Switchlets and Dynamic Virtual ATM Networks

J.E. van der Merwe and I.M. Leslie
University of Cambridge, Computer Laboratory
New Museums Site, Cambridge CB2 3QG.
Telephone: +44 1223 334650. Fax: +44 1223 334678.
email: {jev1001,iml}@cl.cam.ac.uk

Abstract

This paper presents a novel approach to the control and management of ATM networks, by allowing different control architectures to be operational within the same network, and on the same switch. The resources available on an ATM switch are divided into **switchlets**, each of which encapsulates a subset of the physical ATM switch resources. A set of switchlets on different ATM switches combine to form a virtual ATM network. Each virtual network created in this way can potentially use different control and management mechanisms which are collectively called a control architecture. For example one virtual network might run ATM Forum signalling, another might implement an IP switching control architecture, while yet another can be reserved for an in house secure control architecture. In this manner a new control architecture can be introduced into a network without disrupting existing services and applications, thereby facilitating change management in an elegant way. Switchlets and a control architecture from a well known set can be created on demand to allow the dynamic construction of virtual ATM networks of predefined type. Alternatively the control architecture can be supplied (by a 'user') after the virtual network has been created, thus allowing the dynamic creation of virtual ATM networks of arbitrary functionality. Finally, the paper presents a proof of concept implementation as well as ongoing work in this area.

Keywords

ATM, Open Switch Control, Virtual Networks, Network Control and Management, Change Management

1 INTRODUCTION

The inadequacy of existing ATM control and management strategies to meet the demands of for example multimedia applications, has been widely acknowledged. In addition to the more conventional approaches by standards bodies and communities (ATM Forum, 1993), (Cole, 1995), several other solutions to the ATM control problem have been proposed over an extended period of time. (Both (Crosby, 1995) and (Stiller, 1995), present good comparisons of different approaches to signalling and control, as well as some historical perspectives on how the standards process evolved.) More recent approaches such as (Iwata, 1995), (Lazar, 1996b), (Herbert, 1994) and (Ipsilon Networks, 1996) have proved that although there is general acceptance of the superior data transfer capability of ATM, ATM control can still not be considered a solved problem. In this paper, the mechanisms that constitute a particular control and management approach, are collectively called a **control architecture** and an instantiation of a control architecture is called a **control domain**. For example a UNI/NNI signalling implementation is an example of a control architecture, as is the IP switching architecture from Ipsilon Networks(Ipsilon Networks, 1996).

Despite showing some obvious advantages, the uptake by standards bodies of ideas from these new control architectures have been slow if at all. One of the main reasons for this situation is that a new control architecture is normally proposed as a **replacement** of an existing one. Naturally this leads to a reluctance to move to the new untested control architecture, even if the existing one is known to be non ideal. One of the major contributions of this paper is a solution to this problem by allowing different control architectures to be operational simultaneously in the same network and on the same switch.

An underlying and related problem is that of the control interface provided by the ATM switch. This is the interface used by the control architecture to manipulate the switch hardware in order to perform its control and management functions. An example of an operation performed through this interface is the manipulation of bits in a routing table in order to set up a connection through the switch. The proliferation of control architectures mentioned above means that a definite requirement for this interface is that it be **open**, so that different control architectures can be developed to make use of it.

One approach to the open switch control problem is to define a simple low level interface which can be used within any control architecture to exercise control of the physical switch. This approach, followed within the DCAN project (Herbert, 1994), has lead to the design of an open switch control interface, which is described in Section 2. The DCAN approach still has the limitation mentioned above that only one control architecture can be operational at any particular moment in time.

Section 3 addresses this problem by showing how a subset of the ATM switch resources can be presented to a particular control architecture as a **switchlet**. The term switchlet is used in favour of say, 'virtual' switch, to emphasize the fact that real resources are allocated to the switchlet. The switchlet presents the same open switch control interface to its control architecture, which means that the control architecture is oblivious of the fact that it is not in control of the whole physical switch.

Switchlets are combined into virtual ATM networks, each of which can potentially use a different control architecture, i.e. be of a different **type**. This means that switchlets permit the introduction of new control architectures into an existing network in a very elegant and controlled manner.

In the first instance the action of creating switchlets and combining them into virtual networks, is performed by human operators. The process can however be automated so that virtual networks can be **dynamically** created. This process is considered in Section 4. The control architecture instantiated on the newly created virtual network can be one of a predefined set of well known control architectures. Alternatively, an 'empty' virtual network can be created in which the control architecture is provided by the 'user' or the entity that requested the creation of the virtual network. This allows the implementation of an open multi service network (OMSN) (Van der Merwe, 1995), in which 'any user' can construct a network and become a 'network service provider'.

