Adapting Distributed Applications Using Extensible Networks

Scott Thibault, Jérôme Marant, Gilles Muller

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

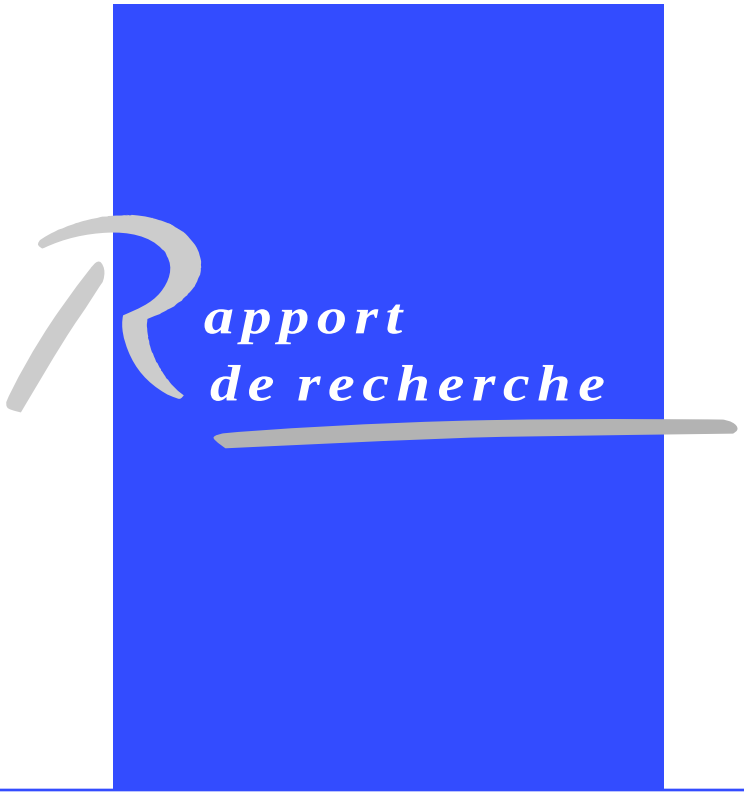
Adapting Distributed Applications Using Extensible Networks

Scott Thibault, Jérôme Marant, Gilles Muller

N ° 3484

Septembre 1998

THÈME 2



ISSN 0249-6399

Adapting Distributed Applications Using Extensible Networks

Scott Thibault, Jérôme Marant, Gilles Muller

Thème 2 — Génie logiciel
et calcul symbolique
Projet COMPOSE

Rapport de recherche n3484 — Septembre 1998 — 22 pages

Abstract: In this paper, we show that the adaptation of distributed software components can be performed by extending network behavior with Application-Specific Protocols (ASPs). We propose to program ASPs in PLAN-P, a domain specific language designed for active networks. We illustrate the application scope of ASPs with three examples: (i) audio broadcasting with bandwidth adaptation in routers, (ii) an extensible HTTP server with load-balancing facilities, (iii) a multipoint MPEG server derived from a point-to-point server. To reconcile portability and efficiency, the PLAN-P run-time system includes a JIT compiler which is generated automatically from a portable interpreter. Measurements show no performance degradation due to PLAN-P; an ASP can be as efficient as a built-in C version of the same program. Finally, we show that our implementation framework for PLAN-P can be easily evolved, since it is based on an interpreter.

Key-words: Adaptation, Active Networks, Domain-Specific Languages, Specialization, Interpreters, Audio broadcasting, Extensible HTTP server.

(Résumé : tsvp)

This research is supported in part by France Telecom/CNET

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Adaptation d'applications distribuées au moyen de réseaux extensibles

Résumé : Dans ce papier, nous montrons que l'adaptation de composants logiciels dans les systèmes distribués peut être réalisée par extension du comportement du réseau par des protocoles dédiés à une application, appelés ASPs pour *Application-Specific Protocols*. Nous proposons de programmer les ASPs en PLAN-P, un langage dédié à la conception de réseaux actifs. Nous illustrons le domaine d'application des ASPs au moyen de trois exemples : (i) la diffusion de données audio avec adaptation de la bande passante dans les routeurs, (ii) un serveur HTTP extensible avec répartition de la charge, (iii) un serveur MPEG multipoint dérivé d'un server point-à-point. Afin de reconcilier portabilité et efficacité, le système d'exécution de PLAN-P repose sur un compilateur juste-à-temps (JIT) qui est généré automatiquement à partir d'un interprète portable. Nos mesures ne montre pas de dégradation de performances duent à PLAN-P ; un ASP peut être aussi rapide qu'une version du même programme résidante dans le routeur et écrite en C. Finalement, nous montrons que le schéma de conception utilisé dans le développement de PLAN-P facilite les évolutions futures, en raison de sa structuration reposant sur un interprète.

Mots-clé : Adaptation, Réseaux actifs, Langages dédiés, Spécialisation, Interprètes, Diffusion de données audio, Serveur HTTP extensible.

1 Introduction

For several years, the evolution of distributed computing has been influenced by several factors including: (i) the diversification of the type of information exchanged from numeric data to multimedia information such as audio and video; (ii) migration of the software architecture from the simple client-server scheme to complex n-clients/m-servers distributed architectures with data broadcasting ; (iii) an increasing variety of hardware architectures (*i.e.,* laptop, PC, workstation) and networks (*i.e.,* mobile phone, wavelan, Ethernet, Fast-Ethernet). This diversity in software, hardware, and data raises the need for solutions permitting the adaptation of existing distributed software components to a changing environment.

Transparent software reuse is one of the key issues in adaptation, since todays software components are often too complex to be easily modified, and their source code is generally unavailable to the end-user. The aim of this paper is to show that certain adaptations can be achieved by changing the way components communicate without alterating the components themselves. For that, we propose to extend the network behavior with Application-Specific Protocols (i.e., ASPs) that can be loaded both in routers and in client/server machines.

