Proc. 8th International Workshop on Interactive Distributed Multimedia Systems
(IDMS'2001), Lancaster, UK, 4-7 Sep 2001,
Springer LNCS 2158, pp. 245-256

Constraint-Based Configuration of Proxylets for Programmable Networks

Krish T. Krishnakumar, Morris Sloman

Department of Computing, Imperial College,
London SW7 2BZ, UK
{tkkumar,m.sloman}@doc.ic.ac.uk

Abstract. Applications such as multimedia streaming for mobile users, or video conferencing, require support within the network for transcoding, compression etc. Proxylets running on servers within the network may be used to transform the media flows in order to meet application or QoS requirements. In this paper we examine the feasibility of performing constraint based configuration 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. Some preliminary implementations of the architecture are presented and we discuss our approach to incorporate dynamic configuration to cater for load and QoS variations.

1 Introduction

The Alpine project [2] is investigating the use of application-level proxylets which execute on servers within the network to support the rapid provision of new services, such as on-demand multi-point video conferencing or multi-media streaming services for mobile users. A proxylet based system provides many of the advantages of active or programmable network type of services [1], such as transformation of media encodings, to take place within the network but without the security risks of programming at the 'fast-path' level within networks. However there is the need to be able to configure the various proxylets required to support particular applications by selecting suitable proxylets from a database of the those available and deciding on the most appropriate servers on which these proxylets should be loaded. These configuration management decisions can be non-trivial when taking into account the various constraints imposed by the specific application requirements, services which the user may be authorized to use, physical capabilities of mobile computing devices and available quality of service (QoS) provided by a wireless network. Typical users will be non-technical and so will not be capable of making the selections required to support the service they require. Thus, there is a need to automate the configuration of proxylets, taking into consideration all the above constraints. This will avoid the errors which are often introduced, even by technically competent users, when configuring complex systems.

This paper describes initial work on the configuration of required proxylets, from an available collection, in a programmable network infrastructure. We are looking at applying constraint satisfaction techniques to facilitate automated provisioning of resources for a particular user/service in accordance with an application request. This

requires integration of constraint based configuration for a dynamically changing network environment. The proposed system would accept user specification of the service required, derive a preliminary set of proxylet configurations, from the proxylets which support the required functionality, check for constraint violations of the above set of configurations and allocate the proxylets to servers, taking into consideration other constraints such as QoS requirements, available bandwidth, server loading, security risks or charging for use of servers.

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 such as a particular proxylet or server for each attribute which corresponds to a configuration variable) 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) [4]. We are using Daniel Jackson's Alloy/ALCOA [15] to express and analyse the constraints. 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.

This paper describes the prototype implementation of a simple scenario using constraint based configuration of proxylets and indicates how we intend to extend the work to cater for dynamic reconfiguration of proxylets. Section 2 of the paper outlines the Application Level Active Network (ALAN) proxylet approach being used in the Alpine project. In Section 3, we formally define constraint based satisfaction in terms of configuration spaces, and introduce constraint variables and values for the ALAN mechanism. The derivation of proxylet configuration and constraint checks are also presented in section 3 with the customer service requirements. The overall architecture of our approach and the current progress in the implementation are described in section 4. In Section 5, we provide a discussion of future work followed by a discussion of related work in section 6.

2 ALAN System Overview

The ALAN system [3] assumes clients access remote servers across the internet using the HTTP protocol. Protocol entities called Proxylets can be dynamically loaded onto intermediate Dynamic Proxy Servers (DPS) within the network, to perform application-specific functions such as compression / decompression, protocol transformations, multicast reflecting, etc. 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 if not already on the DPS, it is loaded from a proxylet repository. The DPS is an application layer 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 client and end-server (Figure 1). We are also using proxylets to perform the configuration management tasks such as proxylet selection and allocation to opti-

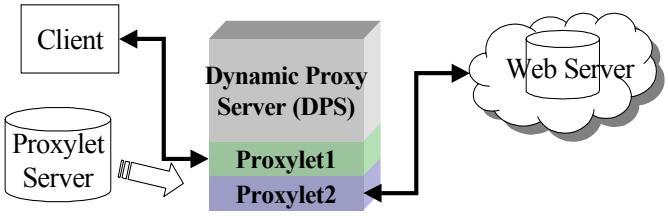

Fig. 1. Application Level Active Networking Architecture

mal DPS servers. The reason is that configuration may be instigated by a mobile user with very limited processing resources, although our architecture does allow for a DPS server to be based on a more powerful mobile device such as a laptop computer.

