Proc. IEEE 6th Int. Enterprise Distributed Object Computing Conference (EDOC 2002), 17-20 Sep. 2002, Lausanne, Switzerland

Constraint based Network Adaptation for Ubiquitous Applications

Krish. T. Krishnakumar, Morris Sloman

Department of Computing,
Imperial College,
London, UK
tkkumar@doc.ic.ac.uk, m.sloman@doc.ic.ac.uk

Abstract

In this paper, we present a constraint-based approach for selecting and deploying active network components, called proxylets, which execute on servers within the network to perform application-aware adaptation for ubiquitous computing. We present a scenario showing how to provide self-adaptive deployment of components to give mobile users ubiquitous access to a service irrespective of the user's end device and location across multiple network service providers. We concentrate primarily on solving the problem of how to automatically configure these proxylets in the network without violating the constraints set for the components and network elements involved. This system permits access to ubiquitous applications performing application specific transformations and adaptation on behalf of the user, as well as to seamlessly adapt to changes due to failures or traffic conditions in the network.

1. Introduction

Ubiquitous computing environments will involve a variety of devices with different capabilities in terms of processing power, screen display, input facilities and network connectivity which can vary between wired and different types of wireless links. It will also permit mobile users to access services from anywhere - in the home, at work, while traveling. Ideally applications should be able to work transparently with powerful desktop computers over wired links as well as more limited portable devices. The objective is to provide users with seamless, ubiquitous access to services irrespective of the user's end device or location.

Current Internet applications and web services often perform this adaptation by generating content specific to particular devices and storing this in web servers. However this does not scale to the tremendous diversity in the types of end devices and applications, which will be available in the future. Dynamic adaptation of information streams is needed based on the user's context in terms current location and device capabilities. The adaptation must also cater for dynamic changes in link capacity due to failures, movement when using wireless connectivity and switching from wired to wireless connectivity during a session.

Context-aware adaptation is needed within the network to cater for this diversity of device and connection capabilities. A programmable or active network will support application specific components to perform this adaptation. However, there may be charges for using the processing and communication resources within the network. Deciding what adaptations to perform, and where the components should be run in order to maximize performance and minimize costs can be a complex constraint satisfaction problem.

The objective of our work is to minimize the degree of explicit configuration management necessary for setting up or using a service or to cater for changes in network conditions. Our aim is to develop methods to solve protocol or transcoder component deployment problems in a dynamically changing active network environment, requiring multiple constraint satisfaction. Our approach distributes the constraint satisfaction problem such that a subset of constraints related to a specific user or application are evaluated 'near' the user in a local network node.

This work has been done within the Alpine project [2] and uses Application Level Active Networking (ALAN) [3] in which active components called proxylets execute within network nodes. The approach builds on our initial work on end-to-end constraint based self-configuration described in [6].

The rest of this paper is organized as follows. Section 2 presents motivation behind our work and a hypothetical scenario. In Section 3 of the paper, we describe the concept of Application Level Active Network (ALAN) proxylet approach being used in the Alpine project and we formally define constraint based satisfaction in terms of configuration spaces, and introduce constraint variables and values for ALAN application scenarios. The derivation of proxylets configuration setup and their constraint checks for an application are presented in Section 4 with the customers' service requirements. In section 5, cost calculation algorithm and cost constraint adaptation are explained. The architecture of our approach and the implementation are described in Section

6. In Section 7, we provide a discussion of related works in network and client adaptations and network management in active networking. Finally, we conclude by reiterating the main contributions of our paper.

2. A Scenario

An ubiquitous computing environment will enable users to access multimedia Internet applications using small handheld devices with extremely limited capabilities. It is likely that the intelligence required for configuration, processing, storage or internet access, would be provided in adjacent network nodes without explicit user intervention. The environment will have to support interaction between users connected to different networks. The ability to freely pull up information and move it to different displays on devices in other networks is one of key useful features for ad-hoc teleconferencing. For example, during a meeting, the Chair can open any web applications on a browser and initiate migration or broadcasting of the linked pages to the browsers of other participants in the meeting, across the world. This type of migration requires automatic setting up of support components within multiple networks without violating the distributed constraints of network service providers or users.

To illustrate the kind of applications we are trying to create, we briefly describe a hypothetical scenario below.

> Bob is playing a multi-player, multimedia video game from New Zealand. He wishes his friends, Pete in Canada and Sam in South Africa, to join the game. Bob sees Pete and Sam in his "on-line buddy list" as active since all three are connected online at the moment. Both Pete and Bob's browsers are the same type (wired, wall-mounted monitors) but Sam's browser is a handheld wireless device. These three devices are not rich in resources such as computation power, memory etc., so they depend on network elements to perform computations on the incoming flow.
>
> All three users connect to different network service providers. With Pete and Sam's permissions, Bob tries to set up the network configuration for the game to be directed to their browsers. Bob uses a pull-down menu on his browser to instruct the network to configure proxylets suiting Pete and Sam's browser types and user profiles. Pete's browser is the same type as Bob's, so the required proxylet configuration is similar to that for Bob. However, Sam will get a warning message on his browser stating that he has been invited for this game but with degraded application quality on his browser. This is due to the fact, Sam's browser is wireless handheld device, which is not able to receive special movie pictures of the game characters. Instead, he is to get level 2 of the quality for moving images. Sam agrees to this. In the middle of the game, Pete decides to go out-doors while continuing to play. He needs to transfer the game to his handheld device from the wall-mounted screen. Pete uses "migrate" button on the display's touch screen to transfer the game to the handheld device. Shortly, he gets the game transferred to the handheld device and continues to play seamlessly without loosing the status of the game but the quality of the moving images of the game characters have been degraded.

