Chapter 6

NON FUNCTIONAL PROPERTIES
MANAGEMENT IN CASP

Introduction

Service oriented architecture fosters the vision of a dynamic binding in order to take advantage of the variability of available services (Kristof Hamann et al (2011)). Service selection is often not only controlled by functional requirements, but also driven by non functional requirements and offers. Non functional properties of a service denote all aspects which can be used by requesters in order to evaluate service quality. Non functional properties describe quality of service as well as context of service execution. Although there are approaches considering only quality of service or context during service discovery and selection, there is a lack of systems taking both kinds of non functional categories into account. In this

The quality of service refers to how well a service performs and is encompassing performance and network related characteristics as well as price models and costs. Context information is based on the execution environment and that of the service’s behaviour. For example if the requester wanted to print a document and search for print service. The service request for the print service may have quality of service properties like price or availability and some printer specific context information, which are refer to the result of the service, the printed document e.g. color depth, resolution. Such criteria are for example the price of the reservation service or the location of the printer service. In contrast to quality of service attributes; context information highly depends on the service domain. Thus the schema for context information will always be domain specific.
Non functional requirements of a service play an important role to differentiate among services of same functionality but which differ with respect to the user’s current situation (Kristof Hamann et al (2011)). Non functional properties can be used to define the quality of the service as well as goodness of the services for example the price, performance; bandwidth the consumption etc. of the service. The Non functional properties incorporate requirements of the requester, provider and the network participants. On the one hand the provider can use these properties to specify the services’ quality. On the other hand the client/requester uses them to specify their constraints.

Suppose the constraint is based on the location of the user, the framework has to provide a matching mechanism to match the use’s context such as location with the Non functional properties of the service provider. One of the major challenges to achieve context aware service provisioning is to build the connections between client context information and the service’s non functional properties. In Service provisioning, context relates to two things (Zef Hemel (2006)) (i) the context and other non-functional requirements of the consumer (ii) non functional properties of the service being discovered. Several parameters such as location, time, user identity and profile, device capability that involve both the requesting user as well as the service provider have great influence in the search for the most suitable service. This fact highlights the need for context management to support efficient deployment of applications that strongly depend on mobility of users and data source. Context plays the role of a filtering mechanism, allowing to short list the services. In order to account for context information, such as user location, proximity and other Non functional properties, the framework which facilitates service provisioning should provide integrated support for context modelling and acquisition. In particular adequate expressiveness for rich and unambiguous representation of users’ their contexts and the network they participate in, are very much needed.

Since user context information normally is quite different from service non functional properties and they are defined by different groups of people. For example the user context may specify that a user has a mobile phone; but the service non functional properties do not have the same descriptions because services may only describe what types of messages they can send. Therefore, there should be a way to link these two. As far as this thesis is concerned, context information consists of active and passive user context and service context. This chapter
discusses the way in which the context can be collected and how to utilize them for service provisioning. A brief discussion on non functional properties is provided in 6.1, Overview of context management and accessing of context is given in 6.2. Section 6.3 discusses the meta data used in the proposed framework. In 6.4 accessing the values of non functional properties were discussed. Service selection using non functional properties were provided in 6.5. Implementation of the non functional properties management in CASP framework is given in 6.6. Finally in 6.7 the evaluation of this work is presented, which is followed by the chapter summary.

6.1 Overview of Non Functional Properties

Stakeholders involved in service provisioning

The stakeholders involved in the process of Service provisioning in Mobile ad hoc networks (Luiz Olavo Bonino (2011)) are:

Service requesters: This entity is responsible for requesting and utilizing the services. It has to provide negotiable terms for the service providers’ offer.

Service providers: This entity is responsible for service itself. The provider is committed with the execution of the activities described in the service advertisement.

Service beneficiary: This is the role played by the primary entity that perceives the benefits of the service, the role of service client and beneficiary may be played by the same entity.

Service executor: This entity is responsible for executing the activities related to the service the role of service executor and service provider may played by same entity.

Context provider: This entity is responsible for supplying mechanisms that allow the supporting platform to request and transparently gather contextual information of the users. These mechanisms include information gathered from user’s personal information sources, from the node (device) involved.
The proposed framework considers the service requester and the beneficiary one and the same. The important challenge of the Service provisioning in the multi hop mobile ad hoc networks is giving due consideration to the capabilities of the stakeholders involved for better performance. Figure 6.1 specifies various stakeholders considered in this thesis and the information provided by them.

![Figure 6.1 Stakeholders of Service Provisioning Process](image)

Satisfying the client, or in other words the application is the primary aim of the service provisioning. Application developers have to have periodically updated information about the execution environment. Application developers should be able to specify which services they are interested in, their preference on the execution parameters, the behaviour to adapt on particular context change. This information can be specified in the form of application profile. Service providers have to specify service descriptions, non functional properties etc. and sent them in the form of service advertisement refer to section 5.2.1 for advertisement creation.
**Service provider redundancy**

Mobile ad hoc networks can be perceived as environments containing many service providers offering a wide variety of services, where it becomes likely to have more than one provider giving the same service. This situation is the result of having different providers host the same implementation, or providers hosting different service implementations having same invocation interfaces. An example is given in Figure 6.2

![Figure 6.2 Provider Examples Offering Same Services](image)

1. Providers $P_1$ and $P_2$ have installed the same service $S_1$ and so they are offering the same functionality
2. Provider $P_3$ has installed service $S_1$ which happens to offer the capability as that of $S_1$ in other providers but has different implementation and non-functional properties but provides same interface. Here $P_1$, $P_2$ and $P_3$ are same.

**Non functional properties of Services**

Non Functional properties of a service denote all the aspects which can be used by requesters in order to evaluate service quality; they play an important role to differentiate among services of same functionality but differ with respect to the user’s current situation. In service based applications, non functional properties of services can drive the design of the applications and the selection of the concrete services from which they are built. The ability of the service requester to choose services based on non functional properties specified in service description is crucial. Ideally, there should be a standardised catalogue of the non functional properties of the SOA services. Some of the non functional properties that were relevant to the requester’s perspective (Hanane Becha (2012)) are: price, response time, reputation, certification, standards
compliance, failure modes, transactional service, security, service versioning, resource
requirements and scalability and server location. These properties can be grouped into the
following heads.

*General properties*: are provided by service providers to describe the provider’s
information such as name of the provider, the location, service versioning and price of the
service and other related properties.

*Usability properties*: may have the elements of availability, communication protocols and
domain specific elements. Availability indicates whether the service is available or not. Policy
tells who and where the service can be used or not to be used. Communication protocol specifies
how the services communicate with the surrounding environment such as SOAP or REST.

*Reliability properties* may have the error rates and stability. Error rates shows chances of
getting unsuccessful results form a service. The stability represents the long term stable
performance of a service. The properties like security, reputation, certification are comes under
this.

*Efficiency properties* may have the execution duration and accuracy. The execution
duration describes the speed of the service for completing the task. The accuracy indicates the
freshness and correctness level of the data produced by the service.

*Domain specific properties* define the extra considerations for some specific types of
services. For example print service may have coloring option whereas weather service will not.

---

*Modelling the Non functional Properties*

Model for non functional properties are used by service requesters or passive users to
objectively distinguish services based on their non functional properties to make the most
appropriate choice amongst a number of services with equal or similar functionality. In the light
of the above statement model for non functional properties is required, which can be used in service descriptions as well as service requests. Due to versatility of non functional properties, it is unlikely to identify a complete set of non-functional properties. Every non functional property will be considered differently depending upon the specific service domain. In general, the nature of Modelling should posses the following:

Properties’ preferences: Service requesters usually have various preferences for the non-functional properties depending upon the situation they are in and of course different situation will mostly have different preferences. A good mechanism should not only express values for each property, but also preferably represent the relations among the preference. For example, in the emergency situation, a financial service may require to consider the security property as more important than privacy. Hence there should be mechanism to specify preferences for different situations.