3 A Simple Scenario

3.1 Example Description

Typical real applications involve hundreds of constraints and values for each variable, but we have focused on a very simplified example based on a user accessing remote media streams from a wireless (wl) or wired device to illustrate the approach. There are three different classes of users, Gold, Silver and Economy which relate to the QoS they will receive as well as whether they can use wireless connections etc. We give some constraint variables for the scenario (see Figure 2):

Browsers: Access to multi media streams from either wireless or wired devices.
Mode: Real time mode (to transcode the video/audio stream as it is being downloaded) and download mode (to first download the compressed media stream, decompress it and then start playing the stream using a suitable media tool).
Proxylet: It has been assumed that multiple proxylets are available for modifying the content presentation to be more suitable for the client device. We assume proxylets are available to cater for WAP phones, compression, decompression, real-time trancoding, a RealPlayer and DPS location selection.
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, intermediate (within the network), nearer to the server.
Cache: This is to indicate whether data can be cached at a local server.
Payload Type: The multi media payload packets may contain video, audio or data streams.
QoS package: There are 3 classes of users - Gold, Silver or Economy which relate to the class of Qos they will receive.

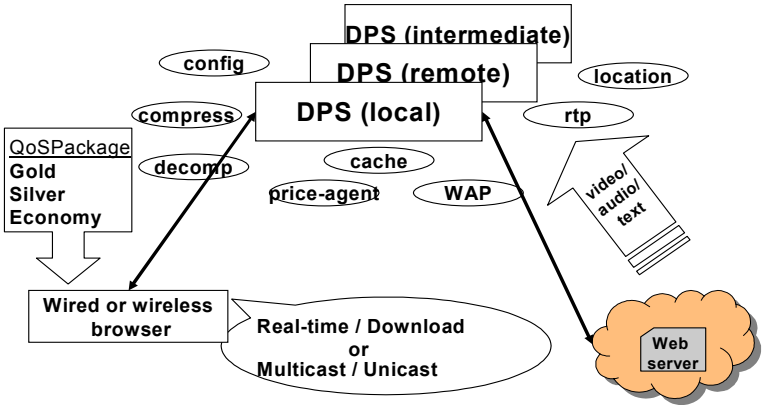

Fig. 2. Simplified ALAN scenario

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. One is *domain knowledge* to describe the objects of an application and the relationships among them. The other is a specification of the desired product which defines requirements that must be satisfied by the product and the structure or topology of the product.

A Constraint Satisfaction Problem (CSP) is defined as $P = <V, Dv, Cv>$, 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 (Cv) to restrict the value assignment of some variables or configure the components according to their behavioral models.

Figure 3a shows some variables and their related values in the example shown in Figure 2. These constraints will be specified via a user interface yielding a text file of constraint specifications. 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.

In Figure 3a, 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 our system.

Domains (D_V):	**Variables (V):**
DPSnode : local, remote, intermediate Proxylet : wap, comp, decomp, deployment, config, rtpRx, rtpTx, realPlayer, handoff Browser : wired, wl Payload : text, audio, video QoS Pack : G, S, E Mode : realTime, download Cache : true, false	n : DPSnode pl : Proxylet b : Browser p : Payload q : QoSPack m : Mode c : Cache
Constraint (C_V): b, p : (wired = text), (wired = audio), (wired = video), (wl = text), (wl =/= video) n, pl : (local = wap), (local = decomp), (local = rtpTx), (local = realPlayer), ... b, q : (wired = G), (wired = S), (wired = E), (wl = G), (wl = S), (wl =/= E) p, q : (text = G), (text = S), (video = G), (video =/= S), (video =/= E), (audio =/= E)	

Fig. 3a. Domain Knowledge - Variables, Values and Compatibility Constraints

In Figure 3b, some of the complete set of constraint based configurations (CBC), which have been derived from the domain knowledge shown in Figure 3a, are listed. These configuration setups in XML will be the output of the system presented in section 4 of this paper.

