Towards Constraint-Based Configuration (CBC) of Proxylets for Policy Implementation

Krish T. Krishnakumar, Morris Sloman

Abstract

With the deployment of Proxylets and Dynamic Proxy Servers for Application Level Active Networks, the streaming of multimedia data over the wired/wireless browser will become very common application requirement. Proxylets may be used to transform the media flows in order to meet application or QoS policy requirements. In this paper, we examine the feasibility of performing Constraint Based Configuration (CBC) of the required proxylets. A set of constraints can be defined to select the required proxylets. A second stage is to define constraints relating to the placement of proxylets on nodes in the network. Eventually we will investigate the use of constraints for dynamic re-configuration to accommodate user mobility, or QoS variation.

1. Introduction

The next-generation internet will need to support the rapid provision of new services, such as on-demand multi-point video conferencing, and the integration of multi-media streams using heterogeneous encodings requiring transformations within the network. This will be supported by programmable network infrastructure which will be able to download control software on-demand to routers and proxy servers. So there will be a demand for policy-based network-resources configuration management for the above programmable network infrastructure.

Configuration management is cumbersome as it is human intensive and involves distributed heterogeneous data for a policy based network. The human action can easily introduce defects to the configuration process because there is no way to verify that configurations actually reflect the policy and if there is any mistake in the process it has to be rectified by hand.

The issue which is being tackled here is the configuration of necessary proxylets, from an available collection, in a programmable network infrastructure. Specifically, we are looking at a possible constraint based configuration to facilitate an automated provisioning of resources for a particular user/service as per the application policy. This require integration of policy based network management with constraint based configuration for a dynamically changing network environment.

The motivation for this proposal is in the belief that more of the configuration process could be automated for the active networks. The proposed system would:

• Accept user specification of the service required with constraints in the form of policies.
• Derive a preliminary set of configurations from the set of available resources in the local resources database.
• Check for constraint violations of the above set of configuration.
• Finalize the configuration which will be utilized to transfer the data stream across the network.
• Analyze a configuration which does not comply with a changed QoS Policy and determine a new configuration which meets the required constraints.

The configuration of software components can be defined by a set of attributes (or components) whose possible values belong to a finite set and a set of feasibility constraints over these attributes which specify their compatible combinations of values. The problem is to find a feasible product (i.e., to choose a value for each attribute) that satisfies not only the feasibility constraints but also some user requirements (such as QoS policy). Real-world problems in computer vision, planning, scheduling, configuration and diagnosing can be viewed as Constraint Satisfaction Problems (CSP).

It is possible to express the QoS that the network should provide in the form of a Policy using the Ponder Policy specification language. A specific set of Proxylet resources will be required to support this policy. We are investigating the feasibility of expressing this selection as a constraint satisfaction problem (CSP) [4]. We are using Daniel Jackson’s Alloy/ALCOA [5] to express and analyze the constraints. The allocation of the proxylets to servers within the network can also be expressed as a CSP concerned with availability of resource such as processing power or bandwidth. Eventually we
would like to extend this approach to cater for dynamic re-configuration of proxylets in which the proxylets may have to move to new nodes to support user mobility, or variations in actual QoS within the network. CBC involves finding values (network resource instances) for configuration variables subject to constraints that restrict which combinations of values are allowed.

Our approach is to have a mechanism to configure the proxylets for a given policy-based content delivery [3]. This mechanism will be able to come up with the necessary configuration setup for any given policy. We have a prototype implementation of a simple scenario using CBC of proxylets.

The remaining sections of this paper are organized as follows: Section 2 outlines the Application Level Active Network (ALAN) approach being used in the project. A simplified example of the ALAN configuration model is explained in Section 3 and architectural implementation is further explained in Section 4; Section 5 and 6 contain our conclusions and plans for further research.