Evaluation of properties: It cannot be predicted how many non-functional properties will be available, as well as types of these properties. For example, the evaluation function to compute the speed criteria will be different from the function to calculate the location criteria. Hence evaluation framework must not only adaptable to the varying number of non functional properties, but also automatically identify the measurement methods that should be used to evaluate each non-functional property.

Dynamic aggregation: when all desired non-functional properties can be evaluated; the next step is to aggregate individual scores to gain a final score for the service. A suitable aggregation method has to be selected.

Automation: A service designer would specify data for the service when making it available and a user would still be able to specify requirements, but selection would be performed without human intervention. Such an automatic service selection methods are essential when considering context-aware service selection.

6.2 Overview of Context Information
In the environment like mobile ad hoc networks, context can be defined as a set of hosts and their associated properties that are of interest to a given node, (i.e) the nodes that can affect its behaviour, can communicate with it, or can carry out activities on behalf of it. Context can be categorised into three categories (Hanane Becha (2012)).

**User context:** context information related to the service requester such as Personal context: health, schedule, activity etc. Social contexts: group activity, social relationship, people nearby.

**Device context:** contextual information related to physical aspects of the context-aware system such as (i) Physical context: location, time etc., (ii) Environmental context: weather, altitude, light, (iii) Informational context: stock quotes, sport cores etc.

**Computing context** contextual information related to computational aspects of the context-aware system such as (i) Application context, (ii) System context like network traffic, status of resources, bandwidth and quality of service.

The main task of any system with context awareness is to acquire and utilize the context information in order to provide services that are relevant to the place, time, user etc.

The reasons for using context information are:

- Service selection and task adaptation: context information is mostly used to select the most suitable service and task to perform actions in a given situation. One of these purposes is how to use context for service provisioning (Maamar (2004)).
- Security and privacy control: context information is used to support adaptive controlling security and privacy management (Wang (2008)).
- Communication adaptation: Context information is used to select communication protocols and optimize the communication.
- Content adaptation: context information is used to adapt content resulting from a request and to return the content in a suitable form to the requester of the context.
This thesis uses context information for service provisioning which has the service discovery & selection and maintenance phase. Service discovery in mobile ad hoc environments is a complex task as it requires facing several technical challenges at the state of the art, such as user/device mobility, variations in service availability and environment conditions and terminal heterogeneity. Users might need to discover services whose names and specific implementation attributes cannot be known in advance, while service providers need to advertise services to the clients whose technical capabilities and conditions at interaction time mostly unpredictable beforehand. Discovery solutions can be improved and brought to their full potential by doing a paradigm shift from an administrativescoped to user centric approaches (Arbanowski et al 2004). This thesis uses the user centric data in the form of user context and resource context.

User Context

In order to select the most suitable service for the user correctly modelling the user context is the first important step. In his article, Shilit (1994) summarised the problems of context modelling. Firstly, there is a need for suitable context model, which describes the relationship between different types and facilitates inference and abstract context. Secondly a suitable, easily developed infrastructure for context acquisition and management is require, which separates the acquisition from the utility of context.

The user context stores user’s personal data and contact information such as

ContactInformation: stores the address, device, preferable contact method. These properties are related to the service’s non-functional properties of how to contact the person and where to contact the person in a dynamic environment.

GeneralInformation: saves user’s id, name and other status. This is used to differentiate the users.
**WorkPlace:** holds the information of the user’s working environment, which defines the role of the person as each role may, has different services for the same task. Set of services assigned for a person may vary based on the role.

**Language:** shows the languages this person knows; in selection aspect this property can restrict the values of language support this person needs. During selection this property may be used to select the set of services only those which are supporting these languages.

**Resource context**

This includes the details of the resources that the person has. The resource context connects to service’s non functional properties, such as which devices are available currently and which communication type is the fastest one.

**Electronic resources:** This has a group of properties in it. Each electronic source property has property such as *devicetype*, which holds information of what electronic devices the user currently has. The *devicetype* also includes details related to the software, device capabilities. For service selection this information is useful to support the matching between user’s electronic devices and services’ usability and also the location of this device.

**Physical resources** stores the information about hardware of the user such as internet bandwidth, memory etc. This information is related to the restriction of service’s speed and hardware requirements for using the service.

**Location** states the location information to perform this activity which links to the location or distance constraints of the service. This should have one of the values specified in the domain regions.

As for the service data the user information will also be divided into capabilities and requirements. For example the internet bandwidth, display size and memory etc. are its capabilities, but if the Bluetooth connectivity is the only means by which the other devices can be connected to, then that is the requirements of the other device to have Bluetooth connectivity.

This framework consists of an entity specification which provides a logical representation of physical spaces (domain) containing on or more regions and device-types, each region
represents a discrete physical space that exists within a domain (such as a room in a building) and each device is a specific type of device that at any point in time resides in a region with the domain. Figure 6.3 provides the device specification hierarchies.

The context model consists of a domain schema specification and a device context profile specification which together provide a way to specify the context of the entities in an environment. A domain schema specifies a single domain in terms of its regions and device types. A device context profile is a collection of properties that describe a particular device’s contextual state. Devices with in a domain use context profiles to share information with one another. Each device must have a device context profile (device schema) from which it generates its context profile. Each device schema in turn must be on domain ontology.

Domain schema specification provides a common vocabulary and structure to be used by all devices with in the domain. They are defined as XML instance documents. A domain schema explicitly defines a domain, which contains a set of one or more regions and a set of one or more device-types. Domains and regions are specified as simpleType XML Elements of type “string”. Device-types are more complex each having a unique name and a set of properties.

```
[Diagram: Flowchart showing the structure of a domain with layers labeled Domain, One or more region, One or more device types, and One or more properties]
```
The domain Element specifies the domain being described and should be given a name. The region Element specifies the region(s) that exist within the domain that is being described. The device type Element represents the possible device-type(s) that may exist with the domain. Each device-type has a “name” attribute and one or more the properties. Each property will have the name, data type, value and one or more optional attributes. Whenever the optional properties are omitted their default values will be assumed. The entire domain schema should have a DomainDeviceType which contains domain-wide properties. This device type should include “DeviceType” and “Location” properties. The values of these two types are restricted by the domain’s available device-types and regions.

**Computing context**

The information related to the system’s internal resources, link status, bandwidth etc. These are dynamic in nature. To provide these details and ease the application development, this proposed framework shoulders the problems of dealing with lower layers and provides the required information to the application. The framework, on behalf of the applications takes care of monitoring the environmental changes and configure according to the behaviour specified by the application. It should detect the changes and adapt automatically, i.e. act according to the designers’ specification. Since the application may require some services to be delivered in different ways, when requested in different context.

All these non functional properties can further be divided in to static and dynamic based on their time of availability. Some properties like Language known of the user and the provision of the service language may be static and available before execution but the status of the internal resource, bandwidth are changing and has to be accessed dynamically to know their values.

Schilit (1994) highlighted the challenges caused by changes in the context information and reaction to these changes. The difficulty of addressing this problem is to cover the gap between the context information and the reaction resources. In Service Oriented Architecture, suitable
reaction resources are the service properties (Functional and Non Functional) and selection methods in the service domain. Normally, the user context is modelled by client experts while service non functional properties are defined by service providers. Therefore, there is a gap between the user’s context information and services’ non functional properties.

In order to bridge the gap, schema to represent the service both non functional and functional criteria were defined. The preference value for these properties can be specified by the user in their request and it is assumed that the same schema is available to the user and the Service provider.