CBC_1(Pack=>Gold, payload=>audio) ={ wired-browser, realtime-listening, **dps-local** => (decompression-Proxylet, realtime-audio-player-applet-proxylet, rtpProxylet, deployment-Proxylet,), **dps-remote** => (compression-proxylet) }	CBC_2 (Pack=Gold, payload=audio) = { wirelessbrowser, downloadmode, **dps-local** -> (deployment-Proxylet, handoffProxylet, wired-wireless-converter-proxylet, decompression-Proxylet) **dps-remote** => (compression-proxylet) }

Fig. 3b. Sample Constraint Based Proxylet Configuration Setups.

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. In this case, the required resources to be configured for a wired-browser are a realtime-audio-player-proxylet, location-proxylet, real-time-protocol (rtp) proxylet to be downloaded on local DPS and Compression proxylet at the DPS which is optimally located close to the server.

4 Implementation Approach

Step 1: Constraint Verification using Alcoa / Alloy

We are using "Alcoa / Alloy" constraint analyzer [15] 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. 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, in only 65 lines of Alloy code. 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 2 below. The main purpose of using Alloy for initial checking is to reduce development cycle time by detecting errors prior to implementation.

We specify the scenario plus relevant constraints relating to the components in Alloy. Alcoa is a compiler which translates the model definition into a very large boolean formula. This formula then gets solved by the SAT solver in the tool and the solution is interpreted back into Alloy language for the model. Next, a paragraph or schema (e.g. a constraint) in the model is selected to be run for verification. Alcoa responds with an instance or with a message if no instance was found. We may then choose to edit the model, recompile or run. So, using this tool, we are able to analyze the constraints more effectively prior to the Java implementation of our constraint based Proxylet configuration framework.

Figure 4 shows the Alloy specification for the scenario. An invariant (denoted by key word **inv**) defines a constraint in the **model** being defined. For example, inv PLC1 indicates that the wireless browsers does not have the feature of downloading a video stream. Condition (**cond**) gives a hint to Alcoa on generating a sample architectural instance over a given scope.

Step 2: Current Implementation

Initially, all the constraint variables, domain values and compatibility constraints, which were analyzed using Alcoa, are manually derived from an Alloy model definition. For example, the wireless browser is constrained by the **PLC1** invariant in Alloy. This invariant is mapped to the format shown in figure 5 for implementation purposes.

The Constraint Solution Mechanism takes in all these parameters and generates partial configuration setups (subset of constraint variables which satisfy all the of the constraints within the subset) for the constraint specified. These subsets are stored in ConstraintDB as initial basic configuration setups (as shown in Figure 6). The ConstraintDB is currently stored in a text file but a relational database or directory would be needed for a large scale system with 100s of DPSs and Proxylets. The Java Constraint Library has been used to implement the mechanism for populating the ConstraintDB. The constraint variables, values and the compatibility constraints can be

```
model ALPINE {
 domain {browser, proxylet, payload, DPS}
 state {
  // The following are the subjects or values of the variables and
  // their relationships in accordance to the ALOCA syntax.
  disjoint wireless, wired : browser
  partition text, audio, video: payload
  disjoint E, S, G: QoSPack
  partition local, remote : DPS
  partition wapplet, decompresplet, compresplet, configplet, ...
  requires: browser -> local!
  runs: DPS -> proxylet+
  ......
  supports: browser -> payload
  comp : compresplet -> payload+
  decomp: decompresplet  -> payload+
  ......
 }
 ......
 inv PLC1 { all b:wireless | b.supports != video }
 inv PL1 { sole p:locationplet | p in local.runs }
 ......
 cond con1 {
  some DPS1:local|some DPS2: remote|one b:wireless|(DPS1 != DPS2)
 }
 ......
}
```

Fig. 4. Alloy Specification of Scenario Constraints

specified or modified via the ConstraintDB UI. When a modification is made to the constraint specification, the ConstraintDB has to be regenerated to reflect the changes and possibly new proxylets will have to be downloaded to an appropriate DPS and executed when there is a request for a HTTP content delivery.

