For the rest of the time, it suffices to send only the information about the presence of the platform in the vicinity of the other platforms. The Cache Managers in the other platforms can update the cache information accordingly. The Platform Advertisement structure contains only the *platform identifier* and the *network address* of the sender platform. Thus is imposes a lot less overhead on the network in terms of message size.

3.3 Request Routing

The main goal of our system is to make a best-effort service discovery amongst agent platforms in an ad-hoc environment. Each platform or node in the system has a local Directory Facilitator and an Agent Management System [11]. When there is a request from any agent to communicate with or discover an agent-service, the request first comes to the local Directory Facilitator (DF). If the DF does not have the service, the DF forwards the request to the Cache Manager. If the service description is found in the local cache, the information about the location of the agent (platform identifier, network address) is taken as hint and provided to the requesting agent. The requesting agent contacts the agent-service using standard Agent Communication Language [10]. The Cache Manager forwards the request to the Forwarding Manager in case of two events:

1. If the hint provided by the local Cache Manager turns out to be false. This happens when the agent fails to communicate with the device that has the service
2. A match is not found in the local Cache Manager

The Forwarding Manager forwards the request to selected agent platforms in its vicinity. Some of those platforms may be a member of its own alliance. It also sets a hop-count field for the request to limit the number of hops or number of adjacent alliances through which the request is going to traverse. The value of the hop-count field is decided by the local Policy Manager. The standard Policy that is used by the Policy Manager is to increment the hop-count from 1 to MAX-HOP-COUNT (maximum number of hops the request might be allowed to traverse) before it discovers the required service. There are two different ways of sending the request to neighboring platforms in the vicinity:

1. Broadcasting:
   The request could be broadcasted to all the neighboring devices. In this case, the request will reach all the nodes in the vicinity. Some of these nodes might not be resource-rich enough to even forward the request any further or some of them might not be members of any other alliance (since there are no other nodes in that node's vicinity).

2. Multicasting:
   The request could be multicasted to a set of particular neighbors. The Policy Manager would specify the method (broadcasting or multicasting). The multicasting could be used to prevent broadcast storms. The device could multicast the request to its most active neighbors. It could identify its most active neighbors by looking into its cache and evaluating advertisement patterns. The "Accessory Information" sent with each advertisement to the Cache Manager is used to relatively evaluate the nodes/platforms in the vicinity where there will be maximum chances of discovering a particular service. For example, the "Accessory Information" could contain the cache size of the sender node. This would provide a measure of the resource-rich ness of that node. It could also contain an idea of how many platforms/nodes are there in the vicinity of the sender node. In the event, that the sender node is a member of another large alliance of powerful nodes, it might be advantageous to send a "Request for Service" to that node since it will in turn broadcast/multicast it to all other nodes in its vicinity. The Forwarding Manager computes a list of nearby platforms where it could multicast the "Request for Service" using such information from the Cache Manager. It then forwards the request after decrementing the hop-count field by one. The strength of our design lies in the fact that all these decisions would be strategically influenced by the local Policy Manager. Since the Policy Manager takes user preferences and local resources into consideration, the solution could be adapted based on user's needs or concerns.

Each device upon receiving a request message can chose to drop the message or process it. The local policy is used to make this decision. If the device chooses to process the request message, it checks its DF to see if it is hosting the desired service. If the service is local then the service description is sent to the requesting node. If the device is not hosting the needed service, it could check the local cache. If the cache contains info about the service then the device replies with hint. Otherwise, the request if sent to the Forwarding Manager. The Forwarding Manager decides to forward the request to other nodes in its vicinity depending on the local policy. Figure 4 shows the control flow between the different components in processing the request. In the event that a service was discovered, the reply is reverse-routed back to the source node. Thus the reply follows the same path that the request had followed, but in the opposite direction. The intermediate nodes may or may not choose to record this as a successful discovery to maintain hit-rate statistics or utility measure of itself. This utility measure could be sent to remote platforms with advertisements to give the other nodes an idea of its efficiency.