Application Specific Protocols

ASPs are derived from active network protocols [16, 34]. While active networks have been targeted toward network routing and monitoring, ASPs additionally perform various operations on packets (*e.g.*, (un-)compression, data filtering, string matching) that enable one to project new behaviors onto an existing application. For instance, it is possible to compress or degrade information, so as to reduce bandwidth consumption and avoid congestion on a high loaded link. ASPs provide a simple way of connecting together existing components while providing new functionalities. For instance, by designing an ASP that routes requests over several machines, one can build an easily-scalable cluster-based server. In such applications, the main advantage of ASPs is to provide a high degree of configurability.

The fact that ASPs can be downloaded into routers permits new quality of service (QoS) functionalities to be added to multipoint applications, such as video and audio broadcasting, as shown in section 3.1. As an example, PLAN-P provides primitives that can be used to degrade a 16 bit stereo audio signal into an 8 bit stereo/monaural signal.

While QoS adaptation can be easily implemented at the end-points in a simple client-server architecture [7, 26], directly transferring this strategy to multipoint applications reduces the global performance to the bandwidth of the slowest segment of the architecture. By performing the QoS adaptation on the router, signal degradation is on a per segment basis without affecting the global architecture. Also, since measurements are performed locally on the router, adaptation can occur rapidly in reaction to changes in the environment. In contrast, feedback-driven approaches [7, 21] must wait for the results of a distributed computation.

Extensible Network Support for Programming ASPs

The introduction of extensibility into networks, however, raises a number of issues [34]. A first problem is that of safety and security; A protocol that is dynamically loaded into a kernel could be a Trojan horse and network routers are shared resources. A second problem is portability. Given the heterogeneity of the network, the module executing ASPs should be highly portable. A third problem is efficiency. The execution of ASPs must be efficient in order to maintain the traffic rate of the application.

We address these issues by programming ASPs in an extended version of the PLAN-P language [36]. PLAN-P is a Domain Specific Language (i.e., DSL) that we originally designed for writing active network protocols. Using PLAN-P provides the following advantages:

- **Portability without loss of efficiency.** The easiest way to offer portability is to have an execution scheme relying on source code download and interpretation. However, source code interpretation generally leads to very poor performance [29]. The PLAN-P solution to the dilemma between performance and portability is to automatically generate a *Just-In-Time* (JIT) compiler from a portable interpreter written in C. JIT compiler generation relies on a program specializer for C called Tempo [10]. From the PLAN-P interpreter, Tempo generates a dedicated run-time specializer (*the JIT compiler*) which can be embedded in the router and host kernels. Binary programs produced by the JIT compiler are efficient; we have shown that a PLAN-P Ethernet bridge can be as efficient as an in-kernel built-in C programmed bridge [36].
- **Safety and security.** Because of their simple semantics and restrictions, DSLs make it possible to determine properties typically undecidable in general-purpose languages (*e.g.,* termination) [16, 19, 30]. PLAN-P programs can automatically be checked to guarantee that: packets do not cycle within the network (without a run-time resource bound), all packets are delivered, and packet duplication is linear. Each of these properties can be automatically checked due to certain restrictions or domain-specific attributes of the language, without which, automatic proof would be impossible.

Contributions of this paper

PLAN-P was initially focused toward packet routing. In order to design ASPs, we have extended PLAN-P with new constructs and primitives that enable concise packet treatment and higher level functionalities. To summarize, the contributions of this paper are the following:

- **Adapting distributed applications with Application-Specific Protocols.** We show that ASPs are a flexible, rapid and efficient solution for adapting distributed applications in order to enrich them with new functionalities without changing the original application code. We illustrate this idea with three examples that demonstrate the application scope of ASPs: (i) audio broadcasting with bandwidth adaptation in routers, (ii) an extensible HTTP server with load-balancing facilities, (iii) a multi-point MPEG server derived from a point-to-point server with intelligent MPEG traffic

duplication. These examples have been tested on a LAN, with SUN workstations as routers and the PLAN-P run-time system integrated as a Solaris kernel module.

From the performance standpoint, no traffic rate degradation is induced by the ASP on the audio and MPEG examples. For the virtual HTTP server example, the ASP-based server is as efficient as a built-in C programmed server. Also, we are able to serve up to 1.75 the load of a single server with a cluster built from two physical servers.

For each of these examples, the average size of the ASP is about 130 lines of PLAN-P. Such conciseness allows easy maintenance and a rapid adaptation to evolving needs. For the audio broadcasting example, this permits testing and evaluating different bandwidth adaptation policies. In the case of the virtual HTTP server, the ASP can be easily changed so as to permit the addition/removal of a physical server, or to match a new network topology.

- **Operating System Design and Domain-Specific Languages**

  We show that our DSL framework for adaptation can be easily evolved to meet changing requirements. Adaptive systems is a research area where the needs and solutions are still rapidly evolving. Therefore, the operating system designer is faced with conflicting problems, such as how to guarantee the flexibility and performance of a system component while still being able to easily debug it. One solution to providing extensibility is to provide a DSL interface to operating system components. Our framework based on DSL and interpreter specialization provides flexibility for an evolving DSL while maintaining the efficiency of a compiled DSL.

  The advantage of our framework is that an interpreter is simple to extend compared to a compiler. Because the language is small, an interpreter for a DSL such as PLAN-P is also small (about 8000 lines of C); that makes even easier its modification. A second advantage of our framework is that new functionalities can be tested within the interpreter, as long as good performance is not required. Therefore, generation of the JIT compiler from the interpreter can be delayed until functionalities are debugged. Due to this high degree of flexibility, evolving PLAN-P from packet routing to ASPs was easy, and did not require major restructuring.

The rest of the paper is organized as follows. Section 2 presents the PLAN-P language, safety properties that can be verified for PLAN-P programs and the execution scheme relying on an automatically generated JIT compiler. Section 3 describes the three applications we have developed with ASPs. Section 4 details related work. Section 5 describes our future plans for enriching PLAN-P and addressing other applications. Section 6 concludes with assessments.

2 Overview of PLAN-P

The PLAN-P language is originally based on PLAN, a Programming Language for Active Networks [16]. While PLAN-P retains most of the SML-like syntax of PLAN, the semantics

are significantly different in order to treat a larger scope of applications, ranging from pure active network protocols to ASPs.

