Chapter 7

SERVICE BINDING AND MOBILITY MANAGEMENT IN CASP

Introduction

Service selection is the process of selecting one service provider from among the set of all known service providers that are providing the desired service. It involves the analysis of the requester’s need and the properties of each provider and selecting the one that it believes well meets the requester’s needs. Upon selecting a particular provider, the requester binds to it by noting the service provider’s address. This address is used to communicate with the service provider when the requester invokes the service. In previous chapters the service discovery process and selection of service provider were discussed. This chapter focuses on the binding and mobility management issues.

Service invocation is the process of a requester actually using the capabilities of a service provider. Like discovery, invocations must also be tolerant to communication delays that are naturally induced by mobile ad hoc networks. The amount of delay that a requester can tolerate depends on the interactivity level needed for reasonably good operating conditions. From requester’s point of view after issuing its request, it may expect one or more responses. According to these requester’s expectations, service provider should know what and how to deliver their services. The presented solutions use remote invocation, copying and rebinding to other service provider depending upon the execution condition and requester’s needs.
Services should not be oriented towards specific environments, but should be flexible so that they could be consumed by different users through diverse communication systems and with various devices. As stated by Banavar (2000), this issue can be addressed by decoupling services, applications, devices and users from each other and viewing them as completely independent entities. They are no longer firmly tied together, but have their own functions and objectives and interact with one another when needed. In particular applications are seen as special entities that perform specialized tasks on users’ behalf. Appropriate infrastructure support allows them to be highly customizable and personalized according to user’s needs, adapt to changing environmental conditions and be independent of underlying communication technology. Not only would it increase service reusability and improve users’ perception of it, but also it would bring current communication standards closer to each other and alleviate the problem of interoperability.

In chapter 5 the discussion on how decoupling provides an elegant way to manage service discovery is provided. This chapter further explores how decoupling can enhance applications in terms of their ability to adapt to network dynamics.

Section 7.1 presents the default remote invocation solutions proposed in this framework, detailing how client can invoke a service and how it receives response from the service provider. Section 7.3 provides the adaptation mechanisms used in this framework in order to handle mobility during service provisioning. Section 7.4 elaborates implementation of framework and 7.5 gives the experimental evaluation of the binding polices proposed in this framework.

7.1 Overview of Service Accessing

The system consists of requesters and service provider nodes. The requester and service providers communicate locally when they share the node, or over a wireless link when they are located on different nodes. Whenever the service requester finds a matching and best service provider the requester may either ‘copy’ the service to the requester’s site or it can invoke the services using the concept of remote invocation. The performance of the service will not be
affected even in the time of mobility in the case of *copying* strategy. From the point when the service has been copied in to the requester’s site onwards it can consider the service as that of its other local services.

When it is decided to do remote invocation, the requester binds to the service provider; it can invoke the provider’s service by sending an invocation message that initiates the invocation. Upon receiving this message the provider executes the service and replies with a “result message” that contains the invocation result. This service invocation pattern may also be influenced by the context.

After sometime due mobility or any other reason, if the requester can’t access the service provider, it can switch to other providers. This is termed as rebinding. Switching providers is assumed to involve no state transfer from the old provider to the new one. This simplifies the framework by eliminating any intrinsic overhead associated with switching providers. That is assuming the set of matching providers is known, the framework can arbitrarily switch between any providers within the set. This will be discussed in section 7.3.

Furthermore this assumption enables the framework to react to network disconnection, which is important for ensuring energy efficiency. Many services like sensing and data routing can be offered in this manner that meets this assumption, though some services like data storage cannot. In the future, this assumption can be removed by implementing a mechanism that determines when a provider is about to disconnect so that the data can be transferred before actual disconnection occurs.

The design goals of service binding and maintenance module are (i) Making a binding between the requester and the service provider, providing mechanisms to invoke the service & get results and monitoring the context changes. The change has to be intimated to the module in charge for rebinding. (ii) Enhance service availability through application-transparent service rebinding. This is necessary due to the transient connectivity between the consumer and the providers. Achieving this requires an algorithm that determines *when* to switch providers. The method of when to switch the service providers is given by the application developers in terms of policies. Processing methods of these polices will discussed in section 7.3.
7.2 Overview of Binding between Requester and Service Provider

A binding represents the exchange of reference between a requester and a provider in order for invocation interactions to work. Traditional client/server computing models have a strong binding between the two entities: a client is bound to a server for all future interactions. In the case of mobile ad hoc networks the availability of the provider may not be sure for the entire duration of the client. In order to handle this situation, service-oriented computing provides late binding.

There are two possible type of binding between the service provider and the requester. Following are the types of binding strategies considered in this framework. Figure 7.1 shows the various binding strategies used in CASP.