6.3 Metadata Used for Non Functional Properties

Context-aware applications have traditionally been developed using one of the following approaches:

1. Each application directly communicates with sensors and other resources of the context information; pre processes the raw data to the required level and evaluates the information to make decisions about how to adapt.

2. Applications are developed with the aid of reusable libraries/toolkits for processing context information which assist with gathering and pre-processing data (implicit context model)

3. Application have their own well-defined context-models and use a shared context management infrastructure to populate the models at run-time using context sources (explicit context model)

In the first two approaches it is the responsibility of context-aware applications to handle faults in context information gathering and pre-processing, which increases the size and complexity of the applications and the difficulty of implementation.

The third approach is based on explicit formal model of context information used by particular applications and therefore allows multiple context-aware applications share a set of common context sources and context information pre-processing components. In addition owing
to the models, it allows the issues of fault tolerance, re-configuration and self-healing to be pushed to the context management system.

The most appropriate models for autonomic context management systems are those that define not only the types of context information required by the application but also its metadata for example classification of information types into sensed, non-sensed and derived types etc. that can be exploited to define dynamic, context-aware mappings between context models and their appropriate sources of information.

Two types of context facts are presented in the context model proposed in this thesis they are profiled, sensed and derived fact types. Profiled information is user-supplied and is therefore initially very reliable but often becomes out of date, while sensed context information is usually highly dynamic and prone to noise. Derived information is inferred from other contexts using a derivation rule. This classification of information types allows context information to be managed and processed according to the characteristics of its type.

This thesis uses two types of metadata namely profiles and policies. Figure 6.4 provides the meta data used in CASP framework.

![Figure 6.4 Metadata used in CASP Framework](image-url)
Policies manage and control choice in resource-binding strategies. The discussion on policies (the one shown with dotted lines) will provided chapter 7. A profile is a modular structure and it is composed of different parts: identification, capabilities, requirements and bindings. Profiles describe user requirements and device properties. In particular, user profile includes information related to user’s context. Device profiles report access devices hardware and software characteristics. Service profiles describe service interfaces as well as the properties that could be useful for binding decisions, such as whether the service can be copied or migrated over the network.

Metadata used in this thesis are specifically designed to support personalised user-centric discovery, so the proposed model focuses on the representation of service/user/device capabilities and requirements where as other parameters such as service preconditions, post conditions and assumptions are not taken into account as they are more relevant to service composition. As a key feature the Meta data model provides fine-grained profile modularization to improve precision and effectiveness in service selection. The comparison between the request and the offer is performed by considering single capabilities and requirements of interest rather than the entire service profiles. This framework allow users to further personalise service discovery by specifying preferences in advance a priority order amongst requested service capabilities and thus relieving from manual selection and ordering discovery results.

Further meta data can be divided into static and dynamic profiles. Static profiles hold attributes that may not change over time. They may be available before the execution starts and will be stored in the respective repositories. Whereas Dynamic profiles hold attributes related to context information and are represented in the same way as static profiles except their value will be determined whenever needed. The service matcher uses dynamic attribute realise to obtain their values before making matches. There may be two sides to a dynamic attribute to be obtained: the requester’s context constraints or the service provider’s context requirements. The requester’s dynamic attributes are either explicitly contained within the service request or implicitly provided by the repositories.

*Service Metadata*
Each service is described by a static and a dynamic profile. The static profile collects data that are relatively stable over time, such as service name and functions, or do not depend upon dynamic operating conditions. Conversely, the dynamic part of the profile includes information that might be change frequently, e.g. location and state of the application. Figure 6.5 provides the schema specification of service profile.

**Static profile**

Each static profile consists of service *identification, capabilities, requirements and service interface*. Identification information provides the details such as name of the service and identifies its location. Service capabilities basically represent logical units of service functionalities. Service capabilities allow providers or advertisers to describe their services in terms of offered functionalities. Conversely, users express their requirements in terms of requested capabilities. Service requirements represent access conditions to a service. Each requirement is modelled as a required capability i.e. it defines which kind of capability a client should possess to be allowed to access the service. For example a remote software update service might require that the client device runs a previous version of the software needing update. The framework uses the weighing factor for each requirement. The service provider may specify the mandatory requirements by assigning 1 and in the range 0 to 1 for other secondary requirements based on their priority. The proposed data model uses base schema for capabilities and requirements, while application specific schema can be defined for each application scenario.

The third part of the static profile is service interface. It includes the needed information for actual service invocation, such as input/output descriptions collectively called as functional properties. That information might be provided according to different specifications, depending on the interface implemented by the service e.g. a method signature for a java object.

**Dynamic profile**

The dynamic profile describes service properties that might vary over time. In particular, it includes information about the service operating conditions. These conditions are dynamically retrieved via external information sources. For example, information about service availability,
the average response time to service invocation. Pseudo code for creating service advertisement
has been provided in figure 5.2.

```xml
<service ID="printservice">
  <static>
    <capability>
      <identification> </identification>
      <general>
        <parameter ID="price">
          <Pname rdf:datatype="&xsd:string">serviceFee</Pname>
          <Ptype rdf:datatype="&xsd:string">numerical</Ptype>
          <pvalue rdf:datatype="&xsd:string">0.5</pvalue>
        </parameter>
      </general>
      <reliability> </reliability>
      <efficiency> </efficiency>
      <security> </security>
      <domain_specific> </domain_specific>
    </capability>
    <requirement>
      <hard>
        <io-cap>... </io-cap>
        <software>... </software>
      </hard>
      <soft> ... </soft>
      </domain_specific>
  </static>
</service>
```
Figure 6.5 Service Profile

User Profile

Service requesting nodes are described in terms of profiles and preferences. User profiles are composed of both static and dynamic metadata. Dynamic properties include for example user locality and state, while the static part has three categories identification, capabilities and requirements. Identification information such as ID Code, a string or URL is used to name and identify the user. Capabilities represent the user’s capabilities and abilities such as language the user can understand. User requirements describe the user defined constraints that must always be satisfied during service provisioning.

Preferences allow the user to express desired choice about service requirements. The framework uses value preferences to specify the preferred value for a service property. Since a request for a service may include multiple capabilities it is allowed the user to define a priority order among the preferences.

Device Profile

Device metadata describe the technical characteristics and operating conditions of a user device. Similar to service profile and user profile the device profile includes static and dynamic parts. It consists of identification, capabilities and requirement parts. The identification part includes device type, names and parameters that allow the device identification with in the network such as MAC address or the Bluetooth ID and device location as per the domain schema. Device capabilities describe the technical characteristics, supported functions and resource state, such as memory, storage size and battery level. Finally, the device requirements that other devices must meet to access services hosted on that device. For example if a device is able to connect to another device only via Bluetooth, the Bluetooth connectivity represents a requirement for the device.
Applications can influence the framework in two ways. (i) **Changes in the execution context:** The application may expect the framework to listen to changes in the execution context and react accordingly. This may require the framework to monitor and raise exceptions accordingly. (ii) **Binding decision:** Since there are many different ways that a particular service is provided. The way in which this service needs to be provided can be specified in the form of policies such as ‘copy’, ‘remote invocation’. Some services needs to be provided in a better way by switching the Service providers if the pervious service provider is not accessible. This can also be specified using rebinding policies such as ”swift”, “conservative”, etc.. Application profile specifies the list of services it needs and their binding policy requirement, type of the application. In general, domain knowledge of the application is provided in this profile. Detailed discussion on this has been presented in the thesis author’s paper (Ponmozhi (2011)).

### 6.4 Accessing the Values of Non Functional Properties

The proposed framework has mechanisms gather information from network and operating system & present it in the format the application could understand. The information is provided either as static (in the form of meta data) and dynamic which has to be accessed during runtime. In this section explains how to access the data from meta data or during runtime.

