CISCO/MOTOROLA PROPOSAL

Using a Real-Time Embedded ORB for ATM Switch Control and Management

Points of Contact:

Technical Matters:
Douglas C. Schmidt
Associate Professor & Director of the Center for Distributed Object Computing
Department of Computer Science
Washington University in St. Louis
TEL (314) 935-4215
FAX (314) 935-7302
EMAIL schmidt@cs.wustl.edu

Administrative Matters:
Fred Kuhns
Senior Research Associate
Department of Computer Science,
Washington University in St. Louis
TEL (314) 935-6598
FAX (314) 935-7302
EMAIL schmidt@cs.wustl.edu

A Statement of Work

The scope of this effort is to create a real-time ORB that's compliant with the new OMG Minimum CORBA [8] specification and embed this ORB in a Cisco ATM switch. This embedded ORB will leverage the advanced real-time features [15, 14] developed by Washington University's Center for Distributed Object Computing to implement switch management protocols efficiently, in particular, the Virtual Switch Interface (VSI) protocol [17].

This proposal describes the specific tasks to be performed during the 12 months of the proposed project.

A.1 Introduction

During the past decade, there has been substantial R&D emphasis on *high-speed networking* and *performance optimizations* for network elements and protocols. This effort has paid off such that networking products are now available off-the-shelf that can support Gbps on every port, *e.g.*, Gigabit Ethernet and ATM switches. Moreover, 622 Mbps ATM connectivity in WAN backbones is starting to appear. In networks and GigaPoPs being deployed for the Next Generation Internet (NGI), such as the Advanced Technology Demonstration Network (ATDnet) [1] 2.4 Gbps (OC-48) link speeds are being deployed. However, the general lack of robust and flexible tools and middleware for programming, provisioning, and controlling these networks has limited the rate at which NGI applications have been developed to leverage advances in high-speed networks.

What is required is a high-performance, real-time and QoS aware communications middleware embedded in network elements, such as ATM switches and IP routers. This embedded ORB provides a uniform access interface to network controls, services, and resources to support control plane and management plane activities.

The ACE ORB (TAO): The **TAO** CORBA 2.3-compliant Object Request Broker (ORB) is developed at Washington University's Center for Distributed Object Computing (DOC) [2]. TAO is an open-source, standards-based, high-performance, real-time ORB endsystem that supports applications with deterministic and statistical QoS requirements, as well as "best-effort" requirements, and is the first ORB to support end-to-end QoS guarantees over ATM/IP networks [13]. TAO's features and optimizations include an ORB Core that minimizes context switching, synchronization, dynamic memory allocation, and data movement [16]; a highly-scalable Object Adapter that demultiplexes requests in constant-time [14]; an optimizing IDL compiler [4]; real-time I/O subsystem [6], and a global resource allocation and scheduling framework [15].

CORBA Protocol model synopsis: CORBA Inter-ORB Protocols (IOP)s define interoperability between ORB endsystems. IOPs provide data representation formats and ORB messaging protocol specifications that can be mapped onto standard and/or customized transport protocols. Regardless of the choice of ORB messaging or transport protocol, a standard programming model is exposed to the CORBA applications. Figure 1 shows the relationships between these various components and layers.

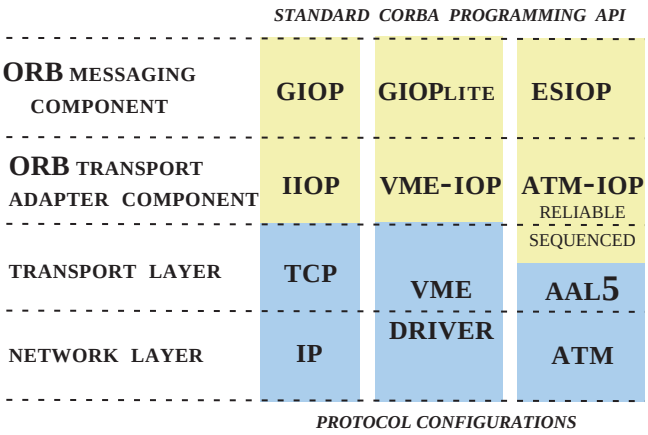

Figure 1: Relationship Between Inter-ORB Protocols and Transport-specific Mappings

In the CORBA protocol interoperability architecture a standard General Inter-ORB Protocol (GIOP) is defined by the CORBA specification [10]. The CORBA specification also defines a transport-specific mapping of GIOP onto the TCP/IP protocol suite. This mapping is called the Internet Inter-ORB Protocol (IIOP) and is required for an ORB implementation to be considered "interoperability compliant." Other mappings of GIOP onto different transport protocols are allowed by the specification, as are different inter-ORB protocols, known as Environment Specific Inter-ORB Protocols (ESIOP)s.