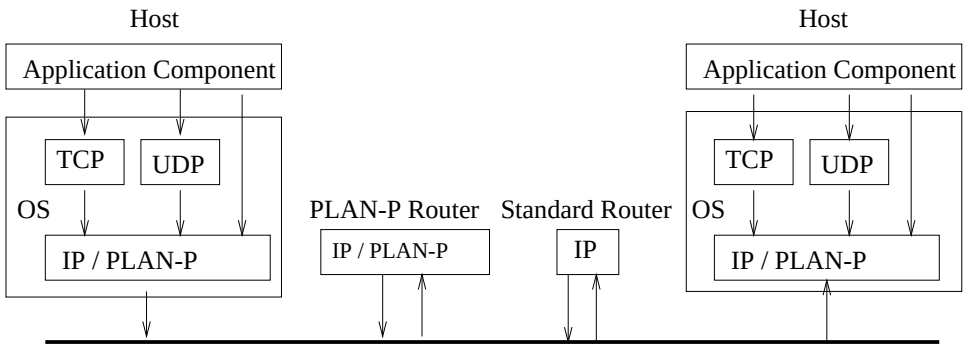


Figure 1: PLAN-P software architecture

Figure 1 depicts the PLAN-P architecture. This architecture is based on IP and allows PLAN-P programs to be used to build applications based on IP, UDP, or TCP. The PLAN-P system does not require any changes to existing packet formats, and thus, PLAN-P routers operate seamlessly within existing networks. PLAN-P programs are downloaded into the IP/PLAN-P layer on the end-host machines and/or any number of routers in the network. These programs replace the standard packet processing behavior of the IP layer in order to perform application-specific processing. A PLAN-P program can specify new behavior for all packets or only packets belonging to a given application.

The definition of a PLAN-P protocol consists of a set of channel specifications. Each channel represents a stream of packets that are treated with the same behavior. Channels are described by a channel function, which defines the name of the channel, the type of the channel state, the type of packets the channel applies to, and the behavior of the channel on these packets. Figure 2 shows an example channel definition. Each channel function has three parameters: the protocol state, the channel state, and the packet. The protocol state is shared between all channels, whereas the channel state is local to the channel. When packets are sent on a user-defined channel, the packet is tagged for identification. However, in order to treat packets from existing applications, channels given the distinguished name `network` specifies behavior that is applied to all packets that match the type specified for the packet argument. For example, the packet type in figure 2 is `ip*tcp*blob`[1], and thus specifies the behavior to be applied to all TCP packets.

The execution model of a `network` channel is as follows. When a packet arrives, its type is determined from the packet headers. If the type matches the type specified in the channel declaration then the channel function will be invoked to process the packet. The function is passed as arguments the current protocol state, the current channel state, and the packet.

[1] This is a tuple type in ML notation. The $\#n$ operator extracts the n th element.

After processing the packet, the channel function can forward the packet or a new packet using the `OnRemote` primitive. Finally, the function must return new values for the protocol and channel state.

The PLAN-P fragment in figure 2 is taken from the load balancing HTTP server protocol (see section 3.2). The fragment detects incoming requests and forwards them to one of two physical servers. The `getSetS` function, not shown, picks a physical server and records the chosen server in a hash table in order to forward future packets on the new connection. All other functions calls shown are to built-in PLAN-P primitives.

```
...
channel network(ps : int, ss : (int*host*host) hash_table, p : ip*tcp*blob)
  initstate mkTable(256) is
  let
    val iph : ip = #1 p
    val tcph : tcp = #2 p
    val body : blob = #3 p
  in
    if (tcpDst(tcph) = 80) then
      -- incoming HTTP requests
      let
          val con : int =  getSetS(ipSrc(iph), ipDst(iph), (tcph), ss, ps)
      in
        if (con = 0 ) then
          -- replace the logical server by server 0
          (OnRemote(network, (ipDstSet(iph, 131.254.60.81), tcph, body));
           (con,ss))
        else
          -- replace the logical server by server 1
          (OnRemote(network, (ipDstSet(iph, 131.254.60.109), tcph, body));
           (con,ss))
      end
    else
      ...
```

Figure 2: PLAN-P fragment of a load balancing protocol for an HTTP server

2.1 Safety and Security

One of the most important advantages of using a DSL for extensible systems is the possibility of automatic verification for safety and security. This is particularly important for system software that operates within the kernel. Security is important because the kernel is shared among users. Safety is important because of the difficulty of debugging code within the kernel. Also, in the case of PLAN-P, applications are distributed, further increasing the difficulty of debugging. In this section, we present several properties that are inherent in

or can be verified for PLAN-P programs due to the fact that PLAN-P is a DSL. These properties are described in more detail in a previous paper [36].

Local termination. Local termination means that the local processing of a packet on a given node is guaranteed to terminate. PLAN-P programs, by construction, are guaranteed to locally terminate. This is a direct result of restricting the language to not allow recursion or unbounded loops.

Global termination. Calls to the `OnRemote` primitive represent a recursive call on a remote machine. If this recursion does not terminate, a packet may cycle forever on the network. Thus, although restrictions on the language guarantee local termination, they are not sufficient to guarantee global termination. One solution is to introduce a resource bound, which is decremented on each hop, similar to the time-to-live field of IP [28]. This is not entirely satisfying because it introduces a safety problem of unintended program termination. However, by making an assumption that the IP routing tables do not contain cycles, we can prove that PLAN-P programs do not cycle.

Guaranteed packet delivery. It is also possible to statically check that all packets are delivered. To prove this safety property, we assume that the underlying network is reliable (*i.e.,* it does not lose packets)[2]. The basis of the proof is as follows. If packets are guaranteed not to cycle (as proven above), the program handles all exceptions (*i.e.,* it can not terminate due to an unhandled exception), and packets are forwarded (or delivered) for all execution paths (*i.e.,* the program does not intentionally drop packets), then we are guaranteed that the packet will be delivered.

Safe packet duplication. Finally, it is possible to verify that packets are not duplicated in an exponential manner. This property is verified by checking that for all execution paths there exists at most one `OnRemote` or `OnNeighbor` statement whose channel argument might create copies of a packet. This property is proved using a standard fix-point induction [2].