#### (i) Meta data processing to access Static properties

In the case of static meta data all the information provided in the form of profiles are parsed and stored in user, device and service repositories. Following Table 6.1 shows the different properties, the entity which define them and form in which they will be provided to the framework.

<table>
<thead>
<tr>
<th>Type of context</th>
<th>Available as</th>
<th>Defined By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Context (static)</td>
<td>Device profile</td>
<td>Device manufacturer</td>
</tr>
<tr>
<td>User context (static)</td>
<td>User profile</td>
<td>User</td>
</tr>
</tbody>
</table>
Table 6.1 Context and their providers

(ii) Cross layering to access dynamic properties

Context management comprises of all the phases from acquisition at lower layers to the offering to the applications/other entities which require the context information. Context aware functionality should only have to produce and send context data and declare their interest to receive them from the supported middleware, while an internal middleware function takes over and transparently executes specific management operations (Paolo Bellavista et al (2013)). Explicitly provided context information can be from many sources such as sensors, databases, or from lower layer protocols. Format of these data may vary depending on the source as well as their capability to store or maintain the data will also vary. So we need to change the formats, store the collected data for later use in an abstract way useable by the middleware. The static data such as user profiles and preferences will be parsed and stored in the database by the parser module.

Cross layering is an innovative form of protocols interaction, placed besides strict layer interactions. In the classical layered model the layers are designed to work independent from network and application constraints. But there are functions which cannot be assigned to a single layer, but should be spanned across many or all layers in the stack.

An adaptive cross layer protocol stack can facilitate the link layer to adapt data rate, transmit power and coding to meet the requirements of the application based on the current channel and network conditions. MAC layer can adapt based on underlying link and interference conditions as well as delay constraints & bit priorities. The application layer can utilize the notion of soft quality of service that adapts to the underlying network conditions to deliver the highest possible application quality. In order to support this adaptation information must be exchanged across all layers. This information exchange allows the protocol to adapt in a global
manner to the requirements of the application and the underlying network conditions. Following Cross-layer information is used in this thesis.

*Speed* of the nodes can be calculated by itself by using the below mentioned formula (6.1). This value is averaged for ‘n’ time slots. The variables Xi, Yi represents the geographical co-ordinates of the positions of the mobile device. It is assumed that there is a way to find these values.

\[
\frac{1}{n} \sum_{i=1}^{n} \sqrt{(X_i - X_{i-1})^2 + (Y_i - Y_{i-1})^2}
\]  
(6.1)

*Bandwidth available* in a node can be estimated by the node themselves. Since the bandwidth is shared among neighbouring hosts, using IEEE 802.11 MAC in mobile ad hoc networks estimating the available bandwidth is very difficult. Each node listens to the channel to determine the channel status and computes the idle duration for a period of time ‘t’. IEEE 802.11 utilizes both physical and virtual carrier sense. In the physical carrier sense, the channel state can be had from the physical activity of the channel. IEEE 802.11 wireless radio has two states: (i) Busy state \(T_{busy}\) (transmit, receive, carrier sense) (ii) Idle state \(T_{idle}\)

The nodes will constantly monitor the channel state changes. It starts counting when the channel state changes from busy state (transmit, receive, carrier sense) to Idle states and stops counting when channel state changes from Idle to Busy state. The node adds all the idle periods over the observation interval ‘t’ to compute the total idle time. Let \(B\) be the raw available bandwidth the available bandwidth can be calculated using the formula given in (6.2). This method is passive a method. This thesis used passive method only. Whereas in active method the Hello packet’s received signal may be used for estimating the bandwidth.

\[
B_{avail} = \left(1 - \frac{t_{busy}}{t_{total}}\right) \times b
\]  
(6.2)
The framework uses the contextual information to (i) increase the accuracy and suitability of selected service (ii) to find binding policy for the current situation and service. The contextual information can be requested for single use or can be subscribed to, if the continuous updates of information are needed.

The framework supports interactions between service providers and service clients. From service provider’s point of view the platform supports the publication of service descriptions along with its context. From the service requester’s point of view the framework provides mechanisms for service discovery, invocation and monitoring. As the types of data may be static or dynamic nature, there should be two sets managing modules to manipulate them. Metadata manager is used to handle static context whereas, dynamic context is manipulated by *dynamic context realiser*.

*Metadata Manager*

This module provides support for the specification, modification, checking correctness, installation and evaluation of different types of metadata used in this framework. The metadata manager provides User Interface with templates to support the user in the task of specifying user/service/device profiles. The use of templates ensures that metadata are encoded in the correct format i.e. complaint to our profile schema. This user interface will be useful for profile specification in a user friendly manner even if the user is a non-technical person.

This framework supports the interaction with its stakeholders by providing a set of interface components, namely provider Interface, requester’s interface. The interface components provide APIs that allow the applications to interact with the platform. Stakeholders interact with the platform by using APIs.

*The service provider’s interface components’ API*: offers methods to retrieve available domain ontologies to be used automatically for service descriptions. A programmer creates the service’s binary, functional description and non-functional description. The interface helps the provider to host the service binary and creates the service descriptors according to the description and context properties. This service descriptor is advertised.
The client interface Components API: Offers method that allow clients to submit their required list of services, their service requests preferences to receive the results of the service execution and to enter information required by the services which could not be gathered as contextual information.

This module is in charge of collecting and processing user requests for service. This module interacts with the user to specify required service capabilities and user preferences through user agent. This framework associates one client agent with each portable device. The user agent covers various activities like retrieving the profile of its companion device/user and coordinating with resource management modules during service discovery. It translates value preferences expressed by the user at access request time. It translates the request into its corresponding object (i.e) an instance of servicerequest class.

Dynamic context realiser

This API set offers methods to manage the retrieval of available context information dynamically. It directly contacts with the sensors.

Sensing module (SR)

This component includes the entities responsible for collecting the context, measuring elements, monitor hooks inside a host operating system, cross-layer information from various layers. This module provides an abstract and unified approach of acquiring and using context information, which addresses many of the shortcomings of current stand alone cross-layer solutions. This approach of using abstracted mechanism may leverage tasks such as network & system monitoring and management in the system.

This thesis proposes mechanisms namely ‘adaptors’ to access dynamic context data across traditional protocol layers and from wider network context to determine the choice adaptation for optimal performance. The details have been presented in the international conference paper of thesis author Ponmozhi (2012). Adaptors were defined for each of the source of information. Since the amount of context data that can be gathered is large, it is inefficient to gather all of them at all times. Also maintaining all of them may not be possible. There may be situation when a particular data is not needed. So it is better to collect data only
when needed. Adaptors will be called whenever a particular data is needed. Figure 6.6 shows the interactions among components of adaptation.

As the gathered data are not readily usable, they have to undergo a simple or complex process to derive new knowledge. A simple process is like converting into the data format required which will be done at the adaptors itself before storing them in the database. Whereas, combination or aggregation of different context may also be needed wherein, this can be done by the use of monitors. Aggregator operation with the goal of merging more context data in a new high-level one is essential even if they are resource demanding, because the context changes claim for continuous updates and also they have to be handled automatically.

![Figure 6.6 Interactions Among Components of Adaptation.](image)

**Monitoring and aggregating module (Monitors)**

This module of the framework is heart of information accessing. The essence of this component lies in an event-notification mechanism that allows interested parties to exchange
information when certain event occurs. The multiplexing capacity of this event mechanism enables both the combination of information from various information sources to be made available to the serviced modules as well as the dispatching of events to multiple interested parties. The decoupling of the information collection process from the utilization of the information permits the features such as aggregation which is used to combine the collected information before it is supplied.