In Section 5, a proof of concept implementation is presented and discussed, together with some indication of ongoing work. Section 6 briefly considers related work and the paper ends with a conclusion.

2 DCAN APPROACH TO OPEN SWITCH CONTROL

Figure 1 illustrates the control of an ATM switch, by control software forming part of a particular control architecture, through an *Ariel* * **open switch control interface**. The premise of the DCAN approach is that switch control be opened up by providing a simple, generic, low level switch control interface on the switch. Switch control software running on a general purpose workstation, invokes operations on the *Ariel* interface in order to control and manage the switch.

The switch control software and the *Ariel* interface have a client-server relationship. The relationship is highly asymmetric because the server is always very simple and lightweight, while the client (the switch control software) is potentially very complex. For this reason, the control software is assumed to run on a general purpose workstation, while the *Ariel* server is simple enough that it can be implemented on very simple switches. The relationship is also one-to-one in that a single switch controller talks to a single *Ariel* interface. (Depending on implementation, the relationship could also be one-to-many, but not many-to-one.)

The switch control software is responsible for performing all the functions required by a specific control architecture, such as setting up virtual channel identifier/virtual path identifier (VCI/VPI) mappings, call acceptance control (CAC), resource allocation etc. A control architecture (or strictly speaking control domain) is not limited to the implementation of a single signalling protocol, for example an implementation capable of both UNI 3.0 (ATM Forum, 1993), and Spans (FORE Systems, 1995) signalling would still be a single control architecture.

Even though communication between the control software and the *Ariel* interface is based on client-server principles, this does not imply or require the facilities of a general purpose distributed processing environment (DPE). Rather communication between the control client and *Ariel* server is considered a 'local affair', for example on a default VCI/VPI pair. In fact the *Ariel* interface specifies the functionality required by an open switch control interface, and implementations based on different mechanisms can be (and have been) done.

2.1 The *Ariel* Interface

The purpose of the *Ariel* interface is to provide an open, generic switch control interface with a useful set of functions. In particular the *Ariel* interface should be useful even if detailed knowledge of the controlled switch is not available. In the ideal case *Ariel* should provide sensible control of a switch of unknown type.

The *Ariel* control interface consists of the following interfaces:

Configuration - the configuration interface is primarily used to **find out** what the configuration of a switch is as opposed to configuring the switch.

Port - a port interface is provided for each port on the switch and deals with a port as a complete entity, e.g. port up/down.

Connections - the connections interface is responsible for basic VCI/VPI mapping, and deals with quality of service (QoS) issues through a context index obtained (by the controller) through the context interface.

Context - the context interface bundles QoS abstractions into a single interface and is explained below in more detail.

Statistics - the statistics interface allows the controller to obtain switch statistics and accounting information.

---

*Credit is due to Sean Rooney of the Computer Lab for suggesting Shakespeare's *The Tempest* as the basis of our name space.

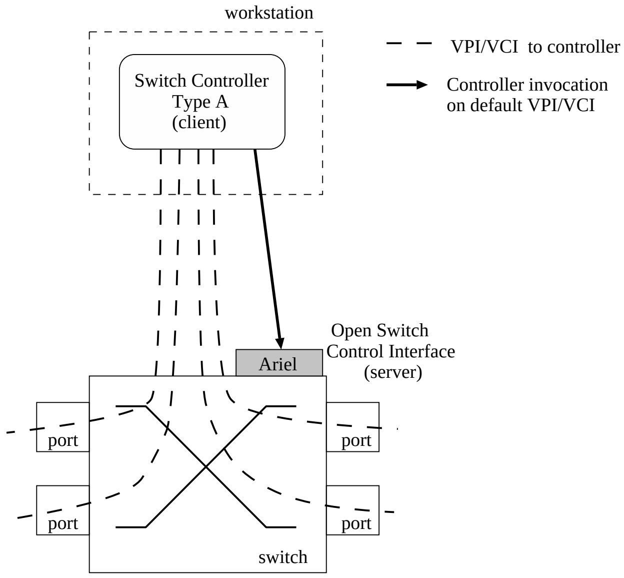

Figure 1 Switch control through the *Ariel* interface

Alarms - the alarms interface allows the controller to be informed when certain events take place on the switch.

QoS aware connections are set up through the *Ariel* interfaces, by first creating a context, and then associating that context with the VPI/VCI mapping during the actual connection setup. This means that all QoS issues are effectively taken out of the Connections interface, which can be kept simple.