It is desirable to verify security properties within the PLAN-P run-time system in order to prevent denial of service due to erroneous or malicious programs. Thus, when programs are downloaded into the network layer, programs should be analyzed and rejected if they can not be shown to terminate or to exhibit non-exponential packet duplication. Although the global termination and packet duplication analyses developed for PLAN-P are conservative, they cover a sufficient number of cases. Even in cases where a "good" protocol is rejected, it is sometimes possible to alter the protocol such that it will pass the analyses. Of course, there are also some legitimate protocols that can not be proven to terminate (*e.g.*, packet forwarding for mobile computing) or may duplicate packets exponentially (*e.g.*, multicast). In this case, authentication should be used to allow privileged users to download protocols that are not verified.

[2]Since we are interested in the reliability of the *program* and not the network, this assumption is appropriate.

The program verification/authentication system has not yet been implemented. However, the analyses are not complicated and could be implemented with existing tools for generating program analyses and model checking. The verification of termination is based on exhaustive state exploration while that of packet duplication is based on a fix-point computation. The size of the state space that must be explored for termination is on the order of $rd2^d$, where r is the number of calls to `OnRemote` or `OnNeighbor` and d is the number of possible end-host destinations. Both of these numbers are typically small. For most protocols, the only two IP addresses available to the program are the source and destination address of the IP header, and thus, $d = 2$. The fix-point analysis to check for packet duplications assigns a boolean value to each protocol channel on each iteration. Thus, the number iterations required to reach a fix-point are at most 2^c, where c is the number of channels.

2.2 Automatic Generation of a JIT Compiler

While the use of a DSL can allow automatic verification of important program properties, these analyses require the program source in order to perfrom *late checking*. Consequently, the DSL program must be interpreted or compiled at run time when the program is received. While interpreters are simpler to write than compilers, they are also slower. We have previously shown that, using a technique called partial evaluation, an efficient implementation of a PLAN-P program can be automatically generated from an interpreter [35, 36]. Furthermore, this can be done at run time, providing the functionality of a just-in-time compiler (JIT).

Partial evaluation is an automatic program transformation that, when applied to interpreters, transforms a language interpreter into a program generator for that language [17]. Traditionally, this program generator generates programs in source code form. Consel and Noël have developed a technique, called run-time specialization, which permits partial evaluation to be performed at run time [11]. When applied to interpreters, the result is a program generator that generates programs in machine code. Therefore, by using run-time specialization, a partial evaluator can be used to automatically generate a PLAN-P JIT from a PLAN-P interpreter.

	Audio Broadcasting (router)	Audio Broadcasting (client)	Extensible Web Server	MPEG (monitor)	MPEG (client)
Number of lines	68	28	91	161	33
Code generation time (ms)	11.6	6.2	15.3	53.9	6.1

Figure 3: Code generation time (ms) for PLAN-P programs)

The approach proposed by Consel and Noël performs run time code generation by assembling and patching machine code templates at run time. These templates, and the program that assembles and patches them are automatically generated and compiled at compile time.

This approach is very portable because the machine code templates are generated using a standard C compiler (*i.e.*, `gcc`). Since this approach only assembles and patches templates at run time, it is very efficient. Figure 3 shows the time required to generate code for the PLAN-P programs used in the experiments described in section 3.

2.3 Extending PLAN-P for ASPs

In order to facilitate the development of ASPs, we have made some extensions to PLAN-P. These changes include the addition of some language primitives and the ability to define multiple channels of the same name (overloaded channels).

Since ASPs represent a slightly different domain than active network applications, it is normal that some new primitives would be required. Extending the interpreter with a new primitive involves defining two C functions. One function performs the calculation of the primitive, while the second function computes the return type of the primitive given the types of its arguments. Since the implementation framework is based on the partial evaluation of an interpreter, after extending the interpreter, one need only regenerate the specializer.

```
val CmdA : int = 1
val CmdB : int = 2

channel network(ps : unit, ss : unit, p : ip*tcp*char*int) is
  if charPos(#3 p)=CmdA then
    (print("CmdA: "); println(#4 p); (ps,ss))
  else
    (ps,ss)

channel network(ps : unit, ss : unit, p : ip*tcp*char*bool) is
  if charPos(#3 p)=CmdB then
    (print("CmdB: "); println(#4 p); (ps,ss))
  else
    (ps,ss)
```

Figure 4: Overloaded channel example

The ability to have overloaded channels was introduced to facilitate the description of protocols for existing applications. When writing new PLAN-P applications, the application sends differently typed packets on different user-defined channels. The channel identifier is stored in the packet. However, when writing protocols for existing applications, packets do not contain explicit type information, and a special PLAN-P channel must be used which treats all packets. In this case it is up to the programmer to filter out the packets that correspond to the application. For example, in the PLAN-P fragment shown in figure 2, the first statement uses the TCP destination port to test whether the packet is an HTTP packet. A filtering problem arises when the same application sends different types of packets

on the same connection. For example, an application might send packets whose first byte specifies the type of the packet, and the data which follows depends on the value of this first byte. Since channel packets are typed, multiple channels are required for the different types of packets. The example in figure 4 shows how overloaded channels can treat these applications.

2.4 Current Status and Availability

The PLAN-P run-time system has been implemented as a Solaris loadable kernel module. This module includes both an interpreter for PLAN-P and the PLAN-P JIT generated using partial evaluation. The JIT was generated using Tempo, a partial evaluator for C which performs both compile-time and run-time specialization [9, 10, 22]. Our experiments with this module have shown that a PLAN-P program compiled with this JIT incurs no overhead in overall system performance in comparison to the same program written in C and compiled off-line with `gcc` [36]. Furthermore, in comparison to Java, which is another mobile code approach, the generated program is twice as fast as an equivalent Java program compiled with Harissa [23], an optimizing off-line byte-code compiler.

The size of the interpreter is about 8000 lines of C, excluding the lexer and parser which are automatically generated. The size of the Solaris module is about 1Mb and includes the lexer, parser, PLAN-P interpreter, and PLAN-P JIT. One third of this size is due to the lexer and parser. These could be removed by downloading a binary abstract syntax tree rather than program text.

Tempo is currently available for Sparc and Intel architecture. It can be retrieved from `http://www.irisa.fr/compose/tempo`. The PLAN-P run-time system and all experiments described in the next section are available from `http://www.irisa.fr/compose/plan-p`.