In order to create proper triggers for appropriate actions, monitors were defined. Monitors are simple classes which are capable of raising events when the assessed value exceeds the given threshold value of the context they are monitoring. The threshold value to be monitored can be passes as parameters. If there is a need to monitor for more than one context data then the aggregateor or aggregateand classes can be used. Three types of monitors namely Bandwidth monitor, Internal Status monitors, Network Status Monitors were defined. They can call the necessary adaptors for the current value of the context data. Figure 6.7 shows the various components used to manipulate non functional properties. The component policy enforced will be discussed in chapter 7.

Repositories used

The platform uses the following repositories namely service repository for storing service descriptions, context repositories for storing context information, the schema repository for storing the entire domain schema.

Service repository

The service’s dynamic attributes will be specified in the service’s description located in the service repository. A service can be associated with two main kinds of dynamic attributes (1) dynamic attributes which can be determined locally by the metadata manager (2) those which must be remotely determined by the service provider. An example of the first type is quality of service (number of hops or delay), the second type falls into two subcategories (i) attributes which are constantly changing (for example service state) and (ii) those which change less (for example print queue).
The metadata manager must remotely query the service provider for the current value of the constantly changing attributes or is notified by the provider when the values changes on a less frequently changing attributes. Service provider will specify each attributes category in the service description. Once the dynamic attributes have been realised they are returned to the service matcher for matching as that of static attributes. Figure 6.8 specifies various components used to manipulate non functional properties management in CASP framework.

*User repositories:* details relating to the human user of the requesting device will be stored in this repository. These may include user preferences explicitly provided by the user when he or she used the system for the first time or data implicitly collected by the architecture.

*Device repository:* each user will be associated with one or more devices. Each device has its own set of specifications; such as processing capability, screen size, input method etc. this information will be collected from the device when it registers with the system for the first time
**Context repository:** This stores diverse type of context. The storage manager is contacted to either store new context or to retrieve stored context. This module provides interface to other modules that interact with it.

**Domain repository:** This stores the domain schema to be used by the Service provider and the client interface will use this schema to verify the input profiles. These domains are assumed to be present in the system defined by the domain experts.

### 6.5 Service Selection Using Non Functional Properties

This framework uses context for two purposes (i) for service selection (ii) service binding & maintenance. The discussion on service binding has been presented to next chapter. During service discovery phase, the services will be compared with the queries. The quality of requirements like service response time, reliability and availability, which are otherwise called as non functional properties will be specified along with the query itself. In order to select the service provider, the services are populated based on the functional requirements of the user. If there are more than one service for the given request, for the inclusion in the selection list each service should satisfy all the mandatory constraints. If any one of the mandatory constraints is not satisfied then the service will be filtered out. After filtering, the service has to be selected based on their preferable constraints /secondary constraints.

**Service selection method:** finding a selection method is not difficult, however, finding a selection method which can deliver the best selection result in a dynamic, context-aware environment is difficult. Context is runtime information which demands a runtime method, which in turn needs to run with reasonable fast and be automatic.

Two difficulties concerned are (i) evaluating individual criteria and (ii) aggregating individual scores into a whole value. From the evaluation point, context information may differ from other data based on their type, e.g. numerical data, using different expression types make
the people and machine to understand its meaning easily. Because of the varying type, separate evaluation functions are needed for each type. Therefore it causes difficulties for automatic evaluation.

In the point of finding the aggregated value, different services may concern different selecting constraints with different importance levels based on user context and service domains. The constraints may be interrelated. Therefore, automatically choosing a suitable aggregation function to aggregate different aspects of the evaluation results is very difficult. Following are the groups of aggregation properties as explained by Dujmovic (2007)) namely, simultaneity and mandatoriness.

Simultaneity: simultaneously satisfying two or more requirements is the most frequent criterion in practical system evaluation. For example, a car buyer wants a car that simultaneously satisfies criteria of performance and safety. This means that a dangerous car must get lower overall score even if the score for the performance is outstanding.

Mandatoriness: some of the evaluation criteria are minimum requirements which must be satisfied. For example if the cost of using a service must be lower than Rs. 50, it means that any service which is more expensive than Rs. 50 must be discarded, regardless of how other properties compare.

Now it needs to decide

(i) How to evaluate the individual criterion for services in a context-aware environment?
(ii) How to dynamically applying the adequate aggregate function?

In order to tackle with these two questions, a suitable context modelling has to be designed to represent context or non functional properties. Context modelling is demanded for the wide range of heterogeneous context information in context-aware computing. It helps application designers and developers to uncover the possible context and simplify the context manipulation (Razzaque (2005)). Two types of context information were identified. They are (i) Measurable data with a certain range and a certain unit. The resolution in dpi and the queue length of the printer are examples for Measurable context information of our print service. (ii) Non-measurable data are constants like the quality (laser or inkjet) and the color
depth of the printed documents. Another orthogonal classification of context information is the differentiation between static and dynamic context. The dynamic context values can change during run-time, e.g. the length of the queue of the printer.

Based on these generic concepts, context domains (domain specific ontologies like PrinterContextOntologie.wsml“) can be defined and used in service descriptions to formulate offers and needs in form of instances of these concepts. In addition to the requirements of a user, providers can define environmental requirements in their service descriptions. So it is possible to demand from the service user a minimal color depth of his display or a minimal bandwidth for using a service. These requirements can also be of both categories - context and quality of service.

Based on the nature whether the context is measureable or non-measurable, the data type of the attributes may vary. If the attribute is a measurable one it will have numeric type, if the property is a non-measurable one it will be of type string or boolean. For example the property color in the case of printer may have yes/no i.e Boolean value, where the mechanism should be one among the following laser, dot-matrix, ink-jet. This thesis provides facilitation to specify the type of the data this property will have. In the proposed work every non functional property has to be specified their values and type as shown in figure 6.8.

Non function properties specification

The request will be formed by the requester by using the same schema which is used to define by the provider. Each request will specify its demand by providing set of non functional properties. The application designer will assign values for the attributes based on the need of the application. Application profile is a collection of these non functional properties. The requesters current context can be accessed transparently form the user and device profile Pascal Bruegger (2010). The values on the request will be matched against the services’ values and ranking will be done in order to select most relevant service.

The request should have (i) Service description where the client specifies the attributes of the service looking for and (ii) Preference attributes for both context and service Quality of Service.

```
<attribute name="color">
  <type> Any one type among this xs:type="xsd:interger|xsd:string|xsd:Boolean"…</type>
  <value>…corresponding value of the specified type.... </value>
  <weight>..</weight>  </attribute> ....
```
In the service type it is assumed that the same schema will be used by the client to specify their requirements. The requirement for a service may vary from application to application. For example if the application needs a financial service then the security is having highest importance where as if the application needs a printing service for which low cost is preferable than security. Similarly it may depends on the user also, i.e one user may want good quality but she/he may not bother about cost, some user may need less cost as highest preferable one than the quality. So it is needed to allow the application to specify its own requirements. In order to facilitate this, assign every operation of a service to a category, for example printing service. Each category will have a set of properties assigned to it. Each property is defined as a set of four values.

\[\{<\text{attribute\_name}>, <\text{type}>, <\text{weight}>, <\text{value}>\}\]

Each property will have its own impact factor if available to an application. For example at some time quality will be more important than cost, at some time the otherwise. This thesis provides facility to assign weight to each of the property from the following weighing table. Weighing table used in this thesis gives the user the freedom of specifying one among ten preferences.

The weighing table may be changed if needed. This will not affect the process of selection as for as the condition in equation 6.3 is satisfied.

\[\sum_{i=1}^{n} |w_i| = 1\]  

(6.3)
The preferences can be specified as weights. Based on the weighting value the properties are categorised as *Mandatory* and *Secondary* properties. The mandatory properties will have weight as 1, which is the maximum value for weight. It means the satisfaction of this property is essential, failing which the service cannot be utilized. So this is used to filter the services. The value can be either positive or negative. Positive value represents the higher value in this attribute is preferable and negative represents the lower value is preferable. For example the property performance should be high where as cost should be less.