2. ALAN System Overview

The ALAN system [1, 2] consists of clients (browsers) and web servers deployed in the internet. The connections between the servers and browsers are enhanced by Dynamic Proxy Servers (DPS) and protocol entities called proxylets. The proxylet is dynamic code in a single jar file which is downloaded and run on a DPS. The proxylet can be referenced via a URL and resides on web server or DPS itself. The Dynamic Proxy Server is an active network node which accepts requests and creates an environment for the execution of proxylets. The DPSs are selected at optimal distance for an end-to-end path between browser and web server (Figure 1). Typically proxylets can be used for compressing or decompressing a media stream, finding an optimally located remote DPS, verifying the security of mobile code and the platforms to be run on, real-time transport protocol transcoding, multicast reflecting, TCP bridging etc.

As far as the ALAN is concerned, the different functional proxylets have to be selectively combined to satisfy a given policy based QoS. This combinational nature of a configuration can be expressed as a constraint satisfaction problem [4] when it comes to a complicated system with many varieties of functional proxylets.

3. An Example

Typical real applications involve hundreds of constraints and values for each variable, but we have focused on a very simplified example to illustrate the approach. The configuration task for the ALAN system involves predefining a set of components, the knowledge of connection between the components or domains for a customer requirement for a specific configuration respecting all the compatibility constraints between the components.

Here are some constraint variables for ALAN (see Figure 1):

**Browsers:** Access to multi media streams from either wireless or wired devices.

**Payload Type:** The multi media payload packets shall contain video, audio or data streams.

**Mode:** Real time mode (to transcode the video/audio payload in real-time as it is being downloaded e.g.: live video multicast transmission) and download mode (to download the compressed media stream and after the completion of the download, start playing the stream using one of the suitable media tool).

![Figure 1: Application Level Active Networking Architecture](image)
**Proxylet:** It has been assumed that multiple proxylets are available for modifying the content presentation to be more suitable for the client. For example, as today we have WAP phones as internet browsers, the wap-proxylet will be required in the ALAN.

**DPS Location:** This is the location of the DPS in relation to the browser/server. For simplicity, this parameter is assumed to have values as such nearer to the browser, intermittent, nearer to the server. In the future work, we intend to have these values replaced by the hostname of the DPS and from which the relative distance is to be deduced.

**Cache:** This is to notify which type of data payload can be cached at local server.

**QoS Package:** It has GOLD, SILVER and ECONOMY as parameters which indicate the 3 possible classes of service values.

### 3.1 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 \( (P \lor Q) \land (\neg P \lor \neg S) \) 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:

1. **Domain knowledge** - describes the objects of an application and the relationships among them.
2. **Specification of the desired product** - requirements that must be satisfied by the product and the structure or topology of the product.

A Constraint Satisfaction Problem (CSP) is defined as \( P = \langle V, D_v, C_v \rangle \), where

- A set of variables \( V \) representing all the variables that may potentially become active and appear in a configuration.
- \( V = \{V_1, ..., V_n\} \).
- \( D_v \) is the set of domains, with \( val_i \) representing the set of all possible values for variable \( V_i \).
- 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 and their related values in our simplified example and these constraints will be specified via a simplified user interface yielding a text file of constraint specification as shown above in Figure 2. Using these variables and values, we can derive an initial constraint model for the configuration. For this scenario, the domain knowledge is essentially static and it would be possible to find an initial configuration of components for example to support a mobile browser that satisfies the constraints. But as the complexity of the implementation grows the need for mobility of the user will be taken into account and this will have the need for dynamic constraint based configuration (DCBC). In dynamic constraint systems, variables and constraints can be dynamically added or removed in the ALAN system. DCBC can be used for dynamic change in configuration to support proxylet migration / hopping for providing constant QoS or supporting mobility of users. The usage of DCBC is further discussed in section 5 of this paper.