3 Experiments

In this section, we illustrate the application scope of ASPs with three experiments. The first experiment describes how to add Qos adaptation to an existing audio broadcasting application. The second experiment shows how to build an extensible HTTP server. The third experiment presents how to transform a point-to-point MPEG server into a multipoint one.

3.1 Audio Broadcasting

This section presents an experiment with a distributed audio broadcasting application. The experiment demonstrates the use of PLAN-P to add new QoS functionality to an existing application, without changing the application. Additionally, this new functionality provides finer-grain adaptation than that possible by modifying the application to use software feedback as in end-to-end approaches [6]. The audio broadcasting application is a simple utility that broadcasts CD quality audio from an audio CD or radio card using IP multicast.

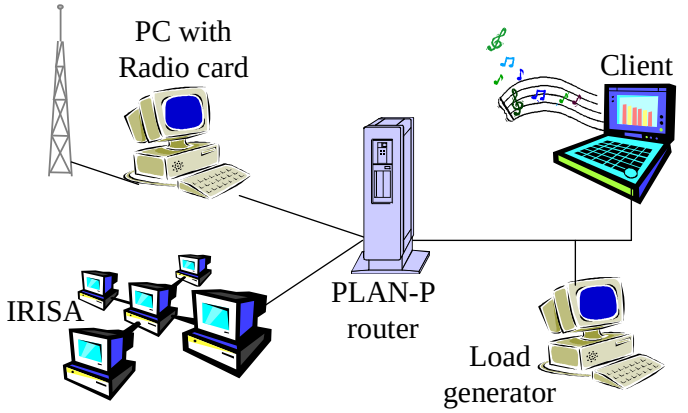


Figure 5: Audio broadcasting network architecture

The goal of the experiment was to use PLAN-P to add network adaptation to the audio application. The adaptation strategy chosen for the experiment is to adapt the audio quality in order to control the bandwidth consumed by the audio traffic. When bandwidth is limited, the audio quality is reduced, and thus, audio traffic is reduced. Although, the client receives lower quality sound (*e.g.*, 8 bit instead of 16 bit), this leads to reduced delays and packet loss which otherwise would result in more unpleasant, choppy sound. The current implementation has three levels of audio quality: 16 bit stereo, 16 bit monaural, and 8 bit monaural. Of course, there are many other strategies, such as layered multicast [21], that one could envision to adapt to network conditions. The advantage of PLAN-P is that strategies can be quickly developed and experimented with. For example, the PLAN-P program in this experiment was written in one day.

The adaptation protocol consists of two PLAN-P programs: one for the network routers and the other for the audio clients. The router program monitors the bandwidth of outgoing links and degrades the audio quality when bandwidth becomes limited. There are two advantages to performing adaptation within the routers. The first advantage is that clients on different paths in the network can receive different levels of quality depending only on the traffic on that path. The second advantage is that adaptation does not require a feedback loop and can adapt to changes immediately. The client PLAN-P program transforms degraded packets into their original format. This approach has the advantage that the audio client does not need to be changed.

We have performed two experiments using the PLAN-P adaptation protocol. These experiments demonstrate rapid adaptation within the router, and that packet delays and

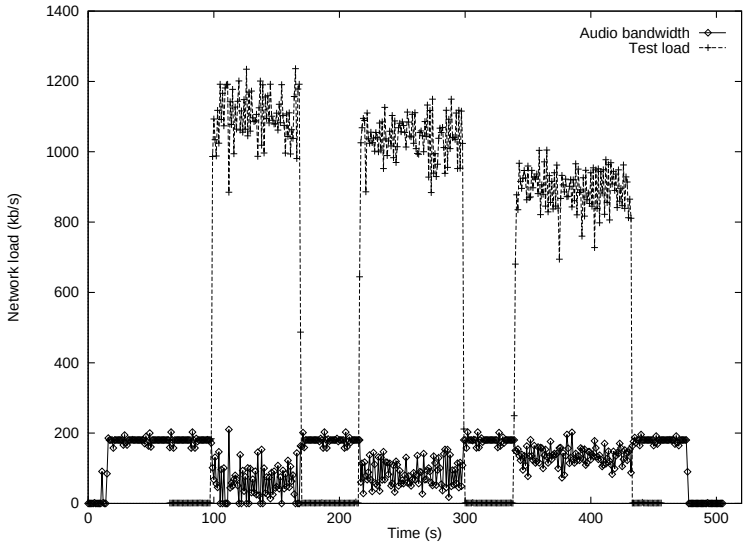


Figure 6: Audio broadcasting load

losses are reduced. Figure 5 depicts the network architecture used for the experiments. The load generator and audio client are connected to the same Ethernet segment and the load generator was used to generate a given load on the segment in order to measure the effect on the audio quality and traffic received by the client. Since adaptation is performed on an per segment basis, audio clients in IRISA may still receive high-quality audio.

Figure 6 shows the measured bandwidth used by the audio traffic for the various loads generated by the load generator. When there is no traffic on the segment, audio is sent in 16 bit stereo, requiring 176kb/s. At 100s, a large load is introduced and the protocol immediately switches to 8 bit monaural audio, requiring only 44kb/s. At 220s, a smaller load is introduced resulting in audio quality that varies between 8 and 16 bit monaural. Finally, at 340s, a small load is introduced and the audio quality is adjusted to 16 bit monaural, requiring 88kb/s. As can be seen in this figure, the adaptation is immediate since the protocol executes directly on the router, avoiding the need for software feedback.

The graphs shown in figure 7 depict the effect of the network load on the received audio signal with and without adaptation. The graphs show the number of silent periods that occur during audio playback in various configurations. This shows that the adaptation does, in fact, reduces the number of gaps in audio playback.