The *Value* attribute may take any value based on the data type of the attribute. It is used to specify the value expected to be for this particular attribute. For example if the data type of the attribute is *Boolean*, then it can have either yes or no to be matched, e.g. the property color will have yes/no as its value; if yes it means color printing service is needed, Where as if the data type is numeric then the value should be in number.

The value of the boolean and string type are used to match with the value specified in the service’s properties exactly. But the value specified in numeric type is used somewhat differently. Consider this situation where less cost for the printing service is needed. If value 10 is specified it does not mean that printing service lesser than that price cannot be selected. If lesser than that is available then that service may be preferred, but the cost should not exceed the specified value. So, for numeric evaluation the proposed framework will find the services which provides lesser value if the weight is negative. This framework compares all the selected services and finds the position in which every service’s value belongs to among the availability. The value property of the numeric data type is used as filter, i.e if the weight is -0.6 or any numeric attribute it means that preference as 0.6 is assigned, but the value expected is lower (as the weight is negative) and if the value attribute is specified or example value is 20, then the framework filters all the service which are higher than this, because only the services which provides lesser or equal to the value specified are needed. This value will be omitted while calculating the effective factor.

As the schema for the attribute value specification has been defined the user can select the values one among the set of values. This can be done if the data type is specified as enumeration. If the number of matching element is high then the service’s metric is high, i.e it is
expected that the maximum number of entries to be matched. Different data types that can be permitted are limited by XML Schema. In the case of context attributes the address/ method how access these values should be specified in the schema.

**Evaluation of individual properties**

Searching for the relevant services to operationalize a particular process is one of the challenges of the service oriented computing. Matching the service description with the service request will be done by *Matcher*. The discovery component performs a matching algorithm that filters those service providers which are providing the functionality the client has requested for. The non-functional properties provide further details of the quality and context of service execution that are needed by the requester. The importance of the attributes for a service may vary for different clients. For example one client may prefer to have speedy reply where as other may be keen on the location of the service. Clients are provided with the freedom of specifying importance of an attribute by the application designer itself. Since different data type for the attributes are allowed the calculation varies slightly for each attribute.

Based on the non functional properties the discovery service will perform a ranking that sorts the services according to the degree of their compliance with the client’s non-functional requirements. Now the detailed discussion on evaluation of each data type follows.

*For Numeric type:* The data types integer, double and currency comes under this. In the case of data like bandwidth, price etc. though they are numeric data the lower values for them are expected. This case can be represented using negative in “weight” of the attribute.

In order to find the best provider, the metric of the provider is calculated by comparing with the maximum and minimum possible values that can be provided for this attribute among the service providers. If $S_{max}$ is the maximum value and $S_{min}$ is the minimum value for this attribute then the metric of a service provider is calculated using the formula (6.4) and (6.5)

*For a positive weight*
\[ M_i = 1 - \frac{S_{\text{max}} - \text{value}}{(S_{\text{max}} - S_{\text{min}})} \]  

(6.4)

For the negative weight

\[ M_i = \frac{S_{\text{max}} - \text{value}}{(S_{\text{max}} - S_{\text{min}})} \]  

(6.5)

For Boolean type of attribute the exact match on the value with the service’s value. If there is a match then metric = 1, else 0. And the metric is calculated as shown in formula 6.6.

\[ M_i = \begin{cases} 
1 & \text{for a match} \\
0 & \text{for otherwise} 
\end{cases} \]  

(6.6)

For the enumeration type data the string specified in the “value” will be matched with the service’s value and the metric is calculated as shown formula 6.7.

\[ M_i = \frac{(v_1 + v_2 + \cdots + v_n)}{n} \quad \text{where } v_i = \begin{cases} 
1 & \text{for a match} \\
0 & \text{for otherwise} 
\end{cases} \]  

(6.7)

For a match 1 is added and for mismatch 0 is added and the final score will be divided by the number of elements in the set. Suppose say service \( x \) provides \{ color printing, credit card and debit card payment and the price is 5 \}. The metric is calculated as follows:

For the first attribute “color” which is a boolean attribute, only an exact match is needed so the metric for the first attribute is 1.

For the second attribute “payment” the enum attribute, it is found that the service provides both the requirement of the requester so the value is 1 as explained above.

For the third attribute “price” which is a numeric attribute and the weight specifies that lower value for this attribute is needed If the maximum printing price is 20 per page and the
minimum is 2 per page then the price provided by provider satisfies the customer to 0.833 \((20-5)/(20-2)\). Color metric\((m) = 1\); payment metric\((m) = 1\) and price\((m) = 0.83\) Like this the metric for each attribute specified in the request against the service’s values will be calculated. To find the metric of the service by the provider the metric values will be multiplied by the weight specified assuming that the sum of weight will be equal to 1.

*Aggregating the individual evaluation*

The non functional attributes specified in the request as Mandatory and secondary based on their specified weights. Mandatory attributes are used to filter out the services if there is no exact match. The weight for these attributes will be assigned as 1. In the case of numeric type the weight can have 1 and the value attribute to some non-negative value to represent that the attribute must have value less than the value specified in the value attribute of the requirement. For example if the requirement is “the cost must be less than or equal to 50” then the services which are providing the service for more than 50 should be eliminated. Then the requirement can be as \{ Name=”cost” weight=1 value=50 type = numeric\}. The steps need for selection is shown in Figure 6.7.

**Step 1:** *Filtering* the services based on mandatory attributes.

*Case 1:* Numeric type property with weight is negative and value specified

Select the service if its value it provides is less than or equal to the value specified in the request.

Eg. \{ Name=”cost” weight= -.6 value=50 type = numeric \}. The services which provides less or equal to 50 only will be added to the list for further processing.

*Case 2:* Boolean type with weight 1 (value must be present)

Select the service if its value is exactly equal to the one specified in the request.

Eg. \{Name=”security” weight=1 value=”high” type=Boolean\}. The services which are providing high security will be added to the list for further processing.

*Case 3:* Enumeration type weight =1 (value must be specified)

Select the service if all the items in the requirement set matches to the availability, else 0.
Step 2: Evaluate on the weights assigned and order the services of n service providers.

\[ \sum_{i=1}^{n} W_i \times M_i \]  

Equation 6.8 is used to calculate the overall value for all the secondary properties and sorted based on the overall value. The Figure 6.9(a) and (b) elaborates the steps.

**Step 1**: Filtering the services based on Mandatory requirements. \( M \) is one of the three functions based on the data type.
**Step 2:** For every service in the SSL calculate the weighted sum. $S_i$ is the secondary requirement value we will apply the functions and find their metric that will be multiplied with the weight specified in the requirement.

(b)

**Figure 6.9 (a) Step1 (b) step2 in Matching Non functional Attributes**

**Matcher**

This module is responsible for performing a matching algorithm between user/device requirements and service capabilities, taking user preferences into account. The matcher is requested to perform its algorithm when the resource manager needs to determine the list of visible service for the user. The matcher receives user/device context created by metadata manager and from service data from the local available services repository. The static profile is used to perform direct matching between user and device requirements and service capabilities. For each capability required by the user the matcher verifies if the service profile contains one or more compatible capabilities. The same algorithm is re-applied to the output of the direct matching to perform inverse matching i.e. to match service requirements against user/device capabilities. Figure 6.10 shows the pseudo code for the matcher algorithm.