*Figure 7.1 Different Binding Strategies used in CASP*

*Copy Bind:* Once the service provider is selected; the binder module will be informed about this so that it can check the type of binding that can be done to this service. After the
checking whether copy can be done, Copy binding will be made by calling the function 
\texttt{copybind()}. If copy bind is not possible, remote invocation can be done. Once the service been 
copied onto the device this service will be considered as a local service and accessed as local 

\textit{Remote Invocation:} rather than moving the services, this strategy modifies the node’s 

binding to refer to the services remotely. This remote invocation is possible because of the 

concept of late binding. A brief note on late and loose binding is given below.

\textit{Late Binding} is a binding characterises, in which the client does not need to be bound to a 

service provider before actually invoking it. This type of binding is possible by the use of 

Service Oriented Computing. The requester starts searching for a service and though it finds 

more than one provider binding is delayed until it makes an explicit invocation request to a 

specific provider. This type of binding is called late binding. As seen in figure 7.2 from the point 
in time $t_0$ when a client discovered a service provider $P_1$ to the point $t_1$ when the client started an 

invocation, the client was not bound to any provider.

Since the service providers may come and leave as they wish the same service provider 

may not be available again for the invocation. This unavailability of service provider mandates 

the requester to find new service provider. If there is a list of available service providers with the 

requester then it can select one among them - this is called reselection among the list. If there is 

no provider in the local directory, request will be formed to be sent in the network for a match, 

which is called rediscovery.

If any other provider is found from either the reselection or rediscovery the client will be 

bound to this new provider for the same service. This type may be termed as loose binding.

\textit{Loose binding} is an effort to push this late binding even more lately. Loose binding is 

proposed, where it allows the client to invoke a service instance without binding to the same 

service provider every time it invokes this service. From the Figure 7.2, it can be noticed that 

from the point in time $t_0$ when a requester discovered a service provider $P_1$ to the point $t_2$ when 

the Requester $R$ received an invocation response, the client was not bound to any provider. Also 

until the request is sent it is neither bound to $P_1$ nor to $P_2$. After $t_2$, the next invocation is sent to 

provider $P_2$. 
Remote invocation can be done at three levels. Firstly, by providing service location and then at the second level by means of a communication mechanism, the application establishes a session with the service. This can be carried out by using remote procedure calls (RPCs), e.g. using remote method invocation in java and downloadable java code, or using Simple Object Access Protocol (SOAP). At the third level some of the protocols support application operations (Zhu (2005)). This thesis uses SOAP invocation.

**Invocation request to service providers**

A requester consumes a service by invoking its functional capabilities. A service may be offered by multiple redundant providers. In order for the provider to receive the request, the
header is having the provider’s id. In addition to the provider’s id it sends the service type and the functional operation, input for that function. It is assumed that the service of a particular service type will have certain functions and their functional interfaces. The default remote invocation behaviour of the proposed framework consists of the following:

- The user agent creates SOAP request according to the description of the selected providers’ functional specification in the descriptor.
- The requester sends its request to the discovered service provider.
- The Service provider can receive the requester’s request. A provider creates a response message and sends it back to the client.

```
message_type = "service_invocation"
service_type = "service type"
provider_id = "16 bit number"
issuedat = timestamp
deadline = time

<SOAP-ENV: BODY>
<convert>
<input1 xsi:type="xsi:int"> value</input1>
</convert>
</SOAP-ENV: BODY>
```

**Figure 7.3 CASP Service Invocation Request Message Structure**

The invocation completes when the user receives the result response for the request. Figure 7.3 shows the invocation message structure.

- The *message type* specifies that it is a request message.
- The *issuedat* specifies the request creation time and date
- *service_type* specifies the type of the service to which this invocation message is sent.
- *provider_id* specifies the provider id for which this message is destined to.
- *deadline* constraints the response time; it sets the validity of the request and its corresponding response. This time constraint will guarantee that only valid requests and responses are present in the network and the obsolete response and request will be deleted and will not be relayed anymore. The client agent can define the deadline based on (i) an
application related delay constraint, before which the client application needs to receive a response, (ii) node’s mobility related delay constraint (i.e.) if the mobile node knows that it will be leaving its current location at certain time, (iii) a communication related delay, that is a constraint of the network environment. Deadline should be greater than the time required for the network to provide a round trip of a message. Using deadline attribute, the client agents specifies its willingness time limit. If the response is not received in that interval then it is assumed that there is a failure.

The payload contains the well formatted SOAP invocation. The header contains the discovered provider node’s identity and the service type. The SOAP envelope will have the input to the function.

*Provider’s response to the requester invocation*

Once the service provider invokes the function it forms a SOAP message and sends it to the requester. The response message structure is shown in figure 7.4. The header attributes of the response of service provider are:

- *provider_id* specifies the provider which sends the response
- *requester_id*: specifies the request id for which this response has been created.
- *message_type* specifies that the type of the message is ‘response’
- *issuedat* the time at which the response is issued.

```
message_type = "service_response"
provider_id="16 bit number"
requester_id ="16 bit number"
issuedat=timestamp

<SOAP-ENV:Body> <covert>
  <result xsi:type='xsd:int'> 4203
</result></convert>
</SOAP-ENV:Body>
```
Response management policy

The framework uses the single provider invocation as default. Group invocation is also provided which may be used based on the type of the application.