3.2 Extensible HTTP Server with Load-balancing Capability

Network Of Workstations (i.e., NOW) has been demonstrated to be a convenient approach for building services that can scale easily and offer better availability by tolerating failures [3,

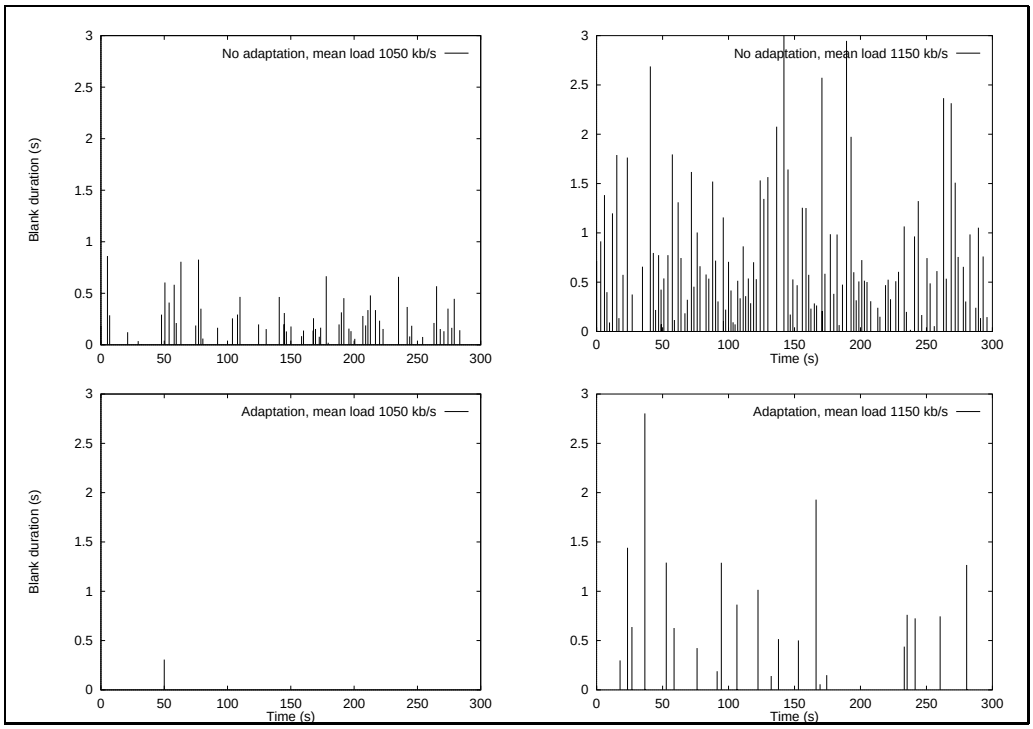


Figure 7: Sound perturbations at the client level

14]. Since balancing requests among servers relies on re-routing messages, PLAN-P offers adequate support for the programming of clustered services.

The basic scheme for implementing load-balancing with ASPs relies on a gateway that associates a physical server, determined by the load-balancing algorithm, with a request when establishing the TCP connection which supports the HTTP request; all further packets will be routed to this physical server. To do this, the gateway replaces the IP server (i.e., the virtual server) address given by the client by the address of the physical server. When sending back results to the client, the gateway replaces the IP address of the physical server by the address of the virtual server. In practice, the gateway function can be implemented either by a single ASP treating both requests and results, or by one ASP treating requests and several treating results, so as to minimize contention induced by a single gateway.

In addition to simplicity, using ASPs to build a cluster based server offers several other advantages that provide a high degree of configurability:

- **maintenance of the cluster architecture.** ASPs can be easily modified to reflect a change in the number physical servers or the topology of the cluster network. Also, an

ASP can be easily moved to any of the cluster machines, depending on administrator's need.

- **transparency of heterogeneity.** Mixing machines with different architectures and systems improves robustness of the whole service by avoiding failures on all systems at the same time. Thanks to JIT compilation, gateway ASPs can be widespreaded on servers regardless of the type of the machine.
- **evaluation of load-balancing strategies.** Different load-balancing strategies can be evaluated by changing the gateway ASP. This allows the server developer to quickly test new strategies, and is also helpful for the administrator in managing service configuration. For instance, the administrator can choose to replicate only a subpart of the web server content on all physical servers.

The main challenge for using ASPs in building clustered services is performance. Since using a gateway introduces a contention point that increases response time, packet treatment should be kept as short as possible. In order to measure the efficiency of ASPs, we have compared an ASP-based clustered HTTP server with a built-in C version.

The configuration used for our tests is the following. The cluster is made from three Sun Ultra-1 170Mhz workstations connected by a 100Mbits Ethernet network. HTTP servers are running Apache version 1.2.6 [1] with 5 to 10 child processes. The client machines are also Sun Ultra-1 170Mhz workstations connected to the cluster by a 10Mbits Ethernet network. In our experiments, we replicated the content of the IRISA web server on all three physical servers. Measurements were done by replaying a real trace of 80000 accesses, so as to minimize server cache impact on measurements. Finally, clients continuously issue requests so as to measure the maximum load the clustered server can handle.

We have tested a configuration made from two physical servers and the ASP gateway on the third cluster machine, so as to separate the cost of routing from the cost of balancing HTTP requests. The load-balancing strategy used for our experiments relies on a modulo on the number of requests. Results are presented in Figure 8. As shown by curves b and c, there is little or no difference in performance between the ASP-based version and the C-based built-in version of the load-balancing server. The ASP-based load-balancing server treats up to 85% of the load of two servers with disjoint sets of clients and 1.75 times the load of a single server. Since both servers are saturated, this shows the impact of introducing a gateway as a contention point.

3.3 Point-to-point to Multipoint MPEG Server

This experiment involves the development of a PLAN-P protocol that permits live video packets to be shared between multiple clients on the same segment using a point-to-point application. Thus, by using PLAN-P, we are able to provide multipoint video delivery with a point-to-point video server.