3. Background

3.1. Application Layer Active Networks

Active Networks [1] pave the way for transforming the current passive network infrastructure into one where new services and protocols can be adopted without the need for standardization. The approach of deploying the protocol elements at the network router level is more complicated. The network provider will not permit the deployment of protocol code from third parties at this level. An Application Layer Active Networking (ALAN) [2], [3] system has been implemented to provide the user with a flexible framework for supporting active services within traditional network boundaries.

The ALAN system (Figure 1) assumes clients access remote servers across the Internet using the HTTP protocol. Protocol entities called *Proxylets* can be dynamically loaded onto intermediate network server nodes (not routers). These nodes provide an application-layer *Execution Environment for Proxylets (EEP)* within the network.

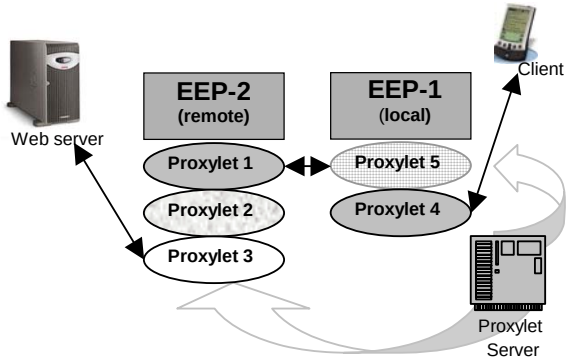

Figure 1. Application Level Active Networking architecture

Suitable EEPs must be selected within the end-to-end path between client and end-server, e.g., near client or near server. We are also using proxylets to perform the configuration management tasks such as selection of specific proxylets to provide the required service and allocation optimally located EEP servers. The reason for using proxylets for managing the configuration is that a new application may be instigated by a mobile user with very limited processing resources. Our architecture also allows for an EEP server to be based on a more powerful mobile device such as a laptop computer.

3.2. Constraint Specification

A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. For example X+Y > 5 defines a constraint on permitted values of X and Y and (P ∨ Q) ∧ (¬P ∨ ¬S) is a constraint on the permitted values of the Booleans P, Q and S. The specification of a configuration in terms of assembling the parts into a required system, involve two distinct phases:

i) Domain knowledge – describes all the objects, which potentially can be used for an application and the relationships among them.

ii) Specification of the desired configuration – requirements that must be satisfied by the structure or topology of the configuration.

A Constraint Satisfaction Problem (CSP) is defined as $\boldsymbol{P} = <\boldsymbol{V}, \boldsymbol{D}_v, C_v>$, where

i) A set of variables V representing all the variables that may potentially become active and appear in a configuration. $\boldsymbol{V} = \{\boldsymbol{V}_1,....,\boldsymbol{V}_n\}$.

ii) $\boldsymbol{D}_v$ is the set of domains, with $\boldsymbol{val}_i$ representing the set of all possible values for variable $\boldsymbol{V}_i$

iii) A set of constraints to restrict the value assignment of some variables or configure the components according to the component's behavioral model.

Figure 2 shows some variables, their related values and constraints for the above application scenario as a Constraint Satisfaction Problem. These constraints are specified to automate the configuration process. For example, to satisfy the constraints in using a hand-phone for an email application (see Figure 3) our system will find the required proxylets to be *text2speach* and *WAP*.

Domains (D_v):

EEP-node : local, remote, intermediate

Proxylet : wap, comp, decomp, deployment, config, rtpRx, rtpTx, realPlayer, handoff, etc…

Browser : PC (wired), PDA (wireless), handphone (hp), …

Payload : text, audio, video

Application: email, radio, movie, TVnews

QoSPack : Gold (G), Silver (S), Bronze (E)

Mode : realTime, download

…….

Variables (V):

n : EEP-node
pl : Proxylet
b : Browser
p : Payload
q : QoSPack
m : Mode
a : Application
c : Cache
…….
…….

Constraint (C_v):

b, p : (PC = text), (PC = audio), …, (PDA = video), (hp = text), (hp = audio), (hp ≠ video)…
n, pl : (local = wap), (local = decomp), (remote ≠ comp), (local = comp), …. (inter = comp), …
b, q : (PC = G), (PC = S), (PC = E), (PDA = G), (PDA = S), (handphone = G), (PDA ≠ E)….
p, q : (text = G), (text = S), (text = E), (audio = G), (audio = S), (video = G), (video = S), ….
b, a : (PC = radio), (PC = movie), (PDA = email), (PDA = radio), (hp = radio), (hp ≠ video), ….
a, p : (email = test), (news = text), (email = audio), (radio = audio), (email ≠ video), ….
pl, q : (wap = G), (wap = S), (wap ≠ E), (comp = G), (h/off = G), (voice2txt = G), (voice2txt ≠ E), ….
……..
……..

Figure 2. Domain variables, values and constraints

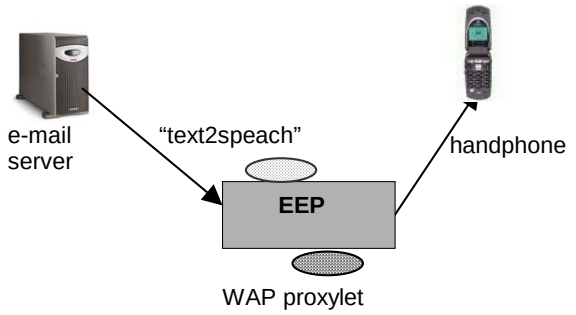

Figure 3. Proxylets configuration for accessing e-mail on a hand-phone

4. Application Configuration

Application components may need to operate in a busy network environment facing possible changes in network conditions. Consequently, applications are likely to undergo frequent reconfiguration: to upgrade or enhance existing components; in response to changing environmental conditions such as wireless capacity variations or as a result of user actions such as switching from wired to wireless connection during a session. Reconfigurations must be performed while the application is running, with minimal disruption to its operation. Specifically, the temporal integrity of active continuous-media streams must be maintained across reconfigurations wherever possible, without violating related constraints in the Programmable Network environment.