Another reason for separating the Connections and Context interfaces, comes from the realization that the Context interface provides a way to 'allocate resources' on the switch, and the Connections interface provides a way to 'use the resources', and that the two need not be tightly coupled. For example, a control architecture can allocate a certain amount of resources on the switch by means of the Context interface, and then 'over commit' these resources by allowing more connections to be created than the resources would suggest, because it has some external knowledge about the behaviour of the connections in question. Note that most current commercial ATM switches will not allow this separation between the allocation and usage of resources.

Dealing with QoS issues in a generic fashion at a low level is very difficult, and may not be possible. The reason for this is that the QoS **capabilities** of a switch are determined by the queueing and scheduling policies employed. These, in turn, are what differentiate one switch from another. It is therefore unlikely that all switch vendors would be willing to make such details about their switches available to be included in an **open** interface, such as *Ariel*.

One way of avoiding this problem is to hide the switch queueing and scheduling policies behind a generic interface. The ATM service categories identified by the ATM Forum (Sathaye, 1995), provide the means for such an abstraction. This approach is reasonable because it can be expected that switch manufacturers will build switches with queueing and scheduling mechanisms which will support a subset of these services. This approach is followed in *Ariel*. It must be noted that should lower level details about switch internals be made available by certain switch vendors, this knowledge can still be used in favour of the abstraction based on ATM Forum service categories.

Five service categories are (currently) defined by the ATM Forum, namely:

UBR - unspecified bit rate
CBR - constant bit rate
rtVBR - real time variable bit rate
nrtVBR - non real time variable bit rate
ABR - available bit rate

These service categories are parametrised by (in total) four QoS parameters and six traffic parameters, a subset of which must to be specified for each service category. Hiding the switch details behind this abstraction obviously leads to a loss of information. However, enough information is still available to perform functions such as call acceptance control (CAC) outside the switch in the controlling workstation.

All that is required to perform CAC outside the switch is the **resource mapping function** that is used by the switch to map QoS and traffic parameters to switch resources, as well as knowledge about the resources available on the switch. Such a resource mapping function constitutes significantly less sensitive information than the mechanisms used to implement it. It can therefore be expected that switch manufacturers will be more willing to provide such information. If switch specific CAC functions are not available, generic CAC functions could be used by control software. As long as the generic CAC functions err on the conservative side, this will result in useful (albeit not ideal) switch control.

3 SWITCHLETS

The approach described in Section 2 allows a switch to be controlled in an open fashion and allows different control architectures to be designed to utilise the *Ariel* interface. This allows much more openness and flexibility than is currently the case, but it still means that at any given time, a single control architecture is operational on a switch and within a network.

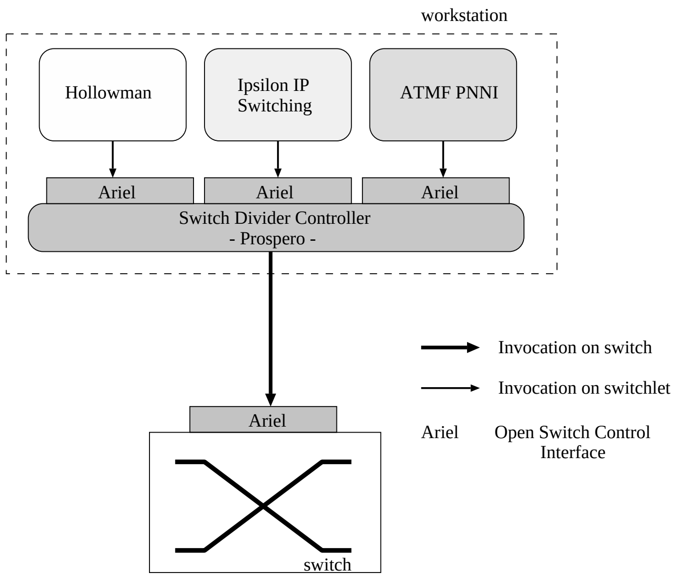

Figure 2 Creating switchlets

Figure 2 shows how the *Ariel* interface on a physical switch can be used by a **Switch Divider Controller** to create several **switchlets**. (In keeping with our name space the Switch Divider Controller is called *Prospero*.) The *Prospero* Switch Divider Controller allocates a subset of the physical switch resources into a switchlet, and makes this available to switch control software through an *Ariel* interface. Switch control software, of a particular type, will control the switchlet by invocations on the switchlet *Ariel* interface, in exactly the same way as it would on a physical switch. As an example, Figure 2 shows three possible control architectures namely Hollowman (Rooney, 1997), IP Switching (Ipsilon Networks, 1996) and the ATM Forum's PNNI (ATM Forum, 1996).

Switchlets can be combined into virtual networks of a certain type, or control architecture. This is depicted in Figure 3, which shows a network of five switches, on which three virtual networks of different type are deployed.