Regardless of whether GIOP or an ESIOP is used, a CORBA IOP must define a data representation, an ORB message format, an ORB transport protocol or transport protocol adapter, and an object addressing format.

Pluggable protocol framework: Within the scope of the CORBA interoperability architecture, ORB developers are free to optimize internal data structures and algorithms [14]. Moreover, ORBs may use specialized inter-ORB protocols and ORB services and still comply with the specification.[1]

We have leveraged this aspect of the standard and developed a *pluggable protocol framework* within TAO. A key feature of the framework's design is its decoupling of ORB messaging and transport interfaces from its transport-specific protocol components. Figure 2 shows the partitioning of responsibilities for pluggable protocols and how it relates to other ORB services. This new framework is transparent to application de-

[1] An ORB *must* implement GIOP/IIOP, however, to be interoperability-compliant.

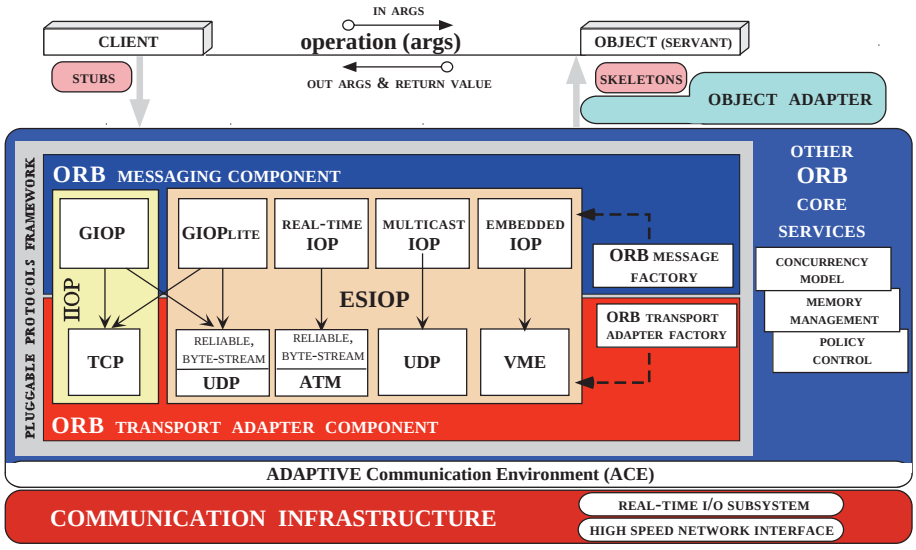

Figure 2: TAO's Pluggable Protocols Framework Architecture

velopers, since protocols can be (re)configured without modifying the standard CORBA programming API.

This design allows custom ORB messaging and transport protocols to be configured flexibly and used transparently by CORBA applications. For example, if ORBs communicate over a high-speed networking protocol with QoS support like ATM, then simpler, optimized ORB messaging and transport protocols can be configured to eliminate unnecessary features and overhead of the standard CORBA General Inter-ORB Protocol (GIOP) and Internet Inter-ORB Protocol (IIOP). Likewise, TAO's pluggable protocols framework makes it straightforward to support customized embedded system interconnects, such as CompactPCI or VMEBus, under standard CORBA inter-ORB protocols like GIOP.

TAO's pluggable protocols framework supports the creation of efficient, high performance inter-ORB *in-line bridges*. An in-line bridge converts inter-ORB messages or requests from one type of Inter-ORB protocol to another. This feature makes it possible to efficiently bridge disparate ORB domains without incurring unnecessary context switching, synchronization, or data movement. An interesting side benefit of this feature is the ability to "plugin" a new ORB messaging protocol, such as the Virtual Switch Interface (VSI) [17] or the General Switch Management Protocol (GSMP) [12, 11].