Reconfiguration is triggered by events generated by the EEPs, which have proxylets monitoring the application and resources such as communication links. The reconfiguration mechanism will perform following procedures:

i) Selecting alternate configuration parameters containing set of proxylets and their corresponding execution nodes (EEPs). These alternate configurations are usually prepared and stored by the system for each application download request.
ii) Making sure that the cost constraints are met for the new configuration.
iii) Deploying the new plan and resuming the transmission.

In the scenario shown in Figure 4, network-A and network-B are two different ISPs providing ALAN services to customers. Bob is using browser-1 to access a multimedia service in Network A and wishes Sam in South Africa in network-B, to view the same video application. He invokes the relevant operation on

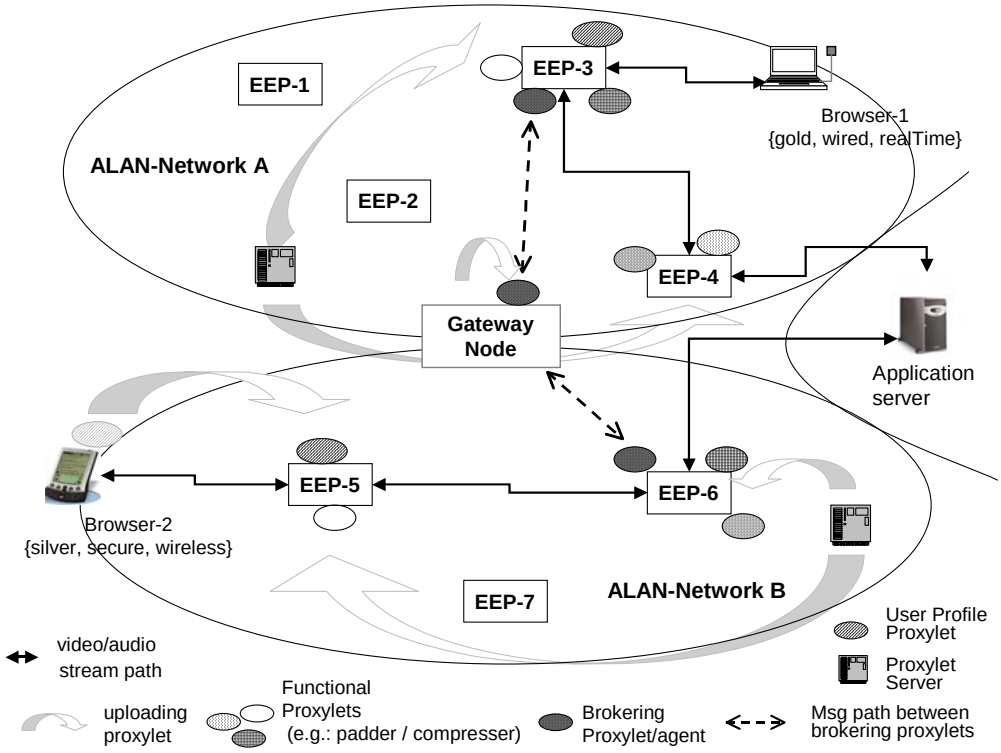

Figure 4. Auto-remote configuration

browser-1 to inform its brokeringProxylet on EEP-3 to communicate to, its counter-part at the Gateway EEP between networks A and B. The details about the required constraint based configuration of proxylets for this application and browser-2's IP address is passed along during this communication. The Brokering proxylet at the Gateway will identify the suitable local and remote EEPs or the browser-2 and check to see that the configuration setup (<*proxylets, EEPs*>), is obtained from browser-1, does not violate any local constraints for browser-2. If no constraints are violated, then the same configuration is applied and the video application is started on browser-2. If constraints are violated, then a new configuration is calculated for browser-2 by an instance of configProxylet executed on an EEP, which is local to the browser-2 and applied in Network-B. A small-specialized proxylet (user profile proxylet) carries information about the user and browser. This information is then used to dynamically generate or to select one from a previously prepared set of configurations. These take the form of a set of proxylets that are already optimized for execution on the target EEPs. This scenario requires more preparation on the home/local EEP, but has the advantage that the downloaded proxylet consumes fewer resources. This is probably advisable when targeting a small set of very heterogeneous environments.

There can be multiple instances of the Brokering proxylet for different applications executing at the Gateway EEP to serve as intermediaries between two neighbouring networks to enable negotiation for access to each other's resources. The Brokering proxylet also performs security checks on a user who is trying trigger remote-configuration for another browser.

4.1. Selection of Components

Assume that browser-1 and browser-2 in our example scenario have the following characteristics:

Properties	Browser-1 (wired)	Browser-2 (wireless)
realTime	X	
securely encrypted		X
reliable	X	

Figure 5. Properties of browsers

Browser-1 requires reliable transmission of media streams in realTime. Browser-2 requires securely encrypted streams, but it may get disconnected during the access (unreliable). Browser-1 is already accessing the video streaming application in network-A, so it has a defined proxylet configuration specification as follows:

```
Configuration_for_Browser-1
  (Package=>Gold, payload=>video) = {
     wiredBrowser, realtimeListening,
   EEP-3(local) => (decompression-
     Proxylet, [1]media-player-proxylet,
     rtp-Proxylet),
   EEP-4(remote) => (compression-
     Proxylet)
}
```

The system receives a configuration setup request for a member of the goldClass, who requires video application stream in real Time onto a wired browser.

The system identifies that the network requiring two nodes, namely EEP-3, which is local to the browser, and EEP-4, which is optimally located close to the server. A set of proxylets to fulfill the supplementary functions on behalf of the user is identified.