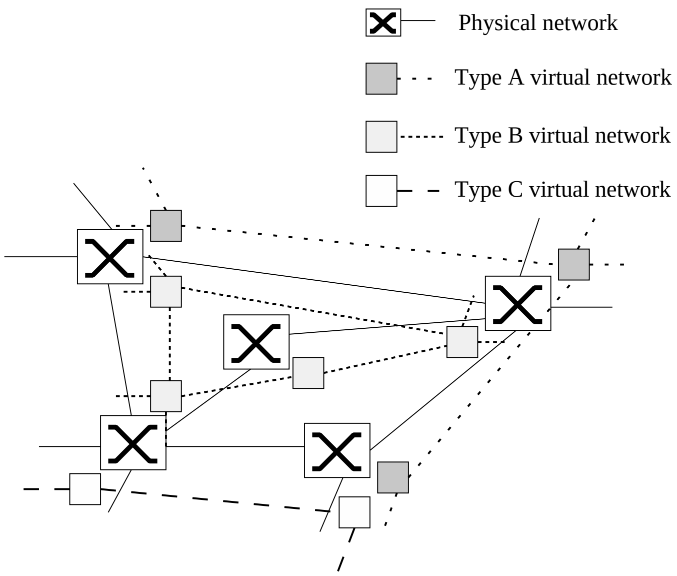

Figure 3 Virtual ATM networks with different control architectures

The switch is completely oblivious of the fact that several control architectures are operational on it, and maintains the one-to-one relationship between an *Ariel* server on the switch and a single Switch Divider Controller. The *Prospero* Switch Divider Controller provides *Ariel* interfaces to the switchlet controllers. Except for the fact that less resources are available to it, the switch(let) controllers are therefore also unaware of the presence of the Switch Divider Controller. The Switch Divider Controller polices invocations on the switchlet *Ariel* interfaces to ensure that switchlet controllers do not utilise resources not allocated to it, or in any other way interfere with the functioning of other switchlets.

The switch divider controller is analogous to the small kernel in the Nemesis operating system (Leslie, 1996), which is responsible (amongst other things) for allocating system resources to scheduling domains, and for policing the actual usage of allocated resources. Indeed the proposed model has some similarities with the Nemesis operating system in that real resources are made available to an 'application', the control architecture in the network model, which is then allowed to use these resources according to some internal policy.

3.1 The *Prospero* Switch Divider Controller

Partitioning of switches into switchlets require the **specification** of a switchlet in terms of a subset of the physical resources available on the switch. Resources on a switch that need consideration include: ports, VPI/VCI space, bandwidth, buffer space, and queueing and scheduling policies.

Ports and VPI/VCI space constitute the connections resources of a switch and can be partitioned at various levels of granularity:

port level - whereby certain ports within a switch are allocated to a switchlet

VPI level - whereby certain VPI ranges on certain ports are allocated to a switchlet

VCI level - whereby certain VCI ranges on certain VPIs on certain ports are allocated to a switchlet

Partitioning at the VCI level, being the most general of the three possibilities, is the approach taken in specifying switchlets.

Bandwidth, buffer space, and queueing and scheduling policies combine to represent the **switching capacity** of the switch. As explained in Section 2.1, the approach taken with the *Ariel* interface is to hide QoS details behind the five ATM service categories. The same approach is followed in specifying switchlets, whereby a certain percentage of the resources for a particular service category will be 'marked' as belonging to a certain switchlet. The control architecture operating on the switchlet can then employ the same resource mapping function mentioned in 2.1, on its subset of resources to perform for example CAC.

A switchlet specification could therefore consist of the number of ports required, and then for each port the following information:

- The range of VPIs required
- The range of VCIs per VPI required
- The service categories required
- The capacity per service category required (Until a better understanding of the problem of dividing switch resources have been developed, service capacity will be specified as a percentage of what is available on the physical switch.)

The *Prospero* Switch Divider Controller has to **know** the capacity of the physical switch, and only allow switchlets to be created until this capacity is exhausted. Allocating resources on the switch to a switchlet does not involve any invocations (or allocations) on the physical switch. Rather, *Prospero* notes the allocation in its internal representation of the switch capacity, and uses that to police future invocations on a switchlet *Ariel* interface. Once connections have been established in a switchlet (and switch), *Prospero* has to rely on in-band policing mechanisms in the physical switch, to ensure that connections from one switchlet do not adversely affect that of other switchlets. This requires nothing new on the physical switch, since this functionality is needed in switches any way.

In the first instance *Prospero* provides a configuration interface, which can be used by human operators to create switchlets and virtual networks. Of more interest though, is the dynamic creation of virtual networks (on demand) by other software systems. Such a **Virtual Network Service** is considered in the next section.