The Virtual Switch Interface (VSI): VSI is a protocol that allows multiple independent signaling processors to interact with an ATM switch control processor to (1) add and delete connections on the switch, (2) automatically discover switch resources, and (3) collect statistics. Key features of VSI are (1) its support for distributed processing (e.g., multiple signaling processors controlling a single ATM switch, as well as one signaling processor controlling an ATM switch that contains multiple control processors), (2) flexible support for QoS, (3) windowing flow control between master controller and slave controller, and (4) fault tolerance support.

VSI is gaining acceptance as a vendor-neutral standard for controlling ATM switches and is the basis of a standardization effort in the Multiservice Switch Forum (MSF) [7]. Given Washington University's pioneering research in ATM switch and network design, real-time ORB middleware, and high-speed network management, we recognize the importance of open signaling and a standard switch management protocol, such as VSI, to further the development and deployment of large-scale ATM-based distributed systems.

When using VSI as the inter-ORB messaging protocol, the TAO pluggable protocols framework can exchange VSI messages with a VSI-enabled network element that does not contain an ORB. Thus, using VSI obviates the need for creating proxy objects to translate CORBA method invocation into VSI control messages.

In our project described in this proposal, we will combine (1) our knowledge of ATM switching and ATM signaling with (2) our expertise in ORB middleware technologies to conduct a 12 month research program that synergistically couples VSI and CORBA to create an OO VSI-based switch control framework called VSI++, which is described below.

A.2 Proposal: Develop a Real-Time Embedded ORB for ATM Switch Control and Management

In our proposed effort we will create a version of TAO that conforms to the forthcoming Real-time CORBA [9] and Minimum CORBA [8] standards so it can be embedded in a Cisco