A proxylet is downloaded onto EEP-4 to compress the video stream before sending it towards EEP-3. This would both speed up the download process and save bandwidth of the link between EEP-3 and EEP-4. At EEP-3, on arrival the data needs to be decompressed and it is achieved by downloading a proxylet with the capability of decompressing the compressed stream. A realTime protocol proxylet is also loaded onto EEP-3 to enable the real-time streaming at the browser. Finally a media player proxylet is executed at EEP-3 to enable the browser to view the video. This completes the network configuration for a given request for browser-1.

The browser-1 in network-A attempts to setup browser-2 in network-B to receive the same media stream it is currently receiving, so it passes the above configuration specification to the brokering proxylet at the Gateway. However browser-1 has no knowledge of any of the constraints for browser-2 in network-B. The Brokering proxylet on the gateway checks for any constraint violation between this configuration and Network-B's local constraint variables for browser-2. In our case, the browser-2 has a wireless connection and so it is not capable of receiving the stream in real time. The initial configuration specification above, provided by network-A, violates some of the type compatibility constraints. In order to identify a set of proxylets satisfying the constraints local to network-B, the Gateway EEP identifies the following configuration using the constraint solution mechanism explained in [6]:

```
Configuration for Browser-2
```

[1] It has been assumed that users will be able to join in the ALAN network without the need to use pre-configured or resource rich mobile browsers (e.g. PDA) so a media player proxylet is loaded and executed on local EEP node in network for the browsers to view multimedia application.

```
   (Package=>silver,payload=>video) = {
      wirelessBrowser,securelyEncrypted
    EEP-6(local) => (decryption-
      Proxylet, decompression-Proxylet,
      [2]audio-player-proxylet, padder-
      Proxylet),
    EEP-5(remote) => (splitter-Proxylet,
      encryption-Proxylet, compression-
      Proxylet, WAP-Proxylet)
}
```

This configuration is for browser-2, whereby, a member of the silverClass, who requires video application stream onto a wireless browser. The system identifies that the network requiring two nodes: local node (EEP-6) and remote node (EEP-5), which is nearer to the server.

When the browser is wireless and the quality of service (QoS) for the user is Silver, the system identifies a constraint which is video payload is restrained for this combination, instead, system suggests for an alternative configuration by converting the video application to an audio-only mode and looks for new configuration. Since the user has subscribed to silver QoS-Package, the service provider introduces additional revenue making efforts such as attaching advertisements during download. In this case, the required resources to be configured are decryption, decompression, audio-player and padder proxylets to be downloaded on EEP-6 and splitter, encryption, compression, WAP proxylets at EEP-5.

Here, the splitter-proxylet splits a <video+audio> stream into an audio-only stream while the padder-Proxylet adds some advertising media frames to its input stream. The encryption and decryption proxylets cater for secure transaction.

5. Cost Constraints

An application flow may require a large number of proxylets executing in EEPs around the network to perform actions like compression, transcoding, caching on the flow itself. This requires resources from EEPs such as processing power and memory, as well as usage of communication bandwidth. It is likely that ISPs will have a charge related both to resource usage and a 'hire' charge for the proxylet code. ISPs may charge according to predicted usage pattern, e.g. low cost for off-peak usage of resources, or they may vary costs according to current actual usage. The end-to-end service can then be tailored to meet the cost requirements of the user.

The placement of proxylets on appropriate nodes now gets an additional constraint relating to cost of running an application. In our scenario, we define application quality of service classes – gold, silver or bronze. This classification has direct impact to our cost model, where the followings are set:

i) Any user is allowed to specify maximum price constraint on their request for an application prior to download action.

ii) Only gold and silver group users are entitled to alter their maximum cost during an application download giving them additional freedom.

iii) Users belonging to any group have different range maximum cost, i.e., gold member (the most privileged user) can set his max cost limitation to be higher than the other two groups enabling him to utilize most advanced or latest version of any functional/transcoding proxylets to access a service with better quality of service.

In our case, network-B allows users to set their limitation on their cost of using the services. We have defined a simple cost function for this scenario. We have predefined-charges for usage of EEP nodes, proxylets and the links. These factors are combined in a cost formula, which we incorporated in our constraint-based model.

Total charge, C_{total}, for an ALAN application is:

$$C_{total} = \left(\sum_{k=1}^{m} E_k + \sum_{i=1}^{n} P_i + \sum_{j=1}^{p} L_j\right) \ldots\ldots\ldots (1)$$

Where:

E_k = Charge for using k^{th} EEP to execute a proxylet; m is the number of EEP nodes used in a scenario.

P_i = Usage cost for i^{th} proxylet; n is the total number of proxylets used for an application. We assume the same cost for all proxylets (P_i).

L_j = j^{th} link usage unit price; p is the number of links used by the application. We have considered the charge on basis of per-hop for link usage. Alternatively, this could be modeled as to be max bandwidth required at each link. This would require constantly active (running) proxylets at each node collecting information about bit rate for each transmission. In this case, the transmission cost per unit of bandwidth at a link j can be defined as the following:

[2] As explained for [1].