In order to determine the list of user’s visible services, the matcher performs evaluator on all locally available services. On the contrary, when processing a specific request, the matcher may either stop at the first occurrence of a compatible service or perform the algorithm on each visible service. In the first case it returns a reference to a single service, while in the latter case it returns a list of services ordered by preference matching degree.
The service matchmaking procedures of the technique are implemented in Java. In both these matchmaking procedures, first it is checked for the behavioral match, then for the functional match and then the non functional properties are matched. This thesis uses java’s DOM methods to retrieve and match. In case of functional parameter match and also the basic type matches. Non functional checking is simple and it is performed using the basic logical and numerical operators of java.

---

**vector match(r)**

*Initialise the array with 0 for all the services in the list*

*Retrive the list of mandatory properties in Request R*

**For each pi property in R do**

*Collect the values from each of the service provider for this property in the array L.*

*evalmandatory(l,r.pi) // Removes the non matching providers from the list*

*Retrive the list of secondary proeprties in Reques R*

**For each pi property in R do**

*Collect the values and weigh from each of the service provider for this property in the array L.*

*evalsecondary(m,l,r.pi) // produces the metics in the array M*

*Call the aggregate function to sort the service providers.*

*return the sorted list.*

---

**Figure 6.10 Pseudo Code for Matcher Algorithm**

Type-based criteria evaluation function: this component is designated to dynamically evaluate the different criteria. The inputs of the evaluation function are the user request and the list of functionally competitive services in the form of object with their metadata. The outputs are
evaluation scores for different services. The function has two features. First it automatically matches the evaluation methods based on the abstract types of the properties. Second it can query all services related metadata and evaluate them according to the requirements.

Automatically evaluating different criteria is one of the challenges in a service selection method. Most of the traditional evaluation functions are tightly bound to a predefined evaluation function. Using static matching function when a new property is added into the evaluation environment a new evaluation function or mapping needs to be predefined as well. The proposed type-based evaluation function which automatically matches the evaluation methods to the property by detecting their abstract types rather than having identify an evaluation function for each property.

### 6.6 Implementation of Non Functional Properties Management in CASP

The functionalities described in the previous section has been implemented and deployed. Using the overall system we have performed experiments to validate the efficiency, but also more importantly to validate in terms of providing the desired functionality. As specified in the previous chapter, the implementation for the above said functionalities were developed using JAVA (J2ME). Since the code is written in JAVA, the easiest way would have been to serialize Java objects and send them over sockets, but this option does not provide for language independence. The next option considered was to use a lightweight parse to convert java objects to XML and transport them over TCP. Unfortunately, the compatibility issues between J2ME and the parser made it difficult to implement. So, it has been decided to use SOAP. SOAP offers language independence and SOAP clients are available for many popular languages. Additionally, it provides a clean transport mechanism, HTTP over which the client and services communicate. kSOAP J2ME library has been used which provides a subset of SOAP 1.1 and has memory foot print less than 50 KB. Further the kXML parse provides fast performance.

In order to manage the context and non functional properties the proposed framework defines packages and classes with required methods. The Metadata package has the classes and functions to parse and create the required context classes such as user context and application profiles. `parseuserprofile()`, `parseapplicationprofile()` are the important methods.
The following are the functions defined in various modules that are needed for non functional properties management. The proposed framework uses cross-layer interaction between the network, MAC, physical layer and application layers. These functions can access information from application layer module and the modules of network, MAC layer and the physical layers. The following are the functions defined.

In order to collect the internal data dynamically the sensor class is used. It has the important functions `cur_bandwidth()`, `nodespeed()`. These functions invoke the adaptor methods defined. The adaptor `getroutes(int node)` is a function to access information from Network layer. This function is used by the binder module which is described in chapter 7 to get all the routes to the specified node. The adaptors `getrxvpower()`, `getcrnt_tx_pwr()` are functions to access information from MAC and the `get_crnt_pwr()` is a functions to access information from to the physical layer. In order to find the remaining batter power the function `rembatterpwr()` uses these adaptors. The functions `cur_bandwidth()`, `nodespeed()`, `remmemory()` and `rembaterypwr()` may be used by the advertisement creation of the service provider.

Matching algorithm needs to access the values defined in the request and service providers. This can be done by the use of the functions such as `getattributes`, `getmandatory`, `getsecondary`, `getprotocol()`. This class also has functions `getmandatory()`, `getsecondary()`, `evalmandatory()`, `evalsecondary()`, `eval()`. `eval` is an overloaded function to evaluate individual properties.

### 6.7 Evaluation of Non Functional Properties Management in CASP

This section is dedicated to discuss the method of evaluation of the proposed non functional property model with the focus on two major aspects: (i) adequacy and (ii) scalability.
Adequacy aspect is demonstrated with a feasibility analysis. While scalability is shown using simulated values.

(i) Adequacy

It is desirable to show the adequacy of the proposed selection method. Adequacy means the most suitable service with respect to the user. However most suitable is difficult to define in general because it depends on different views and concerns. In terms of context-aware service selection, the suitability is composed of two important factors: Suitability with respect to user context, Suitability with respect to non functional property preferences.

Suitability with respect to user context is the basic level of adequacy, because the user should have the ability and convenience to use the selected service. Therefore, the selected service should definitely match the user’s context information.

Suitability with respect to non functional property preferences is the satisfaction level adequacy. The service is a better choice if it reaches the basic level and also satisfies the service selection preferences. Normally, different non functional properties have different important concerns in different situation. Therefore, the most suitable service requires reflecting the non functional property preferences.

This section demonstrates the evaluation of adequacy using 5 services and a request pattern. This demonstration specifies the potential impact of the proposed framework for supporting user preference in service selection with weightage assigned for each preference. It aims to illustrate how considering all the preference results in a major differences in the selection of the service providers. There are five service providers with 10 properties. Three properties with preferences were considered. All the information related to the service providers has been sent to the nodes through advertisements. Nodes will utilise them during matching.

The request is assigned with some preference value and the result shows that the specification of weight affects the selection of the services. Test cases are designed on imaginary services with values assigned to them. The services non functional properties Meta data is presented in table 6.2.
Let us show the evaluation results by considering the service request:

\{\text{Name}="\text{No. in Queue}\" \text{weight}=0.5 \text{ value}=\text{NO} \text{ type}=\text{Boolean}\}\n\{\text{Name}="\text{Location}\" \text{weight}=0.2 \text{ value}="\text{X,Y,Z}\" \text{type}=\text{Enum}\}\n\{\text{Name}="\text{Performance}\" \text{weight}=0.3 \text{ type}=\text{Numeric}\}\n
The evaluation as discussed in the previous section, are shown in Table 6.3. The maximum value among all the services for \textit{performance} is 0.9 (i.e.) $S_{\text{max}} = 0.9$. The minimum value among all the services for \textit{performance} is 0.5 (i.e.) $S_{\text{min}} = 0.5$. The metric $M$ for \textit{performance} of $S_1$ is $\{ 1 - (0.9 - 0.8) / (0.9 - 0.5) \} = 0.75$. Similarly the metric for all the service providers will be calculated.

The metric for \textit{location} is calculated by adding the number of matches divided by total number of match requires. In the above example the \textit{location} to be matched are x, y, z and thus the number of matches to be made is equal to 3. Since Service provider $S_1$ has location x alone the metric for location property of service provider $S_1$ is $(1/3) = 0.33$. Similarly the metric for all the Service providers will be calculated.

The metric for \textit{no. in queue} is calculated as either 1 if there is a match else 0. In the case of service provider $S_1$ since the \textit{no. in queue} property has value “NO” the metric for \textit{no. in queue} property of service provider $S_1$ is 1. The calculation for the rest of the providers goes like this.