Remote access to the *Prospero* interfaces are required for both static (i.e. done by a network administrator on a long time scale), or dynamic (i.e. done by software on an on demand basis) virtual network creation. Therefore both of these actions presuppose the existence of a **bootstrapping (virtual) network**, implementing a **bootstrapping control domain**. The bootstrap control domain is therefore any control architecture, contained within its own virtual network, which provides the required addressing, routing and other facilities to enable communication between the *Prospero* instances and other software entities.

In the prototype implementation, an existing IP-over-ATM (virtual) network is used as the bootstrap control domain. This solution has allowed progress to be made, but is considered too heavyweight because of its reliance on conventional ATM control. On the other hand the use of IP in the bootstrap network has attractive properties, and an alternative implementation based on the much simpler IP switching mechanisms (Ipsilon Networks, 1996) is therefore currently under investigation. Note that the bootstrapping problem is present in all ATM (and other) networks, and is not unique to the environment described in this paper. The bootstrapping facility has however been generalised into something that can provide more sophisticated services.

4 VIRTUAL NETWORK SERVICES

As mentioned in the previous section, sets of switchlets can be combined into virtual networks. The control domains for these different virtual networks could be instantiations of the same control architecture, or a different control architecture could be operational in each virtual network, or any combination of these.

The ability to have different control architectures in the same physical network allows for a very elegant way of introducing new control software into an existing operational network. The new control architecture can namely be made operational in its own virtual network while existing users and applications operate undisturbed in the original (virtual) network. After the new control architecture has been tested, users and applications can be migrated to it and the partitioning of switchlet resources can be modified to reflect the fact that the new virtual network is to become the default (or only) virtual network.

A more interesting possibility is to provide an on demand virtual network service, in which switchlets are dynamically created and merged into virtual networks. If this facility is combined with the services provided by a distributed processing environment (DPE), virtual networks become a service which can be offered, traded and manipulated like any other service. Such a DPE can be one of the facilities provided within the bootstrap virtual network, and is assumed in the following discussion.

The use of a DPE (in the bootstrap virtual network) does not mean that all control architectures have to be implemented by means of a DPE, or even be aware of the existence of a DPE. Indeed, a major strength of the approach presented here is that a conventional control architecture (e.g. an ATM Forum UNI/NNI compliant control architecture) can be instantiated in and confined to its own virtual network. On the other hand, new control architectures are being developed that can make use of the DPE facilities or can even be implemented in a DPE environment. The latter approach might lead to some simplifications in the control architecture. For example, because of the existence of the bootstrap virtual network it might not be necessary for a particular control architecture to implement its own bootstrapping procedures but instead rely on the bootstrap virtual network to provide such services.

4.1 Creating a Virtual Network

This section describes the system services required for the dynamic on demand creation of virtual networks. Figure 4 shows the interaction between the services that are involved in this process.

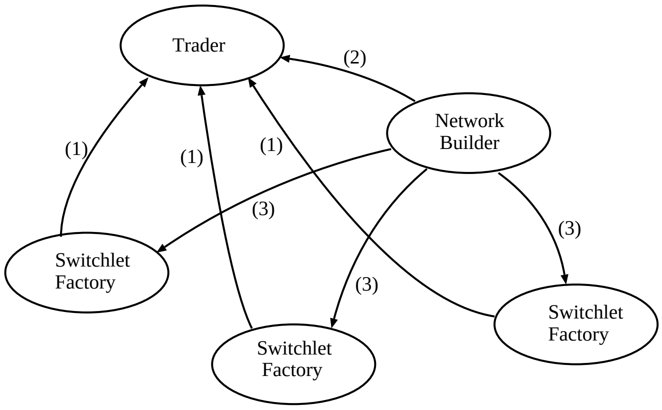

Figure 4 Dynamic Virtual Network Services

The **Switchlet Factory** service encapsulates the *Prospero* Switchlet Divider Controller presented in Section 3. The Switchlet Factory informs a **Trader** service about its existence as well as its switch capacity. (Interaction (1) in the Figure.) Trading is the standard way of matching service providers and consumers in a DPE (APM, 1993). The Switchlet Factory also exports interfaces which allow the creation and destruction of switchlets. When a switchlet is created the Switchlet Factory also exports an *Ariel* switch control interface.

In order to create a virtual network, a **Network Builder** service is provided with the 'specification' of the desired network. This network specification is in terms that hitherto made little sense after network equipment have been commissioned. An example specification could be,

- UNI 3.1 signalling
- CBR capacity of 15 Mbps (Note that in the case of CBR a percentage of what is available is trivially converted to bandwidth and vice versa.)
- Between A and B
- With redundancy
- For three hours

Or could be as simple as 'Cheap network between A and B'.