Constraint \((wl =/= video)\) states that if the browser is a wireless device then video content delivery is not supported. The \((wired = audio)\) means that the wired browsers are allowed to download audio clips providing that the constraints between the other necessary variables are not violated for this audio clip download. Constraint \((wl =/= E)\) states that a user with Economy QoS Package is prevented from using wireless browsers to access the services provided in an ALAN system.

---

**Domains (D_v):**

- DPSnode : local, remote
- Proxylet : wap, comp, decomp, location, config, rtpRx, rtpTx, realPlayer
- Browser : wired, wl
- Payload : text, audio, video
- QoS Pack : G, S, E
- Mode : realTime, download
- Cache: true, false

**Variables (V):**

- n : DPSnode
- pl : Proxylet
- b : Browser
- p : payload
- q : QoS Pack
- m : Mode
- c : Cache

**Constraints (C_v):**

\n n, pl : \((local = wap), (local = decomp), (local = location), (local = config), (local = rtpTx), (local = realPlayer), \ldots\)

\( b, p : (wired = text), (wired = audio), (wired = video), (wl = text), (wl = audio), (wl =/= video)\)

\( b, q : (wired = G), (wired = S), (wired = E), (wl = G), (wl = S), (wl =/= E)\)

\( p, q : (text = G), (text = S), (text = E), (audio = G), (audio = S), (video = G), (video =/= S), (video =/= E), (audio =/= E)\)

\( \ldots \)

\( \ldots \)

**Figure 2:** Domain Knowledge - Constraint Variables and Values
3. 2 Product Selection

A Ponder Policy specification [6] forms a statement of the application requirements for a specific product - the use of a mobile handset (browser) for audio delivery. The policy is analyzed to derive an additional set of constraint to construct possible combination of components for the configuration. The most common high-level criteria used in our policy expressions are User-Group, Application, Application-Type, Time/Date, and Device Type/Role. The constraints based configuration (CBC) generated from the Policy shown in Figure 3 are:

| CBC_1 (Package=>Gold, payload=>audio) = { | CBC_2 (Package=Gold, payload=>audio) = { | CBC_3 (Browser=wireless, payload=>text) = { |
| wired-browser, realtime-listening, | wireless-browser, downloadmode, | downloadmode |
| dps-local => (decompression-Proxylet, realtime- | dps-local => (location-Proxy- | dps-local => (location-Proxy-
| audio-player-applet-proxylet, rtpProxy- | let, handoff-Proxy- | let, handoff-Proxy-
| let, location-Proxy- | let, wired-wireless- | let, decompression-
| let,), dps-remote => (compression- | converter-proxy- | Proxylet, compression-
| proxylet) } | proxylet, decompression- | proxylet) } |

The CBC_1 is for the case when the payload is audio and the QoS for the user is Gold which will lead to a real time listening of the audio stream so the required resources to be configured for a wired-browser are:

1. A realtime-audio-player-proxylet, location-proxylet, real-time-protocol (rtp) proxylet to be downloaded on local DPS
2. Compression proxylet at the DPS which is optimally located in the end-to-end configuration topology.

4. Implementation Approach

The derived constraints are indeed in a form of required configuration model. The set of derived constraints can be stored in a database at the Nodes (DPSs) to be retrieved. These constraints dictate the required service resources for the particular policy. So, there is a requirement for solving these constraints using some sort of algorithm. There are many algorithms available for solving the constraint satisfaction problem.