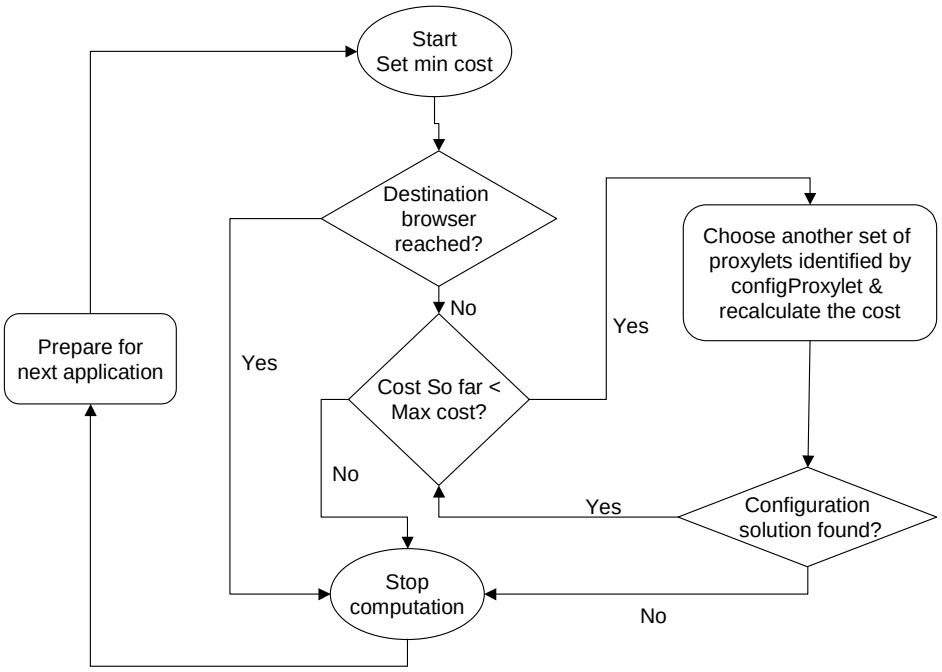

Figure 6. Flowchart of setting usage charge constraint for using ALAN resources.

$$L_j = B\$_j \times R_j \quad \ldots\ldots\ldots (2)$$

Where:

R_j = The rate at which stream flow on link j.

$B\$$ = The transmission charge per unit of bandwidth at link j.

Figure 6 shows the algorithm incorporating the cost function into our constraint-based configuration of proxylets. The users are allowed to specify the maximum cost constraint for a particular service requested. A set of functional and transcoding proxylets and their corresponding EEPs are identified satisfying the browser type, link and resource compatibility constraints. Then, this set of proxylets is now checked against our cost model for cost constraint. Using equation (1) and the algorithm, shown in Figure 6, each set is checked. When the suitable set is found for a required maximum cost, the system will deploy these proxylets.

6. Implementation Approach

6.1. Constraint verification using Alcoa / Alloy

We are using Jackson's Constraint Analyzer called "Alcoa / Alloy" [7] which can perform a semantic analysis of models that incorporate complex textual constraints. Using this tool, we have checked the consistency of our constraints, generate sample configurations, simulated execution of operations, and checked that operations preserve constraints. The Alloy language is used to specify constraints and the Alcoa tool for analysis of the constraints. We used Alcoa to interactively define the configuration scenario, incorporating constraints. However it is not possible to generate executable Java code from this. We use the Java Constraint Libraries to program our constraint based configuration of Proxylets similar to the Alcoa approach, as explained in Step 3 below. The main purpose of using Alloy for initial checking is to reduce the development cycle time by detecting errors prior to implementation. The actual implementation has to be done by other means such as readily available libraries for constraint resolution.

```
// This model is basically for the
// configuration of the EEPs and Proxylets
// to cater a given policy driven content
// delivery.

model ALAN-A {

 domain {browser, package, cache, proxylet,
  mode, payload, EEP}

 state {
   partition G, S, E: package
```

```
   disjoint PC, PDA, handPhone, … : browser
   disjoint realtime, download: mode
   partition text, audio, video: payload
   partition email, radio, ….: Application
   partition local, remote, inter : EEP
   partition  decompresplet,  compresplet,
  rtpplet, …….: proxylet
   belongs: cache ! -> local
   consistsof: package -> browser
   requires: browser -> local !
   runs: EEP -> proxylet+
   finds: locationplet -> remote
   uses: local  -> remote
   supports: browser -> payload
   comp : compresplet -> payload+
   decomp: decompresplet  -> payload+
   needs: mode -> proxylet
   ……..
   ……..
}
inv PDAE1 {
  PDA not in E.consistsof
}

inv PLPDA {
      video not in wireless.supports
}

inv EEPConst1 {
  all d: local | d.runs = locationplet
}

inv cacheConst {
    all c:cache | some d:local | c.belongs
 = d
}

inv proxyletConst {
  all  d:remote  |  d.runs  in  (proxylet  -
 locationplet)
}
……..
inv PDAconstraint1 {
  no b:PDA | b.supports = video
      video not in PDA.supports
  no c:cache | c.caches = video
}
……
inv EEPConst2 {
 all d:remote | local.uses = d
}
inv EEPProxyConst {
  all pr:proxylet | some d:EEP | d.runs=pr
}
inv VCConst {
  video not in cache.caches
}
inv const {
 rtpplet in realtime.needs
 rtpplet not in download.needs
 compresplet in download.needs
 locationplet not in mode.needs

}
cond PDAUser {
   some EEP1:local | some EEP2: remote | one
   b:PDA | (EEP1 != EEP2)
 }
   …….
   …….
}
```

Figure 7. Alloy specification for scenario constraints

We specify the scenario plus relevant constraints relating to the components in Alloy (Figure 7). Alcoa is a compiler, which translates the model definition into a very large Boolean formula. This formula is solved by the SAT solver in the tool and the solution is interpreted back into the Alloy language for the model representing the scenario. Then a paragraph or schema (e.g. a constraint) in the model is selected to be run for verification. Alcoa responds with an instance representing a solution satisfying constraints or with a message if no instance was found. If no solution is found, the scenario model can be edited, compiled and checked again. Using this tool, we are able to analyze the constraints more effectively prior to the Java implementation of our constraint based Proxylet configuration framework.

It was impossible to incorporate cost constraints in our ALLOY / ALCOA model since the cost constraint formation is infinite. We have implemented the cost constraint algorithm as a Java program and incorporated it into the framework.