In Single provider invocation, the service provider ranked top is selected and sent with the request. If there is a failure then, the client issued the second request to another provider. Therefore the client will have a response only after the delay time of deadline time added with response time for best provider. Though the time needed to get the response may longer, the communication overhead is reduced compared to the group communication, where there are many message flows from and to the requester.

With group invocation, the requester can get response from all possible providers. The delay time to get the response is the least time among the possible providers. But the controversy is that there may be many message flows. There may be multiple responses for a single invocation request.

7.3 Overview of Adaptation

In mobile ad hoc networks, applications represent services customized to a form suitable for a concrete network, terminal and user. Since those factors tend to dynamically change, these applications need to be adaptable to possible changes in the environment. As the service provider and the consumer may be located on different devices, the service invocation process may fail e.g. due to message loss. To account for the above said issues a mechanism that notifies a task when and why an invocation fails is provided. This is necessary because service invocations may fail in many ways depending upon whether the service is local or remote. And application may want to handle error conditions differently. For example, local invocations may fail because the service is busy in which case the application may try later, while remote invocations may fail due to disconnection, in which case the task may want to abort and switch to a different provider.
Application’s appropriate reaction to any occurred changes should preserve quality and usability of the provided service. Adaptability itself means only that the application can change its behaviour and knows how to perform, e.g., it may apply different strategy for achieving its goals, or can reconfigure its own structure appropriately. The important issue here is proactivity (Satyanarayanan (2001)). It signifies the capability of an application to anticipate future circumstances and adjust its own behaviour appropriately. Proactivity is the way to intelligent self-tuning (Flinn et al (2001)) that further decreases the level of user distraction. Capturing user intent is a critical factor in achieving these outcomes since applications seek to foresee user actions. The user intent has been accessed in the form of application profile. Please refer to chapter 6 for application profile.

Thus, the objective of adaptation mechanism is to hide service provider’s disconnection from the application. Thus, the adaptation mechanism should prevent the application from being exposed to service invocation failure when other potential providers exist within the reachable limit of the consumer. Specifically, a successful adaptation mechanism must ensure that the results of an on-demand invocation are always returned successfully. Following are the three kinds of adaptations specified in the literature:

Framework adaptation: User preferences are used during service selection, firstly during service filtering and then in service ranking. Mandatory preferences are used during service filtering to remove unwanted services from the discovered service list. Secondary preferences are used during service ranking to rank the remaining services in the filtered list. The ranked service list is used to select services based on providers metric.

Service adaptation Services can have personalised parameters. Services may need to re-personalize due to context changes. When these occur, personalization will notify any updated preference outcomes to the appropriate services allowing them to re-personalise at runtime, dynamically changing to meet the user’s needs.

Protocol adaptation: Depending on the location and preferences of the user and the Quality of Service requirements of the services, different Network routes have to be selected.
The mechanism of framework adaptation is used in this framework. In order to handle the situations mentioned above. The proposed framework provides mechanisms to incorporate one of these actions (i) Adaptation based on Change in the context (ii) adaptation based on network dynamics.

(i) Adaptation based on Change in the context.

This case includes accessing and monitoring the context and identifying the status of the service by inquiring the status information from the service provider.

To take decision on whether to switch over to other provider the internal status or the other context information has to be accessed. Monitors are the components needed to facilitate binding management. The function of the ‘Monitors’ is to detect these critical changes in the system environment and notify the adaptation manager of such changes. The adaptation manager components are responsible for evaluating the policies and enforce them when critical change occurs.

The Monitor module is the collection of three types of Monitors namely i) Internal Status Monitors (ii) Network Status monitors (iii) Bandwidth Monitors. Internal Status monitors are responsible for collecting the internal resources data such as memory, battery. Detailed discussion on this has been presented in thesis author’s publication Ponmozhi (2012). Since many of the services depend on the internal status of the involved devices. Network status monitors look for signal quality; bit rate. The bandwidth requirement of a service has to be satisfied in some of the applications. So the framework provides a set of adaptors for accessing the values of them and monitors for the expected value of those parameters. When the application profile is parsed, the monitors are created. For example if the application profile specifies to look for other service provider when bandwidth become less than ‘t’ then, while parsing the monitor bandwidthmonitor(t) will be instantiated and registered as event listener so that when the bandwidth falls below or equal to ‘t’ then ‘bandwidthlow’ event will be raised.