This protocol was implemented by extending a distributed MPEG player designed at OGI [7] with two ASPs. The first ASP executes on any one of the machines on the segment

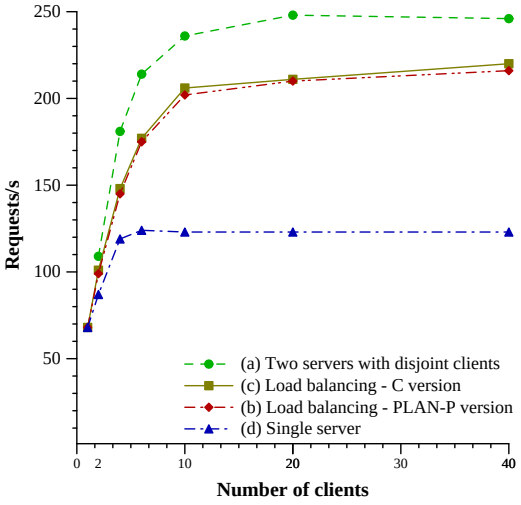


Figure 8: HTTP server performance

and maintains a list of all open connections to the video server. This ASP monitors packets sent to the server's TCP port and records the files being served, the address the video is being served to, and the setup information returned to the client for initialization. When a client makes a request, the client program first makes a request to the monitor ASP to see if the request can be filled by an existing connection. If there are no open connections, the client proceeds normally by connecting to the server, and the monitor automatically tracks the new connection. Otherwise, if an existing connection can be used, the monitor returns the IP and port address the video is being sent to, and the setup information required for decoding the MPEG stream. The second ASP executes on each client. When a client wants to receive video from an existing connection, it captures packets sent to the original address and port and delivers them to the client.

Although the video client was modified to make the additional request to the monitor ASP, the server was not changed. The client could have been reused without modification, if the monitor ASP emulated the connection responses of the server. However, since the video application uses TCP for control packets, this would be difficult to emulate in the current PLAN-P version, as the ASP would have to account for windowing, packet loss, etc. Future work on PLAN-P will be to consider extensions that would make it easier to emulate connections based on TCP.

4 Related Work

The idea that is most related to configuring distributed systems is the use of coordination languages [15]. These languages separate the specification of component behavior from component communication (connectors) and topology in a distributed system. One might consider PLAN-P to be a language for specifying the behavior of the connectors. However, current research on coordination languages considers connectors to perform communication only, and do not address the implementation of connector behaviors.

The PLAN-P language evolves directly from previous research in active networks [16, 34]. With respect to these studies, our contribution is to address a new application domain, the adaptation of distributed software components.

The PLAN-P run-time system provides the system with extensibility by permiting DSL programs to be downloaded at run time. Many other approaches to extensible operating systems have been proposed, such as SPIN [5], Vino [31] and Exokernels [12, 18]. One of the main issues that must be addressed by an extensible system is safety and security. Existing solutions rely on the use of safely typed languages, such as MODULA-3 or Java and software fault-isolation [38]; PLAN-P is also type-safe. In addition, it ensures strong properties like termination that cannot generally be verified for general purpose languages. These properties are verified using late program checking in the router (section 2.2). An alternative to the late program checking approach of PLAN-P is proof-carrying code (PCC) [25, 24]. However, in the PCC approach, the burden of proof is on the user and all systems must standardize on the properties that must be verified. In our late checking approach, properties may easily evolve with the DSL, and network providers may require different properties.

DSLs have already been used in OS design so as to permit an user to specify a certain policy. Examples are TEAPOT [8], for writing DSM coherence protocols, and HiPEC [20], for writing page-replacement policies. While HiPEC relies on a bytecode interpreter, which is less efficient than a compiler, TEAPOT is implemented with a compiler which is more difficult to maintain. In earlier work, we have proposed GAL [37], a language for designing device drivers. GAL was implemented using the same interpreter/specialization framework as PLAN-P, but specialization was not done at run time.

One particularly interesting application of PLAN-P is the use of ASPs to create adaptive systems, as in the audio broadcasting experiment of section 3.1. Odyssey [26] is one of the most recent systems for adaptation which is aimed mainly toward mobility. They define *wardens* which are application-specific components that implement adaptation for a specific data type. ASPs can been seen as DSL for wardens that can be spread over the net, not only in the client as used in Odyssey.

5 Future work

Until now, most of our efforts have been targeted toward having Tempo generating an efficient PLAN-P JIT compiler, and the design of a primitive library for adapting the ap-

plications described in the paper. The results presented in this paper open many new perspectives:

- As for the runtime system, our current and short-term plans are to enrich it with new functionalities. In particular, we are implementing a verifier which can be embedded in routers. Also, we plan to port the kernel module that runs ASPs from Solaris to Linux, so as be able to perform experiments on heterogeneous platforms. Since Tempo already supports the Intel processor, most of this work is related to OS module adaptation. We also plan to implement protocol management functionalities, such as ASP deployment.
- As for applications, our medium term goal is to do adaptation of data traffic such as images and MPEG video over low bandwidth networks. One possible solution is the integration of image distillation support [13] into PLAN-P. Also, we want to enrich the HTTP cluster server experiment with fault-tolerance capabilities and several load-balancing algorithms. This can lead to the development of a toolkit that helps the building and configuration of extensible cluster servers.
- As for the PLAN-P language, our interest is in understanding the constructs that are fundamental to adaptation and the design of ASPs. The interest in experimenting with diverse applications is to acquire knowledge of the needs. As an example, we plan to provide a better language support for TCP connections so as to simplify the task of ASP programmers.

6 Conclusion

Adapting existing distributed software components so as to reuse them in changing environments is one of the challenges of modern distributed computing. In this paper, we have proposed to perform adaptation by the means of Application-Specific Protocols that extend the standard behavior of the network. ASPs can be dynamically loaded both in routers and client/servers. This feature enables the design of original solutions for the adaptation of multipoint applications with both bandwidth preservation and a rapid reaction to environment changes. In addition, we argue that the application scope of ASPs is not only limited to quality of service adaptation; they can be used to add new functionalities to existing components. As an example, we have shown the building of an extensible HTTP server with load-balancing capabilities from standard Apache servers.

To reconcile portability and efficiency issues raised by network extensibility, we have developed a framework based on Domain-Specific Languages and automatic JIT compiler generation from a portable interpreter. From the operating system designer standpoint, this framework proved to be highly flexible, allowing the ASP run-time system to evolve easily with the application needs. Thus, we believe that our framework is a valuable approach for the design of operating components in areas with rapid evolution.

7 Acknoweldgements

The PLAN-P design framework was originally developed with Charles Consel. Pierrick Gachet helped us to setup experiments and provided technical assistance on the Solaris kernel. We wish to thank Gary Lindstrom from the University of Utah for his comments on a previous draft of this paper. We also thank Julia Lawall and Ulrik Pagh Schultz for their numerous comments and careful proof-reading.