Figure 6 shows the interactions involved. A DPS near the user's browser is loaded with the *config proxylet,* if it is not already running. The user-specific constraint variables and values are passed to the *config proxylet* to start selecting a suitable configuration setup (e.g. the parameters required by the config proxylet in the scenario are type of browser, class of service and the content type to be downloaded). In a real system, these parameters could be detected automatically, but in our implementation, they are entered manually via a user interface.

For a given set of constraint variables and their values, the ConstraintDB is queried and searched for a set of suitable configuration subsets to be composed to make a complete parameter list. The configuration process also involves identifying a DPS close to the web server which contains the requested content. During this process, the Proxylet

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

Fig. 5. Constraints between Browser and Payload

Store (PIB) is also checked to determine if the required proxylets are available in the network. The PIB is currently an XML file but could be a database or directory for a large scale system. If the *config proxylet* could not find a suitable configuration satisfying the compatibility constraints e.g. because a particular proxylet cannot be found or a suitable server is not available, then an error is flagged.

The final configuration is translated into a list of the URLs of the needed proxylets and their corresponding execution environments in a single XML document and submitted to the DPS *deployment proxylet* via the load() method. The *deployment proxylet* is started by the *config proxylet* on the same DPS. The configuration setup is also displayed in the GUI in the form shown in Figure 3b. For demonstration purpose, this GUI also allows the user to select whether the device is wired or wireless; class of user; video, audio, or text service etc. The *config* and *deployment proxylets* can contain multiple threads each of which is working on a different configuration for multiple users.

The *deployment proxylet* running at a local 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 the actual service required by the user. The *deployment proxylet* will also be able to determine whether installing a set of proxylets (on one or more DPSs) violate constraints relating to available resources by checking against the properties of DPSs.

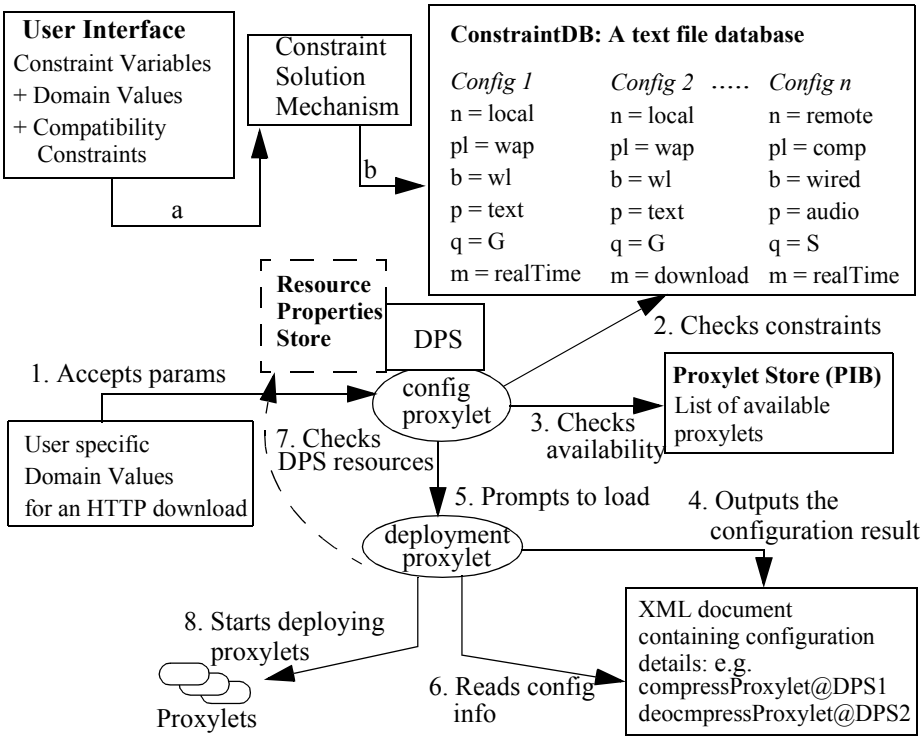

Fig. 6. Prototype Implementation of Proxylet Configuration Framework

However, this feature is not currently implemented in our experimental model. The availability of the resources for the constraint based configuration can be reserved. We intend to extend the *deployment proxylet* to perform checks on the bandwidth of the links between the DPS-local and DPS-remote to make sure it can handle the bit-rate required by the user. The current Framework also has a User Interfaces (UI) for modifying the configuration and specifying constraints.