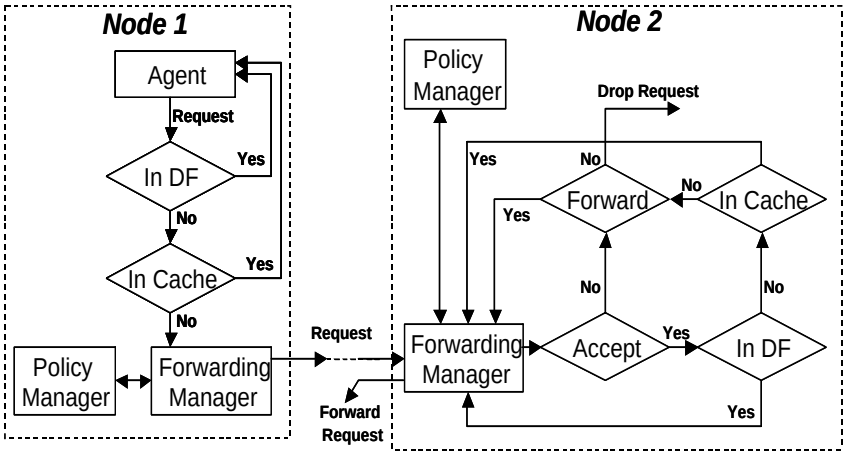

Figure 4: Control Flow for a Request

4 Implementation

To substantiate our design we have come up with a preliminary implementation of our scheme. The platform used to demonstrate our scheme is a JADE [1] based Lightweight and Extensible Agent Platform called LEAP [2]. LEAP is realized on mobile devices by implementing a new kernel for JADE. It is a FIPA compliant open source system that revolves around the idea of making the platform Lightweight and Extensible, as opposed to JADE's focus on environmental functions like ontologies, monitoring facilities etc.

We use a set of Laptops and Windows CE iPaqs as a test set of devices on which our architecture is implemented. Every device (laptops and iPaqs) has a Main Container that has a *Directory Facilitator* (DF) and an *Agent Management Service* (AMS) running. In our system, the functionality of the Main Container has been extended to incorporate a peer-to-peer caching mechanism for the discovery of non-local services in an ad hoc network. The new components in the system include a Cache Manager component, an Advertisement Manager component and a Service Discovery component.

4.1 Advertisement Manager

This component takes care of periodically advertising its local services to all of its one-hop neighbors in the Bluetooth network. Current Implementation considers the Alliance Diameter as 1. Each Advertisement packet has a unique Advertisement Identifier, a Directory Facilitator Agent Description, the device Cache Size and the Advertisement's Time to Live. We use Multiplicative Increase and Linear Decrease (MILD) Algorithm to determine the time interval between successive advertisements. The rate of incoming advertisements will determine the value of this time interval. To send periodic advertisements each mobile device does an HCI inquiry [9] and opens a RFCOMM connection [8] with all the devices found during inquiry. It then closes the connection after sending the advertisement messages.
All incoming Advertisements are cached based on the caching policy of the Platform. In our case, the resourcefulness (measured as a function of memory, processor speed etc.) of the device is taken into account to determine the caching policy.

4.2 Cache Manager

The Cache Manager maintains a Cache entry for services that are present one hop away. The number of entrees in the Cache Manager determines the Alliance formed by a node. When an Agent searches for a service, it first looks up the DF table. If no such service is available in the DF, then the Cache is looked up to check for services available in the node's own alliance. If there is no such entry in the cache, then a Bluetooth Service Discovery is initiated to look for services in other nodes that might not have been cached. When a service is discovered, the DF Agent Description of the service is sent to the agent that is performing the lookup. Further communication can be established using the Agent Identifier of the service provider.

4.3 Service Discovery Component

We have used the Bluetooth Service Discovery Protocol to discover services present in remote DFs. Every DF in the device registers its local services to the local Bluetooth Service Discovery Manager. When a matching service is not found in the local DF or Cache, the Bluetooth Service Discovery Manager first discovers the devices that are in the vicinity of the source and then initiates a request to discover the service in the remote Bluetooth Service Discovery Manager. The remote device may or may not be a member of the Alliance of the requester node. We are using the Axis OpenBT stack for Bluetooth communication. The laptops use Ericsson Bluetooth Development Kit as hardware components and the iPaqs use the embedded Bluetooth in them.