The ALLOY / ALCOA tools can only be used to verify solutions related to predefined scenarios. It would not be practical for non-technical users to try to specify constraints to be solved. However many applications relating to accessing media streams, multi-user games can be analysed to produce a set of potential configurations which can be selected at run-time based on actual constraints identified from users or the environment.

Figure 7 shows the Alloy specification for scenarios. An invariant (denoted by key word **inv**) defines a constraint in the **model** being defined. For example, `inv PDAE1` indicates that economy QoSPackage group users cannot use PDA (wireless devices) for downloading any ALAN applications. Condition (**cond**) gives a hint to Alcoa on generating a sample architectural instance over a given scope.

6.2. Self-Configuration Architecture

Figure 8 shows the implementation architecture of the constraint based configuration system for an application using a given browser. This framework contains three information storage databases and the processes in the form of proxylets. Constraints, such as which servers, onto which specific proxylets should be loaded, are specified by user or service provider and stored in

ConstraintDB database. The proxylet repository (PIB) has the list of available functional proxylets in the network. *Cost Model CheckList* contains the cost break down of the proxylets, EEPs and links of the network.

The user or service provider may load the user's profileProxylet onto an EEP, which is local to the browser. The profileProxylet carries a set of constraint domain variables and values (QoSPackage, type of browser, download mode, user specific constraints etc) specific to a client. This set of data is passed onto the configProxylet to identify suitable configuration sets during an application request. The configProxylet perform the task of checking the constraints and finds out sets of possible *<proxylets, EEPs>* configuration parameters for the request. This version of the configProxylet is slightly different from our previous version developed in [6]. The current version has an interaction with a proxylet named accountProxylet, which ensures that the configuration solution meets the pre-set cost for the application. The accountProxylet implementation is explained in section 6.4.

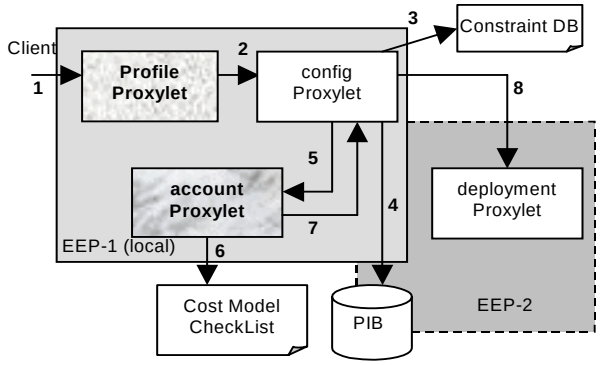

Figure 8. Proxylets interaction diagram for auto-configuration for an application

The first set of components, which the configProxylet selects, may violate the cost constraint. This violation could happen in run time as our system allows user to set the max cost at any time during an application download. If this happens, an instance of configProxylet is loaded again onto local EEP by the accountProxylet. This configProxylet instance repeatedly finds an alternative configuration setup until a satisfactory solution is achieved by both accountProxylet and configProxylet. Typically, at the end, a complete system required for the end-to-end HTTP stream transfer is achieved by meeting all the applicable constraints. The outcome is an XML configuration script (Figure 9), which can be interpreted by the deploymentProxylet to download the required proxylets onto specific EEP servers.

```
<?xml version="1.0"?>
<!DOCTYPE config SYSTEM
"public_html/DTD/PIB.dtd">
<config>
 <EEPurl>//146.169.14.9/~tkkumar/
                    remoteServer</EEPurl>
  <proxylet>
    <name>DecmpressProxylet</name>

<class>proxylet.decompressPxlt</class>
        <version>1.2.2</version>
        <server>book.doc.ic.ac.uk</server>
<url>//146.169.14.8/~tkkumar/server</url>
   </proxylet>
   ……
   ……
   <proxylet>
     <name>realPlayerProxylet</name>
     <class>proxyletaudioPlayerPxlt</class>
     <version>1.0</version>
 <server>book.doc.ic.ac.uk</server>

<url>//146.169.14.100/~tkumar/server</url>
    <init-param>
        <param-name>PolicyFile</param-name>
<!-- Policy File -->
           <param-value>
/tmp/policy.xml</param-value>
 </init-param>
        <init-param>

<param-name>ConstraintFile</param-name>
<!-- Constraint Solution File -->
           <param-
value>/tmp/myfile.txt</param-value>
        </init-param>
   </proxylet>
  ……
  ……
   <proxylet>
        <name>WAP_Proxylet</name>
        <class>proxylet.Pxlt</class>
        <version>1.2</version>
<server>book.doc.ic.ac.uk</server>

<url>//146.169.14.8/~tkkumar/server</url>
   </proxylet>
</config>
```

Figure 9. Proxylet configuration output file from configProxylet

6.3. Remote-Configuration Architecture

Figure 10 shows the implementation approach for the automated constraint-based network configuration for the remote browser in the scenario. This works in conjunction with the self-configuration framework shown in Figure 8.

GatewayEEP is operating as an intermediary node between two networks, which provide ALAN service. When a client is trying to invite another client in different

network to access an application, the brokeringProxylet is loaded onto the GatewayEEP. It accesses the remote user's profileProxylet (a) from which it can obtain any constraints relating to the user's environment. When the brokeringProxylet is instantiated at the GatewayEEP, the application specific configuration parameters <*proxylets, EEPs*>, used for the first client, are also provided. The brokeringProxylet loads a configProxylet to perform the following checks relating to the remote user and its environment:

i) Checks for availability of the required proxylets in this network (b).
ii) Checks for any violation of local variable constraints (c).
iii) Max cost constraint check is also performed (d, e).