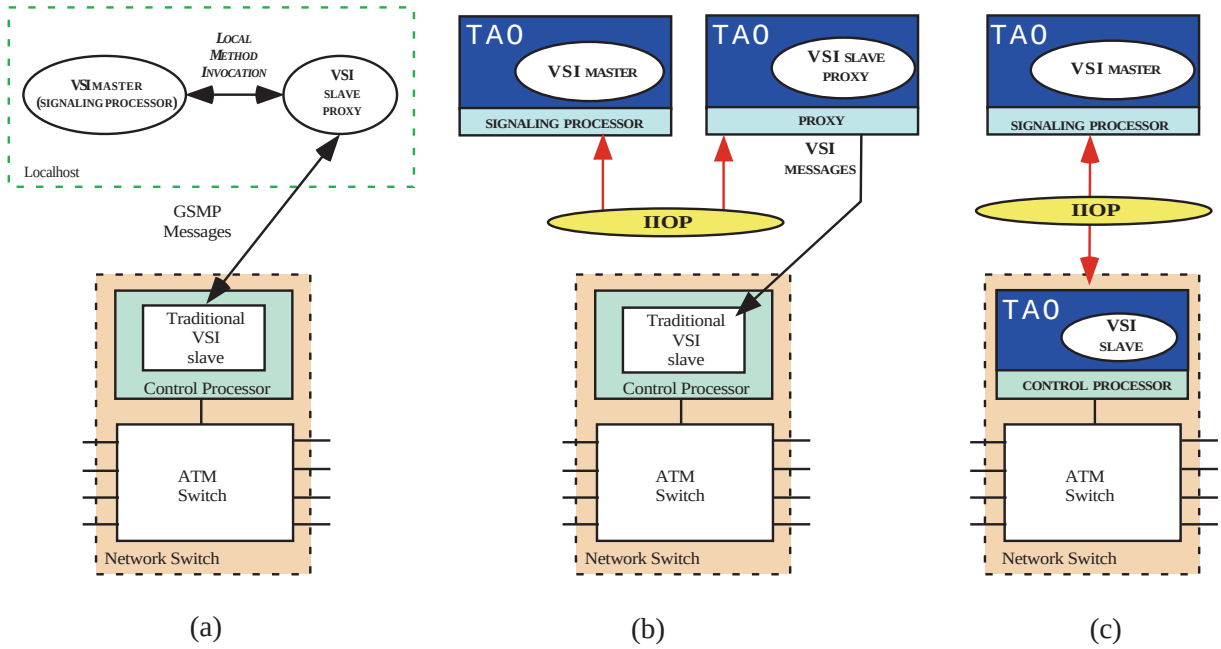

Figure 3: Three Configurations of VSI

ATM switch to improve the flexibility and control of applications and services across high-speed networks without sacrificing performance. Our target switch control environment includes support for VSI 2.0 as defined in the MSF specification [17]. We will use the resulting VSI middleware to build robust, efficient, and scalable ATM signaling applications. In particular, end-user applications can use our VSI++ framework to transparently setup end-to-end flows with associated Quality of Service (QoS) guarantees.

The result of this effort will be middleware-based ATM switch control framework based on VSI and CORBA. Within our VSI++ framework we will have ORBs (1) embedded on the switches and (2) resident on the client hosts and the controllers (*e.g.*, switch controllers). The broader goal of our work is to provide a switch/router control framework that can provision end-to-end QoS guarantees for real-time and high-bandwidth applications running over high-speed networks, such as ATM, Gigabit Ethernet, and high-speed IP routers.

To provide a baseline architecture for this work, we will construct three different CORBA- and VSI-based environments. Figure 3 depicts the three different configuration of master controllers, slave controllers, and ORBs supported by our VSI++ framework. These configuration are described further below:

1. Conventional VSI configuration: Figure 3 (a) depicts a conventional configuration of VSI. In this approach, we will create a VSI proxy object that translates local method invocations in a master controller into VSI messages that are sent to an ATM switch. These messages will be sent using the standard VSI ATM Encapsulation defined in Appendix A of the Virtual Switch Interface (VSI) Implementation Agreement, version 2. The only difference between this and the more traditional approach is that we have encapsulated the functionality in a C++ object rather than simply providing low-level C library function APIs. This configuration will provide us with a baseline implementation to compare against the results of the next two configurations.

Our performance measurements will separate out the overhead of the VSI messaging protocol and library implementation from the switch configuration time. This will enable us to later quantify the overhead associated with an ORB-based implementation from a straight OO library version.

2. VSI proxy server configuration: This configuration is depicted in Figure 3 (b) and will employ a real-time ORB to create a daemon proxy VSI slave controller that can reside on a remote host. This daemon server will translate the command method invocation into VSI messages and forward these to the control processor. It will then wait for any reply and include this in its response to the client. Our performance measurements will isolate the inter-ORB messaging overhead from the actual switch configuration latency. The goal is to isolate the various sources of messaging latency.

The advantage of the proxy strategy is that the proposed VSI++ framework can control switches that have an embedded ORB (described next), as well as those that only accept VSI messages or ATM control cells. In addition, one control processor can control multiple switches. Thus, we can use the VSI proxy server running on a host within the network to control multiple switches, each with the appropriate protocol.

Figure 4 depicts this situation where there are two master

controllers requesting connections across three switches. The proxy slave controller, *i.e.*, VSI proxy, can invoke the appropriate VSI method for the first switch, which contains an embedded VSI server built using an ORB. The middle switch accepts standard VSI control messages from the proxy slave controller. The third switch accepts ATM control cells to configure individual ports using a custom ATM solution.