We have developed a Bluetooth Service Discovery Manager that takes request from the DF service on the Leap platform and performs a bluetooth service discovery. The request is passed in the form on an agent description which consists of the agent-id, service-type and the attributes of the services. Since Bluetooth discovery is done through UUIDs we perform a mapping of these agent service types into UUIDs. We assume that the UUID mapping is same on all the Bluetooth devices running agents. Currently, the policies in each device is primarily based on the resource availability of that device. We use the cache size as a parameter to judge the resource availability of a device.

5 Discussion

Our design presents a policy-based alliance formation between agents in ad-hoc networks. Peer-to-Peer caching of services with controlled advertisements is used for realizing such an architecture. This gives rise to completely decentralized alliances, that are aware of resource-limitations of devices and are adaptive to the mobility patterns in ad-hoc networks.

The policy-driven approach gives users the flexibility to configure the agent platform based on their needs. Moreover, it makes the agent platform adaptable to the differing resource availability on different devices. FIPA mandates that lifecycle of agents should be managed by the agent platform. Our design takes this notion a step further in as much as an agent platform is now governed by the dynamics of the environment it is hosted in and specific preferences of the user.

To illustrate the flexibility of our design, let us consider the following scenario.

An ad-hoc network of nodes in which one node is resource-rich(laptop type) and the others are resource-poor(cell phone type). The policy manager on the laptop might have the policy of storing unlimited service advertisements in the cache. The cell phones might have the policy of not storing any service advertisements. In such a case, all the cell phones would effectively use the cache information of the laptop to discover agents and services. This kind of a network model represents the "structured compound formation" referred to in

the FIPA Call For Technology [5]. An obvious problem with this approach is that if the laptop moves out of the network, all the cell phones would effectively resort to peer-to-peer discoveries. In the case of extremely dynamic environments, this might lead to thrashing. In our design, this notion of a structured compound formation is implicit. Explicit service registrations are replaced by implicit caching of service information. This avoids the overhead involved in election of the central node hosting the DF and AMS services. The laptop has complete information about the alliance formed with it's neighbors. The cell phones, though being a part of the alliance are not aware of it, but based on the policy they can infer that laptop is the only device which is caching service information (enabled by way of accessory information3.2.3 passed with service advertisements).

Moreover, this notion of peer-to-peer caching, makes the network more resilient to node movements and reduces the overhead of service discovery in such dynamic environments. Each device in the network, tries to maintain a cache of services based on the policy specification. In such a case, even if the resource-rich node moves out of the network, the other devices need not undergo a complete reconfiguration to peer-to-peer mode. They utilize the cache information available with other nodes for efficient service discovery. For example, in the previous model, if we had a few PDA type devices where caching was enabled, then the network nodes would be able to utilize this cache information, in case the laptop moved out.

When these nodes are stable in the network, the variable frequency advertising would decrease the network traffic and the caches would represent consistent information. In the case of these nodes being dynamic, caches would become stale soon. The commitment period of service descriptions, graceful exit from the network and/or request misses would help maintain consistent caches.

6 Conformance to FIPA

Our design conforms to the FIPA specification completely. Since we have mandated that an agent platform is always present on the device, directory services, white-page services and lifecycle management are all done by the platform for local agents.

References

[1] F. Belligemine and G. Rimassa. Jade-a fipa-compliant agent framework. In *Proc. PAAM '99. London*, pages 97–108, 1999.

[2] F. Bergenti and A. Poggi. Leap: A fipa platform for handheld and mobile devices. In *presented at ATAL*, 2001.

[3] Connected Limited Device Configuration. `http://java.sun.com/products/cldc/`.

[4] PersonalJava Application Environment. `http://java.sun.com/products/personaljava/`.

[5] FIPA Call for Technology for Compound Formation in Ad hoc Environments. `http://www.fipa.org/docs/wps/f-wp-00020/f-wp-00020.html`.

[6] Micro Edition Java 2 Platform. `http://java.sun.com/j2me/`.

[7] Mobile Information Device Profile (MIDP). `http://java.sun.com/products/midp/`.

[8] Bluetooth White Paper. World Wide Web, http://www.bluetooth.com/developer/whitepaper.

[9] Bluetooth Specification. World Wide Web, http://www.bluetooth.com/developer/specification/Bluetooth_11_Specifica%tioBook.pdf.

[10] FIPA ACL Message Structure Specification. World Wide Web, http://www.fipa.org/specs/fipa00061/.

[11] FIPA Agent Management Specification. http://www.fipa.org/specs/fipa00023.