The framework also provides aggregate monitors which will combine more than one monitor. aggregate_or_monitor and aggregate_and_monitor to raise an event when more than one monitor value has be compared. The raised events will be caught by the policy enforcer module and processed by it which will be discussed in section 7.4.
The second case is used when adaptation needed based on service status. There are different approaches to know the *Service Status* from the service provider. The service status can be obtained by periodically polling, in which a client regularly asks the service status. Another approach is notification, in which the client subscribes to a service status event and any changes on the subscribed event are notified to the subscribers of that particular event. Periodical polling is slightly modified and used in this thesis. The *casp_service_status* message is sent to the service provider for accessing the current status of the service. Status Inquiry is possible only if the service provider has the mechanism to identify and respond to the status request. The user agent after receiving the status will decide for termination of the service or rebinding event which will be handled by the policy engine. Please refer to chapter 5 for different messages provided by CASP.

The method of detecting invocation failure is that if the provider does not respond within a certain amount of time after an invocation message is sent it is considered as failure. This can be detected when the current provider is removed from the neighbour list, which is maintained by lower-level services like routing and MAC layers or from link estimator (Rodrigo Fonseca (2007)).

Breaking of service binding may be explicitly initiated in one of the following cases also (i) by the node providing the service (service provider’s mobility) (ii) by the node using the service (requester node’s mobility).

In the case of its movement the service provider sends *deallocation* message to all its clients, Deletes all the requester nodes’ address from the service_clients table. Now the requester nodes may explicitly initiate the service searching and rebind to the newly selected service provider if needed.

When the service requester node moves it could not get hello packet for a threshold period of time it sends a wait for (Wait-time). It alters the service-provider’s table. Now also the requester may initiate rebinding operation explicitly by the use of *rebind_explicit* event.

Adaptive service reselection strategies are developed to automatically adjust bindings between service providers and consumers in response to changes in the execution context. This is important because mobile ad hoc networks exhibit high levels of dynamics due to node mobility,
exposure to a harsh and dynamic environment and use of low powered radio susceptible to fluctuations in link quality. A key advantage of an adaptive service binding scheme is that it enables application-transparent handling of network topology changes in the framework and thus greatly simplifies application development despite network dynamics. The third type of binding is termed as rebinding which will be invoked when the already selected service provider cannot be communicated further because of the network dynamics and when new service provider has to be selected. For these cases we have provided with two policies namely context_change, explicit_rebind. The events raised for these are also named as context_cng, exp_rebind. The monitors may check for the value and raise the context_cng event. Similarly the exp_rebind may be raised by the application when the service status in not the expected one. The policies for these two are written as a common policy for the application not for the service_type. As they are not coming under mobility issues these two are not shown in the mobility handling policies diagram.

(ii) Adaptation based on network dynamics.

In general, mobile ad hoc networks will have dynamic connectivity only. This dynamicity increases the network load to maintain up-to-date service provider details. This extra network load may challenge the performance of discovery algorithms. In general, three causes of network dynamics can be identified (i) nodes may be mobile (ii) battery-supplied devices might exhaust their energy (iii) the quality of the wireless transmissions might be varying due to unstable propagation conditions.

When a node is moving across the network, its set of neighbours is changing continuously. A temporary lack of energy supply can cause nodes to disappear and worsening propagation conditions may result in broken links between nodes or lost packets. Similarly, due to changing conditions nodes and/or links can reappear.

If a link between two nodes disappears, those two nodes no longer consider each other as neighbours. If lost packets are keep-alive messages that inform about the presence of the node, other nodes in the network may consider this node to have been disappeared. If lost packets are updates, nodes continue routing packets with outdated information and generate redundant
traffic. When links or nodes reappear in the network, corresponding nodes need to add new neighbours.

Before going to a deeper discussion on binding issues let us see the definition of new services. The notion of new services is different whether it is viewed from the application layer or the framework. For the framework a new service is a service offering new functionalities whether by providing new functional capabilities. A service offering functions that are already available in the environment is considered new by the framework if the functional properties and the non-functional properties associated to the services are new in the context whereas, for users and applications, this service is not new as it publishes the same functionalities but with different non-functional properties.

One important aspect of the adaptation mechanism is the fact that it is reactive. That is, it does not actively seek to change providers so long as the current provider remains available. The reason behind this is the fact that energy efficiency is of paramount importance and needlessly searching for new providers when the current one is still available wastes energy. If there is a guaranteed provider has been found by matching the advertisements sent by the provider then the framework can switch over to it. Since there is no guarantee that a more efficient provider exists, so proactively searching for another provider when the current one is available is risky in terms of wasting energy. Finally, some applications may infrequently invoke services. In this case proactive adaptation is wasteful if the application does not invoke service between multiple adaptations. For these reasons, a passive mechanism that reacts to the application invocations and provider disconnection is preferred.

The mechanism for adapting to network topology is responsible for automatically switching provider to enhance service availability. It is essential due to the transient connectivity between nodes in mobile ad hoc networks.