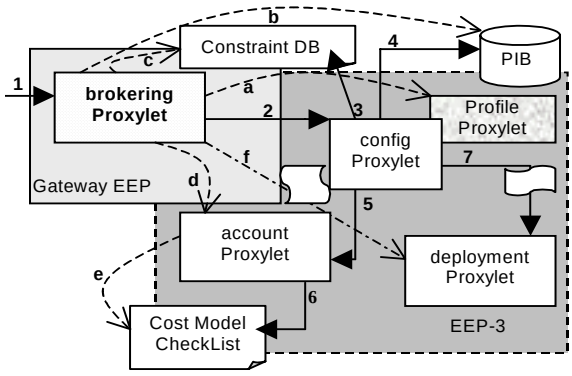

Figure 10. Proxylets interaction diagram for remote-browser configuration

If no constraint violations are detected, the deploymentProxylet is instantiated on a server (EEP-3), near to the remote browser (f) to deploy the same set of configuration parameters as used for first browser. If any of the above checks fails, the configProxylet is instantiated on EEP-3. It starts the process of finding a suitable configuration for the remote browser and the process is same as explained in section 6.2. The new set of configuration parameters will be another set of transcoding proxylets and corresponding execution nodes (EEPs) in this network. GatewayEEP will not take part in the configuration process if a client is trying to invite another client in same network to access an application. Then the brokeringProxylet is loaded onto local EEP to perform the configuration for the client.

This technique is useful, not only, for remote configuration but also for network adaptation when there is a change in network condition such as, link failure or reduced bandwidth during an application session. When there is a change in traffic conditions or link failures, the application data will be re-routed by choosing an alternative configuration.

6.4. Implementation of cost constraint model

We have implemented the cost calculation algorithms (shown in Figure 6) using Java as a proxylet, named accountProxylet.

This proxylet interacts with both bokeringProxylet and configProxylet. It queries the *cost model checklist* to access the budget of proxylets, EEP nodes and links. Cost calculation on a given set of configuration parameters (<*proxylets, EEPs*>) is done with the help of utility functions within accountProxylet and this proxylet is instantiated by either bokeringProxylet or configProxylet.

7. Related Work

A major trend in the Internet research community today is the push for ubiquitous access to rich multimedia content. This requires adaptation techniques within the network, which will transform the media into various formats better suited to the user's current context or to take into consideration network conditions. There are three main adaptation techniques namely Source Adaptation, Receiver Adaptation, Proxy Adaptation, which are being looked into by the researchers for this purpose.

Source Adaptation means that the adaptation mechanism is put at the application host server [8, 15]. It is not suitable for multicasting scenarios because of the potential diversity in end-user device capabilities. Multicasting essentially sends the same stream to each user but each stream should be adapted to the best possible quality suitable to the limitations of the displaying device. It is difficult for Application servers to know about the adaptations required for all potential devices.

With Receiver Adaptation [9], the source (application server) transmits the same quality content to all receivers with adaptation taking place at the receiver device. This implies that the receiving device has the processing capabilities to perform the required adaptation, which may not be the case for low-powered portable devices. Low-bandwidth radio links to devices require adaptation before the final link. Receiver adaptation may also result in inefficient use of network bandwidth as high-bandwidth streams are sent on the paths to all users, irrespective of their final link bandwidth.

Recently, the researchers have recognized the flexibility that a proxy can provide to the implementation and consumption of new services. The Proxy Adaptation strategy provides a compromise between source and

receiver adaptation and avoids the need for modifying either servers or device browsers. By placing compression or filtering proxies close to the source or where streams split for multicasting, adaptation can be used to reduce bandwidth usage. Adaptation within the network can also be used to tackle network problems. Several research groups [10, 11, 14] have addressed network proxy based adaptation techniques, using only a single proxy rather than multiple ones. Our procedure not only allows multiple proxy usages, but also, enables automated selection and deployment at optimal locations in an end-to-end transmission path.

Proxy based adaptation may use substantial resources within the network nodes so accounting mechanisms should be incorporated in the proxy solution in order to keep track of the amount of resources utilized and it may be possible reduce costs by suitable proxylets selection and selecting locations for deployment

The programmable network has been identified in [1] as an important research area from which ALAN has been identified as a key subtopic in [2]. In our current work, we are attempting to address network and client adaptation problems by deploying proxylets and solving issues relating to dynamic changes. We are also aware of some of the work done on runtime resource management for advanced network services, similar to that in active networking [5, 16, 17, 18]. In [18], adaptive application model is envisaged using network layer active networking technology for any fluctuation in network conditions or resource availability. Agents located in active node perform the network resource sharing actions using a given cost model. This approach doesn't support dynamic resource brokering. The contribution in [16, 17] is mainly sharing of network resources at network layer managed by different service providers. The application adaptation for end device capabilities has not been addressed by [16, 17, 18]. In [5], the combination of policies and biologically inspired adaptive algorithms have been used to enable event driven management system of a network offering active services. The Iceberg [20] project at Berkeley, University of California, has been looking into any-to-any communication services and personal mobility services for mainly voice. Their work matches one of our goals that people will continue to use multiple devices for communication. Iceberg approaches the problem primarily at the network layer and it doesn't address on enabling application access on lightweight end devices and it requires underlying infrastructure support. The Ninja project [23] investigates a software infrastructure for next generation Internet services. The service components can be executed closer to the client to enable transcoding. Though, this work involves constraint based Automatic Service Path Creation, it is a function-oriented method and ignores network link properties and node and link resource constraints, unlike our approach.

Work on constraint satisfaction problem (CSP) [4] has a long history in the field of Artificial Intelligence (AI) engineering [12]. The tendency of utilizing the AI techniques and approach in traditional Distributed Network Management has given researchers very fruitful results [13, 19].