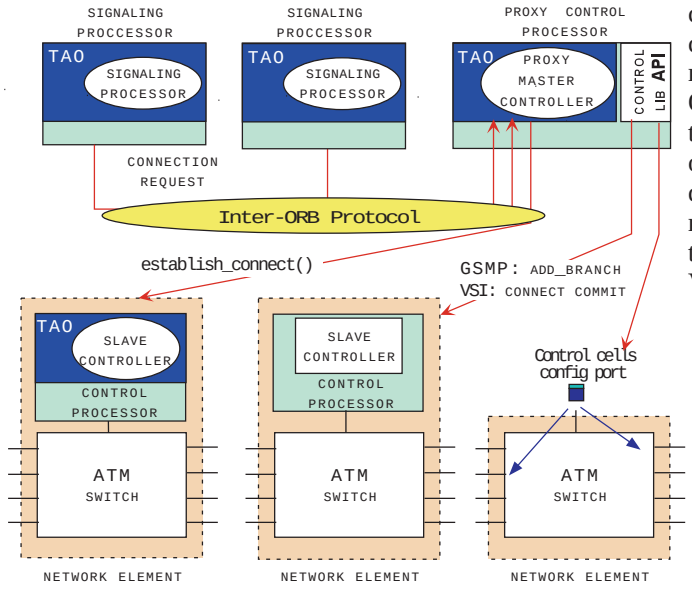

Figure 4: The VSI Proxy Object Configuration

3. Embedded ORB configuration: Using TAO to develop a network-embedded CORBA ORB configuration to provide a uniform access interface to network controls, services, and resources. The Embedded ORB configuration will support standard switch control operations and interfaces specified by VSI. Unlike conventional middleware efforts, however, the Embedded ORB configuration using TAO will be a minimal footprint [8] and real-time [9] version of CORBA that can be installed into a Cisco ATM switch. This configuration will improve the flexibility and control of applications and services across high-speed networks without sacrificing performance.

The Embedded ORB configuration is shown in Figure 3 (c), where the ORB and a VSI object are embedded in the ATM control processor (switch). This configuration is the most powerful and flexible configuration since the ORB middleware framework can be used to control the switch through out the network.

By embedding the ORB in the switch and exposing the VSI interface we can take advantage of the inherent semantics and flexibility of CORBA. There is the immediate advantage of simplifying the programming, provisioning, monitoring, and control of ATM signaling applications. Plus, the applications and services developed using our VSI middleware will yield more modular, extensible, and standard solutions that can be reused across multiple projects and application families. For example, new switch control functionality or interfaces can be added without exposing application developers to underlying details or complexities. Using this configuration, we can add support for new signaling standards or enhanced features while maintaining a consistent interface to developers of control plane and management plane applications.

Figure 5 shows another view of the embedded VSI server configuration where multiple signaling processors communicate with a network switch. In this case, all control and management communication occurs within the context of the ORB. An application running on an endstation will request that a connection be established from between itself and another endstation. The signaling processors will process the request, determine a route, and request each switch along the route to allocate the necessary resources. The master controller communicates with a slave controller using the exported VSI-based interfaces.

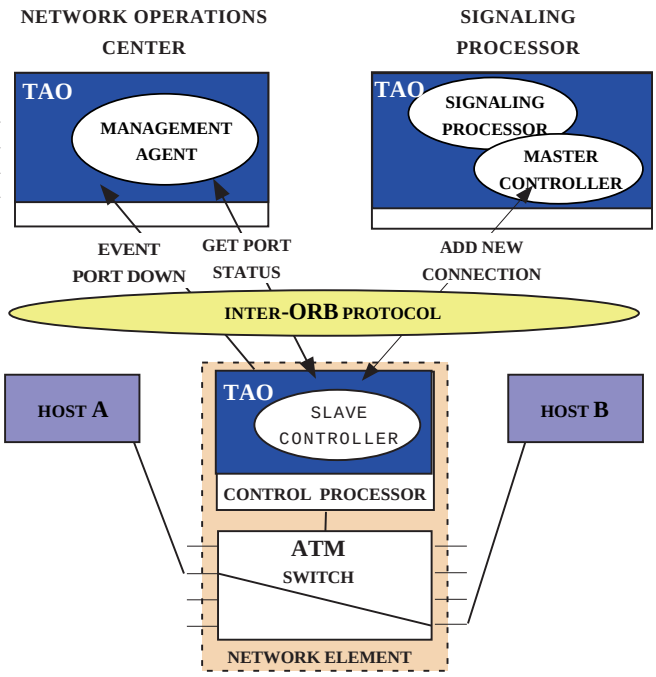

Figure 5: Using the Embedded ORB Configuration for VSI Signaling and Control

The following subsections describe the specific tasks we will perform during the 12 months of the proposed VSI project.

A.2.1 Task 1: Create an Enhanced OO Version of VSI

Activities: In this task, we will build on the ATM signaling work already performed at Washington University. In particular, we will develop a composite VSI++ model that includes

functionality defined in VSI version 2 [17]. The specific steps involved in this task include the following:

Step 1 – Develop an object model: We will survey existing VSI-based implementations and simulators for ATM switch management. Using this information, we will then create an object model and integrate this model into the core classes created in our earlier ATM signaling efforts. This step is primarily concerned with refining the VSI interface, key objects, and their implementation.