In both the cases i.e. whether the search is initiated by the service provider nodes mobility or requester node’s mobility or any other cases which needs to switch service providers, the framework has to look for new service provider to bind with. The framework does two kinds of search based on the use’s choice specified in the application profile namely reselection and rediscovery.
These policies are termed as adaptive service binding (rebinding) strategies. They are developed to automatically adjust bindings between service provider and requester in response to changes in the context or any other events.

The framework performed the entire adaptation mechanism in a manner hidden from the application, except for a few policies required to carry out adaptation.

Reselection and rediscovery policies

When a service is offered by multiple nodes in a mobile ad hoc network, the specific matching between requester nodes and service providers typically refers to as service selection. Service selection has a crucial effect on the performance of the mobile ad hoc networks. Over time, changes in network topology degrade the optimality of service selection requiring clients to continuously re-evaluate their choices of a service provider. This process is referred to as reselection. As new nodes may come and leave the network the availability of the service providers may change, the active probe for the availability of service providers is termed as rediscovery. The choice of these two actions can be represented in terms of reselection and rediscovery policies. Figure 7.5 specifies the mobility handling policies used in CASP.

To optimize the performance, Mobile ad hoc network clients need to constantly re-evaluate their choice of a service provider. Re-evaluation has two components: reselection and rediscovery. Reselection reconsiders server selection based only on the current entries in the network pool of resources. Rediscovery involves probing the network for up-to-date information about available service providers. In designing a re-evaluation policy, application developers need to determine when to do reselection and rediscovery. These will be specified in the form of polices. These policies also are parsed and stored as policy object.
The simplest reselection policy is not to do reselection at all. This policy however will likely result in poor performance. An alternative is to do reselection in reaction to change in the service table, such as a new advertisement received, or learning of a change to service specific metrics changed or do reselection only when there is no valid route to the current server. Applications may choose to trigger reselection either proactively or reactively. Three policies used in CASP are: (i) A fast reactive policy is the one which triggers reselection as soon as the active route the current server breaks (ii) the medium policy triggers a reselection whenever it receive explicit request for rebinding (iii) the slow policy as soon as a new service is entered into the service cache.

Two rediscovery policies were developed namely Swift: switches service providers as soon as it becomes aware of that there is no service provider information in the service table. Conservative switches service providers only when there is no longer a valid route to the current server.

To find whether the current route breaks, the routing table information for that provider has to be accessed. A cross layer adapter for routing layer has been created to access the route information for the server this information is used to trigger the event to the policy enforcer.

<table>
<thead>
<tr>
<th>Policies</th>
<th>Events that triggers the policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>re-selection</td>
<td>current route breaks (fast), explicit server request for</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>re-discovery</td>
<td>No service provider information in the service table (conservative) no valid routes to the service provider (Swift).</td>
</tr>
<tr>
<td>explicit_rebind</td>
<td>Invoked whenever the service status query results is not the expected one. Or when the invocation failure occurs.</td>
</tr>
<tr>
<td>context_change</td>
<td>Whenever there is a change in the specified context.</td>
</tr>
</tbody>
</table>

Table 7.1 Polices and Events Used in CASP

*Context change policy:* The context change policy is needed to handle the following situations. Service adaptation before deployment of a service ensures that the service is tailored to the capabilities of the device (Wagelaar, D (2004)). Service adaptation can also be activated during the execution of a service; each service has to specify the constraints which define working conditions that guarantee proper execution of the provided service. A context change causing certain constraints to be violated will then trigger the runtime adaptation. To not overcomplicate the self-adaptation of the system, the adaptation triggered only by changes in resources and service requirements. This information is usually readily available and requires no intensive processing.

To do rediscovery or reselection the service must be capable of a handover. As an example, a print service is not capable for handover, because the user does not expect to collect his/her printed papers belonging to a printer job from different printers in different places when she/he moves but, a video service is portable, since the user can watch a movie in different places on different screens.
For realisation of such a scenario, an attribute must be defined to specify whether the service is portable or not. If this attribute is set, this means that the service can be handed over to another service provider.

The current implementation assists applications in selecting the best available service provider, it is not aware of the actual client-server communication state and does not provide additional support for migrating client session between service providers. The stateless service migration may be simple such as opening a new connection to the server but, stateful services requires guaranteeing that application-specific state is consistent across the servers. Application state and session migration is outside the scope of this thesis.

Many traditional adaptable systems are composed of a single adaptation manager that is responsible for the entire adaptation process; which consists of the following phases (i) monitoring, (ii) adaptation selection intelligence and (iii) performing actual adaptation. Since the intelligence to select appropriate adaptations and the mechanism to perform these adaptations are embedded directly within the adaptation manager, this type of systems becomes inflexible and inappropriate for general use. By decoupling the adaptation mechanism from the adaptation manager and removing the intelligence mechanism that select or triggers adaptations, the adaptation manager becomes more scalable and flexible.