8. Conclusion

We are investigating the problem of proxylet allocation in ad hoc active networks in terms of distributed constraint satisfaction (DCS). The DCS problem is NP-hard, but our approach shows how to form self-configuring transcoding proxylets in an Active Network. Our initial work described in [6] shows how to create different paths between various end devices and web servers, and we have extended this to include dynamic adaptation to failures, link conditions and also to include cost constraints.

The goal of our research is to create technologies for automated self-managing and self-configuring active mobile components for ubiquitous applications. Our approach also allows the client module to be extremely lightweight. The security issues relating to proxylet downloading is not addressed in this paper. The ANDROID project [21] at UCL focuses on security management [22] for the Application Level Active Networking.

9. Acknowledgements

We gratefully acknowledge the support of British Telecom for ALPINE [2] research project as well as comments and suggestions from our colleagues involved in this project.

10. References

[1] D. L. Tennenhouse, and D. J. Wetherall, "Towards the Active Network Architecture", *ACM Computer Communications Review*, vol. 26, no. 2, pp. 5-18, Apr. 1996.

[2] "Alpine: Application Level Programmable Inter-Network Environment", http://www.cs.ucl.ac.uk/research/alpine/

[3] M. Fry, and A. Ghosh, "Application Layer Active Networking", *Computer Networks*, 31, 7, pp. 655-667, 1999.

[4] Mittal and Falkenhainer, "Dynamic Constraint Satisfaction Problems", *In proceedings of the 8th AAAI*, pp. 25-32, 1990

[5] I.W. Marshall, H. Gharib, J. Hardwicke, C. Roadknight, "A novel architecture for active service management",

Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management (IM'2001), Seattle, May 2001

[6] K.T..Krishnakumar, M. Sloman, "Constraint-based Configuration of Proxylets for Programmable Networks", *Proceedings of the 8th International Workshop*, IDMS2001, LNCS 2158, Sept. 2001, pp 245-256.

[7] D. Jackson, I. Schechter and I. Shlyakhter, "Alcoa: the Alloy Constraint Analyzer", *In Proceedings International Conference on Software Engineering*, Limerick, Ireland, June 2000.

[8] M. Kazantzidis, L. Wang, M. Gerla, "On Fairness and Efficiency of Adaptive Audio Application Layers for Multi-hop Wireless Networks." *In Proceedings of the IEEE MOMUC'99*; San Diego, Nov. 1999; pp. 357-362.

[9] T. Nandagopal, T. E. Kim, P. Sinha, V. Bharghavan, "Service Differentiation through End-to-End Rate Control in Low Bandwidth Wireless Packet Networks". *In Proceedings of the 6th International Workshop on Mobile Multimedia Communications*; San Diego, CA, USA, Nov. 1999.

[10] http://muffin.doit.org/, "MUFFIN: World Wide Web Filtering System".

[11] M. Margaritidis, G.C. Polyzos, "MobiWeb: Enabling Adaptive Continuous Media Applications over Wireless Links". *IEEE International Conference in 3G Wireless 2000*; San Francisco, June 2000.

[12] E. C. Freuder, A. Mackworth, 1992. Special Volume, "Constraint-Based Reasoning", *in proceedings of Artificial Intelligence*, 1992, pp.58.

[13] C. Leckie, "Experience and Trends in AI for Network Monitoring and Diagnosis", *Proceedings of the IJCAI-95* Workshop on AI in Distributed Information Networks.

[14] B. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanarayanan, "A conceptual framework for network and client adaptation," *ACM MONET Journal*, vol. 5, No. 4, (Dec. 2000), pp. 221-231.

[15] C. Y. Hsu, A. Ortega, M. Khansari, "Rate Control for Robust Video Transmission over Burst-Error Wireless Channels". *IEEE Journal on Selected Areas in Communications*; May 1999; 17(5): pp. 756-773.

[16] P. Chandra, Yang-Hua Chu, A. Fisher, J. Gao, C. Kosak, T. S. Eugene Ng, P. Steenkiste, E. Takahashi, and Z. Hui, "Darwin: Customizable Resource Management for Value-Added Network Services", *IEEE Network*, vol.: 15 Issue: 1, Jan 2001, pp. 22-35.

[17] E. Takahashi, P. Steenkiste, J. Gao and A. Fisher "A Programming Interface For Network Resource Management", *In Proceedings of the Second IEEE Open Architectures and Network Programming* (OPENARCH'99), pp. 34-44, New York, NY, March. 1999.

[18] L. Yamamoto, and G. Leduc, "An Agent-inspired Active Network Resource Trading Model Applied to Congestion Control", *In Proceedings of the 2nd International Workshop on Mobile Agents for Telecommunication Applications* (MATA'2000), Sept. 18-20, 2000, Paris, France.

[19] Elfe, E. C. Freuder, and D. Lesaint., "Dynamic Constraint Satisfaction for Feature Interaction", *BT Technology Journal*, vol. 16, number 3, July 1998, pp. 38-45.

[20] H. J. Wang, B. Raman, C. N. Chuah., R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J. S. Shih, L. Subramanian, B. Y. Zhao, A. D. Joseph, and R. H. Katz, "ICEBERG: An Internet-core Network Architecture for Integrated Communications," *In IEEE Personal Communications* (2000): Special Issue on IP-based Mobile Telecommunication Networks, vol. 7, 2000, pp. 10-19.

[21] ANDROID: Active Network DistRibuted Open Infrastructure Development at University College London (UCL) http://www.ee.ucl.ac.uk/~pants/projects/android/

[22] O. Prnjat, T. Olukemi, I. Liabotis, L. Sacks, "Integrity and Security of the Application Level Active Networks"; IFIP Workshop on IP and ATM Traffic Management WATM'2001 and EUNICE'2001; Paris, France, Sept. 2001.

[23] S. Gribble, "The Ninja Architecture for Robust Internet-Scale Systems and Services," *in special issue of Computer Networks on Pervasive Computing*, 2000.