The network specification could be the output of another service, or be provided by a human being.

The Network Builder has knowledge about existing virtual networks, and coordinates the creation of virtual networks so that, for example, the VCI space allocated to two switchlets in the same virtual network, and in adjacent switches will overlap. The Network Builder contacts the Trader and asks for all switches with the required capabilities and capacity according to the supplied network specification. The result of the query, invocation (2) in Figure 4, is a list of 'interface references' to Switchlet Factories matching the criteria. Using the supplied network specification and topology information obtained from some topology service, the Network Builder determines which switches should form part of the virtual network. The Network Builder then invokes required operations on appropriate Switchlet Factories to create the switchlets.

Two possibilities exist in terms of the type of the control architecture which will be instantiated on a newly created virtual network:

A predefined control architecture from a well known set can be started up when the virtual network is created. An example would be the creation of an ATM Forum UNI/NNI compliant virtual network. In DPE terminology this would be called a traded typed virtual network. In this case the appropriate software entities will be started up by the network builder as soon as the virtual network has been created.

Alternatively, a blank virtual network or virtual network without a control architecture can be created. In this case the control architecture is supplied or filled in by the entity that requested its creation. This would be called an 'anytype' virtual network in DPE terms. In this case the network builder will not start up the control architecture, but rather will return an interface reference to each switchlet to the entity that requested creation of the virtual network. Since this involves the services of the DPE in the bootstrap control domain, these control architectures will normally be required to make use of the DPE. In this manner a new control architecture can be **composed** out of **base components**. This means that by adding or modifying base components virtual networks with control domains of arbitrary complexity and functionality can be constructed. For example, the composed control architecture can use all base components but replace the routing component with a special purpose one for its particular application.

The newly created virtual network then proceeds to perform its own initialisation, bootstrapping and operation, all of which is confined to its 'own' switchlets and control domain. Note that in the case of a conventional control architecture, with its own bootstrap procedure, the operations of the control architecture can be truly confined to its own virtual network. This is not true if the control architecture use facilities provided by the bootstrap virtual network, or rely on the DPE in the bootstrap virtual network for its communication. This is an important issue in terms of the resources, both network and processing, which have to be allocated to the bootstrap virtual network.

Following the creation of a switchlet, the Switchlet Factory has to update its available capacity in the Trader, or potentially remove its trader entry if it has no capacity left. The decomposition of a virtual network happens when the requested time period expires and the Switchlet Factory claims back switchlet resources, and updates its Trader entry. Alternatively, if an undefined time period is required, the Network Builder will be responsible for periodic 'keep alive' invocations on the Switchlet Factory to keep the virtual network intact.

Something that has not been considered in the above discussion is how potential users of the new virtual network get to know about its existence. It is assumed that creation of the virtual network is the result of a request of potential users, or that potential users will be able to obtain this information through some external mechanism.

5 PROOF OF CONCEPT IMPLEMENTATION

In order to proof the feasibility of the Switchlet and Dynamic Virtual Network concepts, an implementation has been done on one of the ATM networks at the Cambridge University Computer Laboratory. The only switch resources that were taken into account for this implementation were switch ports and VPI/VCI space. (i.e. none of the capacity and QoS resources mentioned in Section 3.1 were considered.)

The part of the service ATM network used consists of three Fore Systems ATM switches (one ASX-100 and two ASX-200s), which interconnect a number of file servers, workstations, one router and several ATM video adapters (Fore Systems AVA-200s). These switches are an essential part of the Computer Laboratory's infrastructure, and as such minimal disruption and downtime were of key importance. The approach taken was therefore to utilise the existing IP-over-ATM network provided by means of Spans and UNI Signalling in the Fore Systems Switches as the bootstrap control domain. The VPI/VCI space available to this bootstrap control domain was however limited, which means the remainder of VPI/VCI space was made available to a *Prospero* Switch Divider Controller implementation. This arrangement is illustrated in fig 5.

The communication mechanism between *Prospero* and the switch depends on the switch model. Specifically, in the case of the ASX-100, an *Ariel* server implementation runs on the switch. For comparison purposes, several *Ariel* impersonations using different mechanisms were implemented. This included both message passing as well as remote procedure call (RPC) based implementations. For the two ASX-200 switches, invocations are made by means of SNMP (over IP provided by the bootstrap control domain) with a SNMP daemon running on the switch. Communication with the ASX-100 can also use the SNMP mechanism, however the *Ariel* implementation is more efficient and architecturally cleaner, and is therefore to be preferred.