The discussions on the situations when re evaluation of binding policies may be needed were given in the previous paragraphs. The realisation of these polices in CASP will be discussed below. Policy specifications maintain a very clean separation of concerns between adaptation available, the adaptation mechanism itself and the decision process determines when these adaptations are performed. Adaptation policy rule is usually made up of an event specification that triggers the rule, which is often fired as a result of monitoring operation; an action to perform in response to the trigger; and a target object that is part of the managed system upon which the action is performed. This event-condition-action (ECA) format is the standard for rule based adaptation system. Polices specified in this thesis are following this ECA system.

Mobility handling policy specifications that the framework uses when binding to these services are embedded in application profile. Mobility handling policies for adaptation guide
binding-related management operations. They include binding policies that define which binding strategy to apply and when. XML is used for policy specification.

<table>
<thead>
<tr>
<th>Policy rebind-reselection-slow</th>
<th>Policy rebind-rediscovey-swift</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;event&gt; new_service_available &lt;/event&gt;</code></td>
<td><code>&lt;event&gt; no_route_found &lt;/event&gt;</code></td>
</tr>
<tr>
<td><code>&lt;action&gt; reselection-slow&lt;/action&gt;</code></td>
<td><code>&lt;action&gt; rediscovery_swift&lt;/action&gt;</code></td>
</tr>
<tr>
<td><code>&lt;target&gt;binder&lt;/target&gt;</code></td>
<td><code>&lt;target&gt;binder&lt;/target&gt;</code></td>
</tr>
<tr>
<td><code>&lt;condition&gt; &lt;/condition&gt;</code></td>
<td><code>&lt;condition&gt; &lt;/condition&gt;</code></td>
</tr>
</tbody>
</table>

(a) (b)

<table>
<thead>
<tr>
<th>Policy context_change</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;event&gt;context_cng&lt;/event&gt;</code></td>
</tr>
<tr>
<td><code>&lt;action&gt; rediscovery&lt;/action&gt;</code></td>
</tr>
<tr>
<td><code>&lt;target&gt;binder&lt;/target&gt;</code></td>
</tr>
<tr>
<td><code>&lt;condition&gt;memory&amp;lt&amp;100&lt;/condition&gt;</code></td>
</tr>
</tbody>
</table>

(c)

**Figure 7.6 Few Policy specifications used in CASP** (a) Policy rebind-reselection-slow (b) Policy rebind-rediscovey-swift (c) Policy context_change

Figure 7.6 shows some of the policy specifications used in this thesis. (a) Policy rebind-rediscovey-swift defines that if no-route-found event arises then it invoke the action i.e. the method rediscovery_swift of the target as binder. In the case of reselection-slow policy when the event new_service_available is raised the target binder’s method reselection_slow will be invoked. In the case of context_change event the context_cng method of the binder is invoked.

Rebinding to needed services must be properly rearranged to maintain accessibility when users and service components move to new locations. Once the service is found it has to be
bound to client. The module binder will take care of the adjustments to the reference objects and other low level activities.

*Adaptation Manager Module* of CASP supports policy specification and enforcement. To enforce binding and copy strategy policies, the Adaptation manager detects changes in operating environment that are relevant for binding management. It then notifies policy subjects about the event occurrence and interprets policy specifications to activate appropriate low level management actions. Components of the adaptation manager are shown in Figure 7.7.

Adaptation manager consists of three sub modules: (i) Interface for the service provider and application developer to create & modify the policies. (ii) The Policy decision point and (iii) The Policy enforcer

The Interface for service provider and user agent exploits the tools for editing, updating and removing policies. There are tools to transform high-level policy specifications into low level policy representation. This module generates individual java objects for each policy. After creating a policy object it stores and distributes it to the interested subjects. If a new policy is entered its corresponding code must be provided. In this way, the framework can extend the set of behaviours it provides with great flexibility. Service providers can enable the *user agent* to access needed services, even when disconnected by specifying *copy policy* at service deployment time. When the *user agent* needs to disconnect from the current attachment point, *copy policy* commands the Binder to move needed services to the device if the device has enough disk space.

*The Policy decision* point module coordinates policy enforcement. It retrieves newly instantiated java policy objects and parses them to retrieve relevant information: events, subjects, targets and actions. It then registers the significant events with the event service on behalf of the policy subjects. Whenever the event occurs, the system dispatches events to all the interested policy subjects.

The policy enforcer enforces polices. In particular event service notifies a subject of policy event occurrences and the subject delegates the adaptation manager to interpret the triggered policy specifications. Policy interpretation comprises of parsing policies, controlling dynamic conditions for policy applicability, extracting policy actions and notifies the binder to activate the corresponding operations in the system.
*Policy Enforcer* is the core part of policy implementation. It is in charge of making final decision for choosing service provisioning, based on context information. The adaptation to environment is triggered, managed and activated by the policy enforcer. The Policy enforcer needs to respond to the following questions such as (i) when to apply the adaptation? (ii) how to transparently do the adaptation?