Step 2 – Develop Proxy Object In this step we will implement the classes and algorithms identified in step 1. We will use this to create a convention OO application that controls an ATM switch using the VSI messaging protocol. As a result of this step, the VSI objects and class hierarchy will implement the VSI object model identified in step 1.

Step 3 – Develop a Proxy Server: We will enhance the application written in step 2 to use CORBA in order to implement a proxy server configuration. In this scenario, client objects will communicate with the VSI server using CORBA and IIOP. We will experiment with collocating the server object with the client, as well as placing it on a remote host. In order to expedite this step, we plan to use the VSI master/slave controller simulator developed by Cisco.

Step 4 – Test and analysis: In the final step we will perform tests and compare the performance of the two approaches.

Deliverables: The deliverables for Task 1 will include the VSI++ class definitions, IDL definitions, testing results, and a report documenting the reference architecture that we have formulated. This report will identify and describe the key components, policies, and mechanisms related to the OO definition of VSI using CORBA. The deliverables in Task 1 will provide the basis for subsequent tasks described below.

A.2.2 Task 2: Port TAO to an ATM switch and Implement Optimal IOP

Activities: In this task, we will embed TAO in a Cisco ATM switch and add appropriate VSI protocol support into TAO using its new *pluggable protocols framework*. The specific steps involved in this task include the following:

Step 1 – ATM switch evaluation:

1. Acquire Cisco ATM switch and control processor software and verify operation.
2. Evaluate the configurability of the switches and develop mappings from VSI configuration messages to ATM switch specific control cells and/or commands.
3. Determine available computing resources, *e.g.*, memory footprint and processor cycles, that are necessary to support the ORB and VSI objects in the switch.

Step 2 – Port TAO to ATM switch: Evaluate both the ATM switch environment and required ORB functionality and determine which TAO and ACE subsets are required. Based on this analysis, build a minimal TAO with only the components required for the Cisco ATM switch environment. As part of this step, we will port ACE+TAO to the Greenhills Embedded C++ (EC++) compiler. EC++ will significantly reduce the memory footprint of ACE+TAO. This porting effort will require removing the use of multiple inheritance in ACE, as well as other minor changes to conform to the embedded C++ features.

Step 3 – Evaluate inter-ORB protocol requirements: The standard CORBA general inter-ORB protocol (GIOP) uses the Internet inter-ORB protocol (IIOP) as its transport protocol [10]. Since IIOP is implemented over TCP certain functionality like adaptive retransmissions, deferred transmissions, and delayed acknowledgments can cause excessive overhead and latency for ATM signaling applications with real-time QoS requirements. Therefore, we will evaluate alternate Environment-Specific Inter-ORB Protocols (ESIOP) that are customized for ATM/AAL5 and potentially UDP over ATM.

One approach is to use VSI as the Inter-ORB messaging protocol since it has several desirable features that lend itself to this role:

1. Request/response semantics are consistent with a standard IOP communication model.
2. Ability to communicate with switches that speak the VSI protocol, but which do not have an embedded ORB.

Step 4 – Implement inter-ORB messaging protocols: We will leverage TAO's *pluggable protocols framework* to define an optimized Inter-ORB Protocol that uses ATM and the ATM adaptation layer type 5 (AAL5). Three potential modification to the Inter-ORB protocol will be considered during this phase of the project. The first possibility is to use a modified version of the IIOP. The modifications will be minimal and targeted to supporting the ORB within the switch environment. For example, a lighterweight version of GIOP [14] may be employed to reduce overall message size. This IOP will operate over TCP/IP and ATM.

The second set of modification will be centered on developing a new ORB transport protocol that will operate under either GIOP or the lightweight version of GIOP identified above. In this case we will not require TCP/IP to be used as the underlying transport protocol. Instead, we will develop a lightweight ORB transport protocol using ATM and the ATM adaptation layer type 5 (AAL5). This will be augmented to provide a byte-stream interface to the ORB messaging component (GIOP). Depending on the switch environment UDP also may be considered before the underlying communications transport protocol.