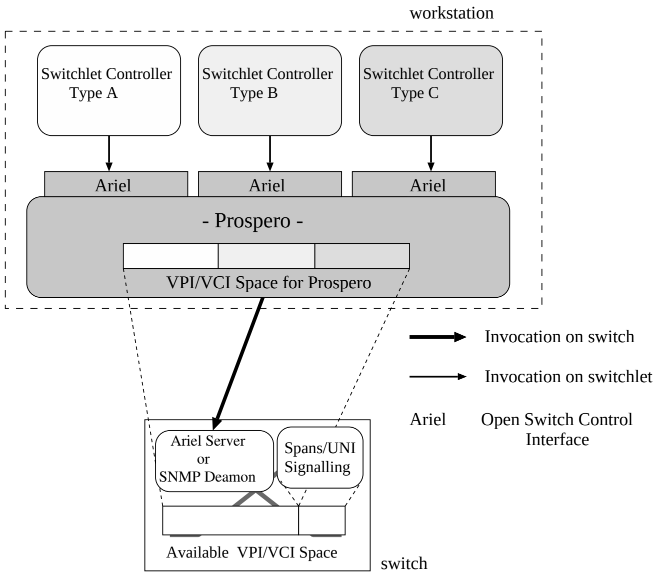

Figure 5 Proof of concept *Prospero* implementation

The Dynamic Virtual Networking Services were implemented using an implementation of the Distributed Interactive Multimedia Architecture (DIMMA) to provide its DPE (Li, 1995). DIMMA is a framework Object Request Broker (ORB), which provides a common base for the construction of domain specific ORBs.

As mentioned in Section 4 the network builder keeps track of all current virtual networks in the system. The network builder is also the only entity which is allowed to create and destroy switchlets. This allows the network builder to perform garbage collection on virtual networks for which the control architecture malfunctions. Because users are allowed to supply 'their own' control architectures, this is an important function to ensure the integrity of the network as a whole. The current implementation also provides a trivial access control mechanism, whereby a list is kept of users who are currently allowed to request creation of virtual networks. In similar fashion, the current implementation obtains topology information from a manually constructed topology database.

The implementation described in the above paragraph clearly has some limitations. In particular the use of SNMP in some cases to talk to the switch is suspect from a performance point of view. More seriously however from an architectural point of view, is the fact that the Spans/UNI signalling is still running on the switch and is not using the *Ariel* interface. The security provided by the trivial access control mechanism described above, while adequate for an experimental environment, will also need to be significantly extended in a real world implementation. However, the fact that the implementation has been done on an existing service network with minimal disruption is testimony to the flexibility of the switchlet concept.

The virtual network environment described above has been used (and is still being used) in the Computer Laboratory to implement several novel control architectures (Rooney, 1997), (Van der Merwe, 1997). Work is currently being undertaken to implement more conventional control architectures in this environment.

6 RELATED WORK

A speculative application programmers interface (API) for the control of ATM switches was presented in (Lazar, 1996a). This API seems to provide similar functionality to the *Ariel* interface presented in Section 2.1 but contains no interface for obtaining switch statistics or for receiving alarms. The generic switch management protocol (GSMP) (Newman, 1996) is another low level switch control mechanism. GSMP closely match the functionality provided by *Ariel*, but has a much simpler notion of QoS. In fact, since no QoS issues were taken into account for the current implementation, GSMP is one of the *Ariel* impersonations which the current *Prospero* implementation can use.

Virtual ATM networks based on virtual paths (VPs) have been proposed as a mechanism to segregate different traffic types into more homogeneous (and thus more easily manageable) groups (Fotedar, 1995), (Farago, 1995). In (Gupta, 1995) the partitioning of resources to form virtual networks in a packet network is presented. In this case the purpose of the resulting virtual networks is to reduce connection setup times for 'real-time' connections.

In the Xbind project (Lazar, 1996b) networking devices such as switches are abstracted into virtual entities which export DPE interfaces. Since different algorithms are allowed to operate on these high level abstractions, some notion of virtual networking is possible within the Xbind environment. However, since it is impossible to reduce a control architecture, such as for example a PNNI implementation (ATM Forum, 1996), to a mere set of algorithms, this provides a fairly limited virtual networking environment.

7 CONCLUSION

The paper presented the concept of ATM **switchlets** whereby a subset of physical ATM switches are presented to a control architecture to manipulate as it sees fit. The switchlets are presented to the control architecture as an *Ariel* open switch control interface. The *Prospero* **switch divider**

controller, controls the physical ATM switch by means of an *Ariel* interface on the physical switch, and polices invocations made by the different control architectures operational on the switch.

Since the partitioning of switch resources is done at a very low level, very few restrictions are being imposed on control architectures implemented in the switchlet environment. This allows both conventional control architectures based on message passing protocols, as well as control architectures implemented using DPE methodologies to be accommodated.