References

[1] Apache: HTTP server project. URL: http://www.apache.org.

[2] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[3] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[4] *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, Cambridge, Massachusetts, October 1996. ACM Press.

[5] B.N. Bershad, S. Savage, P. Pardyak, E. Gün Sirer, M.E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, safety and performance in the SPIN operating system. In SOSP95 [32], pages 267–283.

[6] J-C. Bolot and A. Vega-García. Control mechanisms for packet audio in the internet. In *Proceedings of IEEE Infocom '96*, pages 232–239, San Fransisco, CA, April 1996. IEEE.

[7] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A distributed real-time MPEG video audio player. In *Proceedings of the 1995 International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pages 151–162, New Hampshire, April 1995.

[8] S. Chandra, B. Richards, and J.R. Larus. Teapot: Language support for writing memory coherence protocol. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, pages 237–248, Philadelphia, PA, May 1996. ACM SIGPLAN Notices, 31(5).

[9] C. Consel, L. Hornof, J. Lawall, R. Marlet, G. Muller, J. Noyé, S. Thibault, and N. Volanschi. Tempo: Specializing systems applications and beyond. *ACM Computing Surveys, Symposium on Partial Evaluation*, 1998. To appear.

[10] C. Consel, L. Hornof, F. Noël, J. Noyé, and E.N. Volanschi. A uniform approach for compile-time and run-time specialization. In O. Danvy, R. Glück, and P. Thiemann, editors, *Partial Evaluation, International Seminar, Dagstuhl Castle*, number 1110 in Lecture Notes in Computer Science, pages 54–72, February 1996.

[11] C. Consel and F. Noël. A general approach for run-time specialization and its application to C. In *Conference Record of the* 23^{rd} *Annual ACM SIGPLAN-SIGACT Symposium on Principles Of Programming Languages*, pages 145–156, St. Petersburg Beach, FL, USA, January 1996. ACM Press.

[12] D.R. Engler, M.F. Kaashoek, and J.W. O'Toole. Exokernel: An operating system architecture for application-level resource management. In SOSP95 [32], pages 251–266.

[13] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. In ASPLOS96 [4], pages 160–170.

[14] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In SOSP97 [33], pages 78–91.

[15] David Garlan and Daniel Le Métayer, editors. *Coordination and models, Proceedings of the second international conference, Berlin, Germany*, number 1282 in LNCS. Springer Verlag, 1997.

[16] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. Network programming using PLAN. In *Workshop on Internet Programming Languages*, Chicago, May 1998.

[17] N.D. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* International Series in Computer Science. Prentice-Hall, June 1993.

[18] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. Briceño, R. Hunt, D. Mazières, T. Pinckney, R. Grimm, J. Jannotti, and K.h Mackenzie. Application performance and flexibility on exokernel systems. In SOSP97 [33], pages 52–65.

[19] D. A. Ladd and J. C. Ramming. Programming the Web: An application-oriented language for hypermedia service programming. In *Fourth International World Wide Web Conference*, Boston, Massachusetts, December 1995.

[20] Chao Hsien Lee, Meng Chang Chen, and Ruei Chuan Chang. HiPEC: High performance external virtual memory caching. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 153–164, 1994.

[21] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 117–130, Stanford University, CA, August 1996. ACM Press.

[22] G. Muller, R. Marlet, E.N. Volanschi, C. Consel, C. Pu, and A. Goel. Fast, optimized Sun RPC using automatic program specialization. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998. IEEE Computer Society Press.

[23] G. Muller, B. Moura, F. Bellard, and C. Consel. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code. In *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, pages 1–20, Portland (Oregon), USA, June 1997. Usenix.

[24] G. Necula. Proof-carrying code. In *Conference Record of the* 24^{th} *Annual ACM SIGPLAN-SIGACT Symposium on Principles Of Programming Languages*, pages 106–116, Paris, France, January 1997. ACM Press.

[25] G. Necula and Peter Lee. Safe kernel extensions without run-time checking. In OSDI96 [27], pages 229–243.

[26] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In SOSP97 [33], pages 276–287.

[27] *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, Washington, October 1996.

[28] Internet protocol. RFC 791, September 1981. ftp://ds.internic.net/rfc/1094.txt.

[29] T. H. Romer, D. Lee, G. M. Voelker, A. Wolman, W. A. Wong, J-L. Baer, B. N. Bershad, and H. M. Levy. The structure and performance of interpreters. In ASPLOS96 [4], pages 150–159.

[30] B. Schwartz, W. Zhou, A. W. Jackson, W. T. Strayer, D. Rockwell, and C. Partridge. Smart packets for active networks. Technical report, BBN Technologies, January 1998. URL http://www.net-tech.bbn.com/smtpkts/smart.ps.gz.

[31] M. I. Seltzer, Y. Endo, C. Small, and K. A. Smith. Dealing with disaster: Surviving misbehaved kernel extensions. In OSDI96 [27], pages 213–227.

[32] *Proceedings of the 1995 ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, USA, December 1995. ACM Operating Systems Reviews, 29(5), ACM Press.

[33] *Proceedings of the 1997 ACM Symposium on Operating Systems Principles*, St-Malo, France, October 1997.

[34] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[35] S. Thibault and C. Consel. A framework of application generator design. Rapport de recherche RR-3005, INRIA, Rennes, France, December 1996. To appear in ACM SIGSOFT Symposium on Software Reusability (SSR'97).

[36] S. Thibault, C. Consel, and G. Muller. Safe and efficient active network programming. In *17th IEEE Symposium on Reliable Distributed Systems*, West Lafayette, Indiana, October 1998. To appear.

[37] S. Thibault, R. Marlet, and C. Consel. A domain-specific language for video device drivers: from design to implementation. In *Conference on Domain Specific Languages*, pages 11–26, Santa Barbara, CA, October 1997. Usenix.

[38] R. Wahbe, S. Lucco, T.E. Anderson, and S.L. Graham. Efficient software-based fault isolation. In *Proceedings of the 1993 ACM Symposium on Operating Systems Principles*, pages 203–216, Asheville, NC, USA, December 1993. ACM Operating Systems Reviews, 27(5), ACM Press.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399