Since this is goal oriented that the services in the request list has to be satisfied, the specific demand has to be formulated by the applications of the environment in the form Application Profile. Users specify their demands explicitly and the environment tries to respond to these demands. A detailed discussion has been presented in thesis author’s paper (Ponmozhi (2012)).

*Binder* module is in charge of performing the management operations seamlessly to rebind the moved requesters and service providers. It is working as inter/locality forwarders for service requests/replies.

The binder dynamically readjusts bindings between user agent and needed services. The policy decision triggers binding rearrangements according to the specified metadata. The binder triggers the creation of the reference object that uses java RMI. In the case of copy movement strategies the binder forces the creation of reference objects that exploits java serialization and either the service or its copy is serialized. In case of rebinding the reference object initiates the checking of availability of the service equivalent i.e. it starts re selection or rediscovery of services. Service table is the data structure that stores all the active descriptors for that user agent. This table will be altered by binder according to the binding adjustments.
Figure 7.7 Components of Adaptation Manager

*Provider agent* is responsible for providing and executing services. It consists of a Service Registry and Remote Invocator. The service registry contains the specifications of all locally-provided services. It also caches the specifications of services provided by neighbouring nodes that have recently been advertised by other nodes. This reduces the overhead of broadcast for service discovery by increasing the likelihood that a matching service is found in the service registry of the host node itself.

When a script invokes a remote service, the user agent sends the input parameter to the Provider Agent on the device that provides the service. The remote invocator component within the service provider receives the input parameters and passes them to the service registry, which executes the service.

*User agent* is responsible for discovering, matching and invoking services on behalf of tasks. The user agent consists of binder, binding table, adaptation manager, resource manager and monitors. The resource manager is responsible for finding services and enforcing the binding by invoking policy enforcer. The Resource Manager calls the matcher to find the best service. If a client wants to consider the energy efficiency in the service selection process, it can include an energy attribute in its service specification, forcing the user agent to select a service that meets the energy-efficiency specification. The best service provider will be selected based on the equations mentioned in chapter 5.

When a service provider is selected, its address is stored in the binding table. The binding table maps the service specification to the provider that will perform the service. It is updated when the resource manager finds a new service provider and when a task explicitly unbinds from a service.
The user agent carries out the actual invocation. It takes the input parameters and sends them to the service provider and waits for the results to arrive. Once the result arrives, it can pass it to the application which can then process the results. If the results do not arrive within a certain time, the user agent aborts the operation and notifies the task of error to the application. The user agent monitors the continued presence of the service provider and notifies the task if the service provider is disconnected while the service is still being involved.

7.4 Implementation

A policy-based dynamic adaptation system should make it possible to edit the rule set and have them re-interpreted to support the dynamic addition of new rules or change in policy. User agent in CASP provides facilities to do policy editing.

Event driven execution model makes it easier to construct adaptive services. Policy enforcer is a class with many event handlers for each event. These event handlers are executed when it’s corresponding event occurs. Events may be raised by monitors, or advertiser module when-ever it adds a new service provider or may be explicitly invoked when the mobility is recognized and the reselection or rediscovery policy enforcement decision has been taken by the decision point. Java runtime will handle the basic event handling operations. The framework provides a number of operations for handling mobility. These include bind () which specifies a handler to be executed when a particular event occurs and raise () which raises a specified event. Arguments to be passed to the handler can be specified in both the bind and raise operation. Other operations are also available for creating and deleting events, unbinding a handler from an event.

Following are the major events that can trigger functionalities of policy enforcer:

new_service_available. Whenever a new service provider for that service type comes in to the requester node, after adding that service in the table it will raise new_service_available event.
**rebind_rediscovery:** The changes in the environment may lead to adaptation of applications or the working feature of the framework. Service adaptation replaces the vanishing services with others published by other providers with different non-functional properties.

**context_cng:** The application may decide to check for a particular context change and if those context changes are found then the event context_cng event will be raised by the monitor.

**no_route_found:** the adaptor for Network layer is used to check whether there is any route available for the service provider. If no routes found, then this event will be raised.

The modules such as binder and adaptation manager facilitate rebinding. The binder dynamically and transparently readjusts the bindings between the service provider and the requester. The adaptation manager triggers binding rearrangement according to the specified policy.

Binder mediates the requesters’ access to the service provider and dynamically adjusts binding strategies according to the specified policies. In the case of copy movement strategy, the binder accesses the objects from the service provider. It uses Java’s serialization and de-serialization concept. The methods in the binder class are: rediscovery_swift, rediscover_conservative, reselection_fast, reselection_slow, reselection_medium, context_change.