It was shown how the switchlet concept can be used to introduce new control architectures into an existing network in a flexible and non disruptive manner. Indeed this has been proven by means of a proof of concept implementation.

Finally, the paper showed how the switchlet concept can be used to create **virtual ATM networks** of arbitrary topology on demand and to run arbitrary control architectures in these virtual networks. This means that virtual networks becomes a service which can be offered, and traded like any other service in a DPE environment. The control architecture instantiated in these virtual networks can be known a priori or can be supplied by the entity requesting network creation. This allows users to supply and manipulate their own control architecture in the created virtual network.

8 REFERENCES

APM Limited (1993) The ANSA Model for Trading and Federation. Tech. Rep. AR.005.00, APM Limited, Castle Park, Cambridge, UK.

ATM Forum (1993) *ATM User-Network Interface Specification - Version 3.0.* Prentice Hall.

ATM Forum (1996) Private Network-Network Interface Specification Version 1.0 (PNNI 1.0). ATM Forum document: af-pnni-0055.000 .

Cole, R.G., Shur, D.H. and Villamizar, C. (1995) Ip over atm: A framework document. Available from: http://ietf.cnri.reston.va.us/.

Crosby, S.A. (1995) Performance Management in ATM Networks. Tech. Rep. 393, University of Cambridge, Computer Laboratory, UK.

Farago, A. et al (1995) A New Degree of Freedom in ATM Network Dimensioning: Optimizing the Logical Configuration. *IEEE Journal on Selected Areas in Communication*, vol. 13, pp. 1199–1206.

FORE Systems (1995) SPANS UNI: Simple Protocol for ATM Signalling. Release 3.0. FORE Systems Inc., 174 Thorn Hill Rd, Warrendale PA, USA.

Fotedar, S. et al (1995) ATM Virtual Private Networks. *Communications of the ACM*, vol. 38, pp. 101–109.

Gupta, A. and Ferrari, D. (1995) Resource Partitioning for Real-Time Communication. *IEEE/ACM Transactions on Networking*, vol. 3, pp. 501–508.

Herbert, A. et al (1994) Scalable Distributed Control of ATM Networks. Project proposal, University of Cambridge, Computer Laboratory, UK. Project overview available from: http://www.ansa.co.uk/DCAN/index.html.

Ipsilon Networks (1996) IP Switching: The Intelligence of Routing, the Performance of Switching. Available from: http://www.ipsilon.com/productinfo/techwp1.html.

Iwata, A. et al (1995) ATM Connection and Traffic Management Schemes for Multimedia Interworking. *Communications of the ACM*, vol. 38, pp. 72–89.

Lazar, A.A. and Marconcini, F. (1996a) Towards an Open API for ATM Switch Control. Available from: http://www.ctr.columbia.edu/comet/xbind/xbind.html.

Lazar, A.A. et al (1996b) Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture. *IEEE Journal on Selected Areas in Communication*, vol. 14, pp. 1214–1227.

Leslie, I.M. et al (1996) The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communication*, vol. 14, pp. 1280–1297.

Li, G. (1995) DIMMA Nucleus Design. Tech. Rep. APM.1551.00.05, APM Limited, Castle Park, Cambridge, UK.

Newman, P. et al (1996) Ipsilon's General Switch Management Protocol Specification Version 1.1. *Internet RFC1987*.

Rooney, S. (1997) An Innovative Control Architecture for ATM Networks. IM'97, San Diego.

Sathaye, S.S. (1995) ATM Forum Traffic Management Specification Version 4.0. ATM Forum Technical Committee - Contribution 95-0013.

Stiller, B. (1995) A survey of UNI Signalling Systems and Protocols for ATM Networks. *ACM Computer Communications Review*, vol. 25, pp. 21–33.

Van der Merwe, J.E. and Chuang S.C. (1995) Support for Open Multi Service Networks. Regional International Teletraffic Seminar, Pretoria, South Africa. Available from: http://www.cl.cam.ac.uk/users/jev1001/.

Van der Merwe, J.E. and Leslie, I.M. (1997) Service Specific Control Architectures for ATM. In preparation.

9 BIOGRAPHY

Kobus van der Merwe received the B.Eng. and M.Eng. degrees from the University of Pretoria in 1989 and 1991 respectively, and is currently working towards the Ph.D. degree at the University of Cambridge Computer Laboratory, Cambridge, U.K. His current research interests are in network control and management.

Ian Leslie received the B.A.Sc in 1977 and M.A.Sc in 1979 both from the University of Toronto and a Ph.D. from the University of Cambridge Computer Laboratory in 1983. Since then he has been a University Lecturer at the Cambridge Computer Laboratory. His current research interests are in ATM network performance, network control, and operating systems which support guarantees to applications.