5 Further Work

We intend to apply constraint specification to resource management so that allocation of Proxylets to nodes can take into consideration DPS loading, link bandwidth requirements etc. This includes routing decisions for choosing a path from source to destination. Dynamic configuration of an application may take different forms:

- Programs and users may interact with an editable model of the underlying application - adding or removing proxylets, changing intermediate-DPS or modifying the settings of connections. 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 for constraint compatibility.
- User mobility may result in variations of QoS due to fading or loss of radio signal, which DPS is local to the user etc. Thus the configuration constraints will dynamically change requiring updates to the topology. Connections between DPSs may have to be rerouted, or a proxylet performing a ‘local’ function may have to be migrated to a different DPS which is now local to the user. This means that the topology which was originally adopted has to be reconfigured using a different CSP.
- We are also looking into using the Ponder policy specification language [6] to define authorization policy in terms of what services a user is permitted to access and QoS policy in terms of what resources such as bandwidth should be allocated for an application or a user. Our approach is to translate these policy statements into constraint specifications for the proxylet deployment.

There are several interesting issues emerging from this work that need further investigation. One of the key issues is *Proxylet Migration.* It is a mechanism to continue the execution of a proxylet on another DPS node 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 required QoS. This may be handled by adopting a constraint based re-routing mechanism.

We are concentrating on the infrastructure support for constraint based configuration, but in order for this to be useful to non-technical users, a very simple user-interface

(UI) would be needed. We consider that it will be easier to investigate the UI once we have a clearer idea of the full functionality of the infrastructure support. It may be possible to make much of the functionality completely transparent to the user. For example, the user may not need to select a class of service, but would be allocated one depending on what service they have signed up for.

6 Related Work

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 problems in deploying proxylets and solving issues in dynamic routing with proxylets for programmable networks. We are also aware of some of the work done on runtime resource management for advanced network services, similar to that in active networking.

Work on constraint satisfaction problem (CSP) has a long history in the field of Artificial Intelligence (AI) engineering [10]. CSP has been used for a range of applications such as aircraft maintenance scheduling, route planning for setting up the links for virtual private networks and various monitoring and control applications [11], but we are unaware of any other applications of CSP to configuration management in a programmable or active networks.

Programmable networks is a very active research area, but there is not much work on configuration of components. In the early work on ALAN [3], a similar application scenario based on streaming audio was described, but the proxylet configuration was performed manually. BT Colleagues in the ALAN project are using policy-based adaptive control techniques with genetic algorithms to configure proxylets for services [5, 13]. The idea is that the 'useful' proxylets, e.g. those that generate income for a service provider, will be replicated and propagated around the network whereas less useful ones will gradually die out. In our environment this would improve the probability that a required proxylet is already loaded in an appropriate DPS, but it would still have to be configured into a specific application. The paper does not address the issue of how to select a set of distributed proxylets for a particular application or service.

Lancaster University colleagues are working on composing a complete functional proxylet for a service from predefined set of sub-components specified using XML [12]. The selection criteria could be based on the characteristics of the specific DPS platform. Part of our future work activities will be to investigate whether our constraint approach can be used to automate the composition of proxylet sub-components at runtime or whether the funtionality of the proxylet can be adapted by means of invoking management operations with new configuration parameters during runtime.

There is considerable interest in using policies as a means specifying adaptive behaviour in programmable networks. Some initial work on policies for both management and security is described in [7] and work on policy based content delivery in an ALAN environment is described in [14]. However, none of the above have addressed configuration of proxylets in a policy based AN. In ALAN, an enforcement of management or authentication policy means systematically making use of functional proxylets in the

system. Some of our work will look at how the constraints can be deduced from, for example, authentication policies.