### 7.5 Experimental Evaluation of Service Binding and Mobility Management in CASP

The proposed work was modelled in a computer simulation environment, by defining a framework for representing the network, mobility and service discovery scenarios and measuring proper performance metrics. The proposed model is simulated in NS-2 v 2.31.

The simulation is done with 100 nodes randomly placed in rectangular area (300m X 2000m). Nodes were moving randomly with random speed and with no pause time. The maximum speed a node can travel is of 20m/s. Each node might contain one service. Requesters communicate with servers by sending 100 byte packets at a Constant Bit Rate (CBR). The
simulation was conducted with 1, 2, 4 & 6 number of service providers and number of requesters 30, 50 and 70. The results are averaged for five movement scenarios.

Effects of motion on the rediscovery and reselection strategies were studied. Figure 7.8 shows the performance of rediscovery strategies in heavily loaded network. SLP-CENT-PING is a version that ping all known servers every 60 seconds and selects the one with the shortest round-trip-time. The results for a 60 second pinging period were presented, because this is the period that achieves the best throughput for the network defined in this simulation setup. SLP-CENT which selects servers at random is included for comparison. All protocols run on a network with 4 servers and 50 clients with CBR sending rate of 7.5 packets per second. Nodes move with a Vmax of 20 m/sec. Experiments with Vmax to set to 2 m/sec shows similar trends and therefore not shown here.

CASP-medium achieves the best performance. Since no additional packets are being transmitted into the network, CASP-fast suffers from network congestion as a result of sending rediscovery requests as soon as the active route breaks. The slow policy on the other hand is too conservative. Because of the network mobility, Holding rediscovery until all routes to all known servers have been tried makes the client wait for a long time before doing a rediscovery and resuming sending data to a service provider. Slow also chooses longer routes over a rediscovery leading to poor locality.
Figure 7.8 Effectiveness of Reselection Strategies

Figure 7.9 Effectiveness of Rediscovery Strategies
As shown in figure 7.9, CASP-Swift achieves the best performance in heavily loaded network. Since no additional packets are being transmitted into the network, CASP-Swift does not incur penalty for switching to the best possible services as it becomes available. CASP-conservative on the other hand waits for the active route break and thus misses opportunity of switching to better servers.

SLP-CENT-PING shows that server rediscovery based on timing measurements such as pinging can increase network throughput. However, a substantial gap still remains between the best cross-layer rediscovery technique and the application layer implementation. Two factors account for SLP-CENT-PING lower performance. First, pinging adds a significant amount of message overhead to an already heavily-loaded network, creating more congestion and interference. Second, to be effective the pinging period needs to be compatible with the rate of network topology change.

Figure 7.10 and Figure 7.11 shows results of rediscovery and reselection strategies for lightly loaded network with 4 servers and 50 clients with a low CBR sending rate of 2.5 packets per second. Nodes move with Vmax of 20 m/sec. SLP-CENT-PING performs better in this less congested scenario.
Figure 7.10 Effectiveness of Reselection Strategies in Lightly Loaded Network

Figure 7.11 Effectiveness of Rediscovery Strategies in Lightly Loaded Networks

Figure 7.12 shows the performance of rediscovery strategy CASP-Medium for networks with 1, 2, and 4 servers. In all cases, there are 50 clients with CBR sending rate of 7.5 packets per second. Nodes move with a Vmax of 20 m.sec.
These results show that CASP-Medium is able to keep traffic localized despite network mobility and therefore can take advantage of an increase in the number of servers.

In summary, in mobile ad hoc networks, even with an optimal initial service selection, throughput of network degrades to that of random selection. This shows the need for effective rediscovery and reselection techniques to offset the effect of topology changes and improve network performance. Physical routing information such as route lengths, context information such as battery power and moving speed have proved to be the best tool for localizing traffic during mobility. Round-trip-latency measures were shown to be effective in cases with small network saturation, but using them too often leads to congestion. Moreover, for heavily utilized networks roundtrip latency measures are ineffective and fail to localize traffic. In addition, periodic rediscoveries fails to capture the rate of network topology change, thus re-evaluating server status either too often or too rarely. Also, pinging adds significant amount of overhead to an already congested network, causing even more interference. In contrast, triggering rediscovery and reselection after detecting changes in the topology prove to be effective.
Summary

Service invocation and adaptation mechanisms of the proposed framework were presented in this chapter. Interfaces have been designed for the users to input the (i) policies required to rebind the requester to a new service provider and (ii) conditions when to do rebinding. Methods to find the changes in the environment namely monitors has been designed. These monitors will trigger the actions to be performed when a condition satisfies. Invocation and response message formats has been designed. Inquiry message to know the current status of the service has been designed. Rebinding policies namely reselection and rediscovery has been implemented. This work has been compared with SLP-PING protocol and it shows 25% improvement in throughput in a heavily loaded environment.