The outcome of the constraint filtering would be the final configuration setup which might contain multiple choices of configuration setup for a given policy. The final configuration setup can be translated into a list of the URLs of the needed proxylets and their corresponding execution environments in a single document using a language such as XML and submitted to the DPS in the load() method. So, we expect a proxylet, that needs to use some form of high-performance zero-copy environment, to be developed once for any configuration setup of the ALAN. The nearest DPS chooses a suitable configuration setup by checking against its resource capability before downloading the computational proxylets or requesting the other DPSs to download the computational proxylets to perform actual service required by the user as stated in the policy. The High level structure of the configuration process is shown in figure 4.
In the ALPINE project, the complex interactions among proxylet system components complicate the configuration process. Not all components are compatible, and certain combinations will not meet functional policy requirements. The first set of components the configure process selects may violate some of these constraints. When this happens, configure (see Figure 4) repeatedly modifies the design until a satisfactory solution is achieved. Typically, a complete system required for the end-to-end HTTP stream transfer is achieved by adding one component at a time, checking for constraints and policy violations after each addition, and making any necessary design modifications as soon as a problem is detected. Checking for constraint violation can be implemented using search (e.g. backtrack) or inference (e.g. arc consistency) algorithms.

The proposed solution also contains a layer across the policy definition and proxylets configuration to employ consistency checking, change propagation of policy, rollback and recovery functions. The proposed system maintains consistent configurations by imposing constraints in the form of Object Constraint Language.

Translating the policies into constraints and policy scripts, then deploying and populating a DPS server with the proxylets will be the ultimate aim of this proposed work. We also have used a Constraint Analyzer called “Alcoa / Alloy” [5] which can perform a deep semantic analysis of models that incorporate complex textual constraints. Using this tool, we have checked the consistency of our constraints, generated sample configurations, simulated execution of operations, and checked that operations preserve constraints.

5. Dynamic Configuration

As our work progresses, we intend to integrate the constraints specification into another aspect of DPS node’s resource management. Allocation of Proxylets to nodes, taking into consideration DPS loading, link bandwidth requirements etc. This includes routing decisions for choosing a path form source to destination.

Note we hope to make use of other work related to route selection. Dynamic configuration of an application may take different forms. At the very first level, programs and users interact with an editable model of the underlying application - adding or removing proxylets, changing inter-DPSs connections or modifying the settings of routings. One of the major requirements for a dynamically configurable system is that it should allow the possibility of modifying the provisioning of the components, without requiring an application restart or any changes to the existing code. In order to integrate a new component to the current topology, the system needs to perform a series of checks to verify whether still constraints are met. There are several interesting directions emerging from this work that need further investigation and they are listed below:

1. Proxylet Migration: It is a mechanism to continue the execution of a proxylet on another node (DPS) and it includes the transport of data stream and execution state of the proxylet. Migration only makes sense under certain network and resource conditions. So, here the basic motivations for proxylet migration are, to support user mobility or re-routing the data stream to maintain demanded QoS at all time. For example, transmission quality varies and fluctuates for a wireless communications due to fading signal which may lead the link between the client and the DPS going down suddenly. This means that the topology which was originally adopted using a constraint has to be reconfigured using another possible CSP. This may be handled by adopting the constraint based re-routing mechanism and the proxylet migration.

2. Dynamically Composing Proxylets (Figure 5): A proxylet can be built using a set of subcomponents [8]. Our research on the policy implementation of a required service in ALAN will further investigate dynamically composing a customized proxylet for a given policy.

3. It is also interesting to further look at extending the component framework of proxylet interface to enable passing of parameters to the proxylet while it is running in order to cater for changing the encoding, compression rate, or adding redundancy during playout of media stream (Figure 5). For example, based on the bandwidth of link and the required QoS for a service, proxylets which can take in compression/decompression factor during runtime, should be available to be deployed into the ALAN network - rather than limiting configuration solutions to a set of pre-calibered functional proxylets.

![Figure 5: Composing Proxylets from Policy](image-url)
4. Active Network Configuration GUI: This is to display the set of DPSs and the available Proxylets in their respective Geographical locations so that they can be chosen by privileged users to configure or customize their own network for individual services in line with the applicable policy. This should enable a drag-and-drop technique [7] for Proxylets to be downloaded on DPSs. This GUI should also be able to check for the constraints of the configuration.

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

6. References


[8] Component Compatibility Markup Language (CCML) for Proxylet components:
http://www.activenet.lancs.ac.uk/ccml/