In our project, we considered a novel approach for the configuration of proxylets in ALAN with two main domains in mind: constraint based configuration and support for dynamic routing. This led to the design of new framework for the configuration management of proxylets. First we targeted the new trend in networking related to "policy based content delivery", then we approached the problem of how the proxylets can be configured without violating any pre-defined constraints. The aim is to produce an interactive real-time display for immediate diagnosis of configuration problems. There is not much work currently being done on interactive configuration management systems for active programmable networks. Our past work on interactive configuration management of distributed systems [8] inspired us to look at whether any of these ideas can be applied to proxylet configuration supporting the mobility of users. The Active Node (DPS) built with our above-mentioned framework can allocate resources to the various virtual end-to-end networks, undertaking configuration and reconfiguration functions with routing decisions and making the network more responsive to users' demands.

7 Summary

In this paper, we described our initial implementation of constraint based configuration of proxylets. We have shown that it is possible to automate the selection and deployment of proxylets while satisfying diverse constraints related to a QoS class, what the users are permitted to do and what types of devices they are using. Although we have focused on configuration of proxylets in a programmable network environment, this approach could be used for deployments of software components in any distributed processing environment.

An ubiquitous computing environment [16] could benefit from our framework. The mobile users will be able to join in the ALAN network without the need touse pre-configured or resource rich mobile browsers. The on-demand services could be dynamically composed and executed on local or intermediate nodes with the help of functional proxylets. For example, the user does not have to figure out which system supports an on-demand service since our framework could deploy the necessary components and protocols along the end-to-end path to enable a service to be reached at an end system.

8 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.

References

1. D.L. Tennenhouse and D.J. Wetherall, “Towards the Active Network Architecture”, ACM Computer Comms. 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, pages 25-32, 1990
5. I.W.Marshall and P.Mckee, "A Policy Based Management Architecture for Large Scale Active Communication Systems" in Policies for Distributed Systems and Networks, LNCS 1995, ed. Sloman, Lobo and Lupu, Springer-Verlag 2001
6. N. Damianou, N. Dulay, E. Lupu, and M. Sloman: “The Ponder Specification Language”, Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39.
7. M. Sloman and E. Lupu, “Policy Specification for Programmable Networks”, In proc. 1st Int Working Conference, IWAN’99, Berlin, Germany, June/July 1999, LNCS 1653, p. 73 ff.
8. H. Fossa and M. Sloman, “Implementing Interactive Configuration Management for Distributed System”, Int Conf on Configurable Distributed Systems (ICCDS’96), Annapolis, Maryland, May 1996, Proceedings pub by IEEE press.
9. “Java Constraint Library”, http://liawww.epfl.ch/~torrens/Project/JCL/.
10. E. Freuder, and A. Mackworth, “Constraint-Based Reasoning”, Artificial Intelligence, 1992, 58.
11. C. Lecki, “Experience and Trends in AI for Network Monitoring and Diagnosis”, Proceedings IJCAI-95 Workshop on AI in Distributed Information Networks.
12. S. Simpson, P. Smith, M. Banfield, and D. Hutchison, “Component Compatibility for Heterogeneous Active Networking”, Presented at IEE Informatics, Nov. 2000, London. http://www.activenet.lancs.ac.uk/papers/ieealan2000.pdf
13. I.W. Marshall and C.M. Roadknight "Adaptive Management of an Active Services Network" BT Technical Journal special issue on "Biologically Inspired Computing", 18, 4, pp78-84 Oct. 2000.
14. G. MacLarty and M. Fry, “Policy-based Content Delivery: an Active Network Approach”, 5th Int Web Caching and Content Delivery Workshop. http://www.terena.nl/conf/wcw/Proceedings/S7/S7-2.pdf
15. D. Jackson, I. Schechter and I. Shlyakhter, “Alcoa: the Alloy Constraint Analyzer”, Proc. ICSE, Limerick, Ireland, June 2000.
16. D. Milojicic, A. Messer, P. Bernadat, I. Greenberg and W. Schroder-Preikschat, “ψ-Pervasive Services Infrastructure”, HP Labs, Palo Alto, USA, http://www.hpl.hp.com/techreports/2001/HPL-2001-87.html.