<table>
<thead>
<tr>
<th>#</th>
<th>Color</th>
<th>Price</th>
<th>Available</th>
<th>Location</th>
<th>Performance</th>
<th>Reliability</th>
<th>Security</th>
<th>resolution</th>
<th>No. in queue</th>
<th>File type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>yes</td>
<td>3%</td>
<td>yes</td>
<td>X</td>
<td>0.8</td>
<td>0.6</td>
<td>high</td>
<td>high</td>
<td>No</td>
<td>pdf, doc, bmp</td>
</tr>
<tr>
<td>$S_2$</td>
<td>no</td>
<td>2%</td>
<td>yes</td>
<td>X, Y</td>
<td>0.5</td>
<td>0.5</td>
<td>high</td>
<td>high</td>
<td>No</td>
<td>pdf, doc, bmp</td>
</tr>
<tr>
<td>$S_3$</td>
<td>yes</td>
<td>5%</td>
<td>no</td>
<td>A</td>
<td>0.9</td>
<td>0.8</td>
<td>medium</td>
<td>medium</td>
<td>yes</td>
<td>doc, bmp</td>
</tr>
<tr>
<td>$S_4$</td>
<td>no</td>
<td>2%</td>
<td>no</td>
<td>X, Y, Z</td>
<td>0.9</td>
<td>0.9</td>
<td>high</td>
<td>high</td>
<td>no</td>
<td>pdf, doc, bmp</td>
</tr>
<tr>
<td>$S_5$</td>
<td>yes</td>
<td>4%</td>
<td>yes</td>
<td>B, Z</td>
<td>0.5</td>
<td>0.6</td>
<td>high</td>
<td>low</td>
<td>yes</td>
<td>doc, bmp</td>
</tr>
</tbody>
</table>

Table 6.2 Sample Service Descriptions
<table>
<thead>
<tr>
<th>#</th>
<th>Performance</th>
<th>Location</th>
<th>No. in Queue</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Weight</td>
<td>M</td>
<td>W*M</td>
<td>Weight</td>
</tr>
<tr>
<td>S₁</td>
<td>0.3</td>
<td>0.75</td>
<td>0.225</td>
<td>0.2</td>
</tr>
<tr>
<td>S₂</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>S₃</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>S₄</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>S₅</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 6.3 Evaluation Values**

From table 6.3 it can be seen that when it considers only performance S₃ and S₄ are having equal values which is shown in figure 6.11, whereas considering location alone S₄ is at the top and S₂ comes next, which is shown in figure 6.12. When considering all the requirements with varying preferences weights the value will be different which is depicted in Figure 6.13. From figure 6.13 it can be seen that S₄ is the service on the top then, S₁, S₂, S₅ and finally S₄. So, if the request with the preferences for all the required properties is specified, it can select the appropriate service for the requesters’ need.
Figure 6.11 Evaluation Chart by Considering Only the Performance

Figure 6.12 Evaluation Chart by Considering Only the Location
By analyzing the different service selection cases, the proposed method always provided the expected selection results and hence is adequate with respect to the adequacy definition.

(ii) Scalability

Scalability is an important issue for service selection methods. In order to accurately discuss the scalability, two test cases were considered.

(i) Increasing the number of services with fixed number of criteria
(ii) Increasing the number of criteria with fixed number of services,

Test case 1: Increasing number of services

The test case focuses on measuring the selection time with regard to increasing number of services. Different increasing numbers of services with 6 criteria were selected.
Processing time requirements for these services were plotted in figure 6.14. Linear increase in runtime, hints that the proposed method provides good scalability with regard to the number of services with a fixed number of criteria.

Test case 2: Increasing number of criteria

The service discovery time of a single service while varying the contexts from 0,5,10 specified in the request. The discovery time increases between 1.0ms to 1.5ms for each added context. The service time includes all the accessing time and evaluation time of all the context and processing time between all communicating modules.

Figure 6.15 shows the processing time required when we increase the context attributes. The actual time increase is due to the context matching with the service advertisements. As the framework separates the context as mandatory and secondary the list produced based on the mandatory reduces the time for processing the rest of the contexts. If it is not separated the contexts as mandatory and secondary the time taken is increased 0.2ms for every context. If the mandatory context does not match the framework immediately remove the service from the list for further processing saves 0.2ms for every context.
Thus, even a very limited amount of context information itself will give better selection of the providers and thus the performance of the discovery. So, for the marginal increase of the processing time, a gain in better performance can be attained. By analysing test results it can be seen that the proposed method has a good scalability against large number of criteria.

Comparison with other protocols

For comparison with other protocols two major aspects have been taken one is the overhead induced because of non functional properties introduction and the other is recall and precision factors.

Overhead

Two services are considered in order to show the overhead introduced by the framework. Though the key-value pairs for service attributes consume more compact messages than those produced by XML descriptions, it can be argued that the invocation and calculation time takes very less compare to the key-value pair. Two services called simple an imaginary service with zero attributes and a second service ‘light path’ which used as an example by Limam et al (2007) were taken for consideration. By using one service without attributes and another with multiple
ones it is aimed to demonstrate that the difference in message sizes diverges with an increasing number of attributes. Figure 6.16 shows the advertisement size for various protocols.

![Figure 6.16 Advertisement Size for Various Protocols](image)

There is no mentioning of time requirement for the evaluation of the properties. There is no possibility of specifying user preferences with weight values in the protocols. Only exact match can be made. So, the comparative study on those protocols was omitted.

*Quality of the matching algorithm*

Both the quality of matching algorithm and overhead introduced by adaptation of the metadata, non functional property modelling and techniques were evaluated. A variable search space of 30-50 profiled services with capabilities/requirements was considered. Each service has either one or two abilities. Each user’s request has a variable number of preferences from 1 to 4 on a single capability.

To evaluate the quality of the proposed matching algorithm the recall, i.e. the extent to which all relevant registered services are retrieved and precision i.e. the extent to which only relevant services are retrieved were measured. The formula to calculate recall in given in (6.9) and formula to find precision is given in (6.10)
\[
\text{recall} = \frac{\text{number of relevant items retrieved}}{\text{total number of relevant items}} \quad (6.9)
\]

\[
\text{precision} = \frac{\text{number of relevant items retrieved}}{\text{total number of retrieved items}} \quad (6.10)
\]

Being complete with respect to the proposed service schema, the matching algorithm’s recall is optimal. It means that the proposed prototype is able to find all services whose capabilities have a match with requested capabilities according to the schema designed. To calculate precision, the case of services with two capabilities and user requests with two preferences has been considered and compared the number of services retrieved by the proposed framework and by ADDER—which representing service capabilities in plain text attributes. The proposed framework has demonstrated an improvement in precision of roughly about 55\% over ADDER in the considered test search. Figure 6.17 depicts this result. This outperformance is due to the adoption of a service schema that separately defines service capabilities and requirements. While ADDER only allows defining and looking for service attributes, the proposed framework allows defining either capabilities or requirements with different meanings. Users requirements and preferences are only, matched against service capabilities.
Non functional properties were categorised into requirement and capabilities. This thesis provides facilities to specify user preferences on non functional properties in the form of weights ranging from 0.1 to 1. Based on their importance they are divided into mandatory and secondary where mandatory needs exact match and secondary properties needs approximate match. This categorisation ensures only relevant services will be selected based on users’ current needs. A detailed description of how to (i) specify meta data (ii) access the meta data and context (iii) matching the user requirements and the service providers’ offer has been provided in this chapter. Influence of non functional properties in service selection has been shown. Proposed work has been compared with the other protocol. Proposed framework has demonstrated an improvement in precision of about 55% over ADDER in the considered test search. This outperformance is due the adoption of a service schema that separately defines service capabilities and requirements. It also shows that the recall ration of CASP is above 70% where konark achieves 55% and ADDER achieves 20%. It has been shown that the proposed scheme scales well.