The third set of modification will consider the creation of a new ORB messaging protocol. This new ESIOP will use

VSI as the ORB messaging protocol. This will operate over an unreliable, datagram protocol, such as UDP over AAL5 or simply ATM AAL5. Naturally, it will also be possible to use the standard GIOP/IIOP protocols to interoperate with other CORBA ORBs.

We plan to conduct extensive empirical tests using these three Inter-ORB protocols.

Deliverables: The deliverables for this step will be the embedded ORB, VSI and Inter-ORB Protocol. In addition, we will write papers and technical reports detailing the porting process and selection of IOP for optimal ATM switch control.

A.2.3 Task 3: Prototyping and Benchmarking the Real-Time Embedded ORB for ATM Switch Control and Management

Activities: This task will bring together the elements developed in tasks 1 and 2 to develop a prototype implementation of VSI with the ORB embedded in the ATM switch. We will perform extensive empirical testing of the embedded ORB and VSI implementation in the Cisco switch. These results will be compared to those collected in Task 1, where we used a proxy server and a proxy object that forwards requests to the switch using the standard VSI messaging protocol.

Our implemented VSI protocol will support the definition of QoS attributes. To the extent this is supported by the underlying network/end-system hardware/software we will experiment with defining classes of service and allocating network resources to applications. The specific steps involved in this task include the following:

Step 1 – Implement VSI within the Cisco ATM switch and Embedded ORB environment: We will first port our VSI server application to the switch environment on the control processor.

Step 2 – Implement VSI QoS features: VSI version 2 can be used to support different QoS classes required for multimedia support and telecommunication call-setup. QoS-related features we will explore include:

1. Real-time characterization and measurement;
2. Repeat Task 1 to check if QoS guarantees are being met;
3. Admission control policies at ATM switches;
4. Mapping admission controlled method invocations to real-time scheduling primitives, *e.g.*, thread-per-request, thread-per-object, and thread-per-client;
5. ORB modifications at switches to implement chosen policies.

A final dimension to be explored is providing a mechanism for associating QoS guarantees to the switch control and management messages, *i.e.*, the VSI messages themselves. For example, switch configuration messages, such as `connect_commit`, could be assigned a high priority both within the network and the embedded ORB. While routine management messages such as statics collection could have a relatively low, best effort priority. Alternatively, event notification messages that signal an error condition could have the highest priority and lowest latency.

As a part of this task we will explore the possibilities and their impact on non-control traffic, connection establishment latencies, error reporting latencies and overall endstation QoS negotiation times.

Step 3 – Conduct fault tolerance experiement with VSI++: The MSF VSI specification describes a set of fault tolerance features that allow a master controller and switch to resynchronize following a loss of connectivity. In this step, we will conduct experiements using the fault tolerance mechanisms being added to TAO under a proposed project with Motorola to determine how well they can be used to support the VSI++ fault tolerance policies.

Deliverables: The deliverables for this task will include a modified implementation of the VSI protocol in the QNX Neutrino operating system and TAO's CORBA IDL compiler, as well as sample applications and benchmarking results. In addition, we will work with the OMG to integrate TAO's pluggable protocols framework and QoS extensions into the CORBA standard.

A.3 Related Work

There are several efforts underway to define an open programming interface for network control and signaling. The OPENSIG group focuses on issues dealing with network control issues related to signaling, middleware, and service creation for various environments. This has resulted in Proposed IEEE Standard for Application Programming Interfaces for Networks, known as IEEE P1520 [5].

The P1520 effort is also based on VSI. Thus, we plan to follow it closely. This standardization effort is based on the XBIND and qGSMP research at the Columbia University COMET group [3]. We expect our proposed effort of embedding an ORB on an ATM switch and providing an VSI interface will be particularly relevant to the work of OPENSIG.

The Multiservice Switching Forum (MSF) is actively involved with architectural and switch interface issues. The MSF switching framework is designed so that alternative signaling and routing protocols can be employed without conflict. Consequently, different services and signaling protocols that are being defined by several IETF working groups, the ATM Forum, ITU, and others can operate in within the same infrastructure.

This multi-protocol, multi-service, and multi-vendor approach is at the heart of the MSF standardization effort for the Virtual Switch Interface (VSI). The VSI has a similar purpose to GSMP however it provides a much richer interface

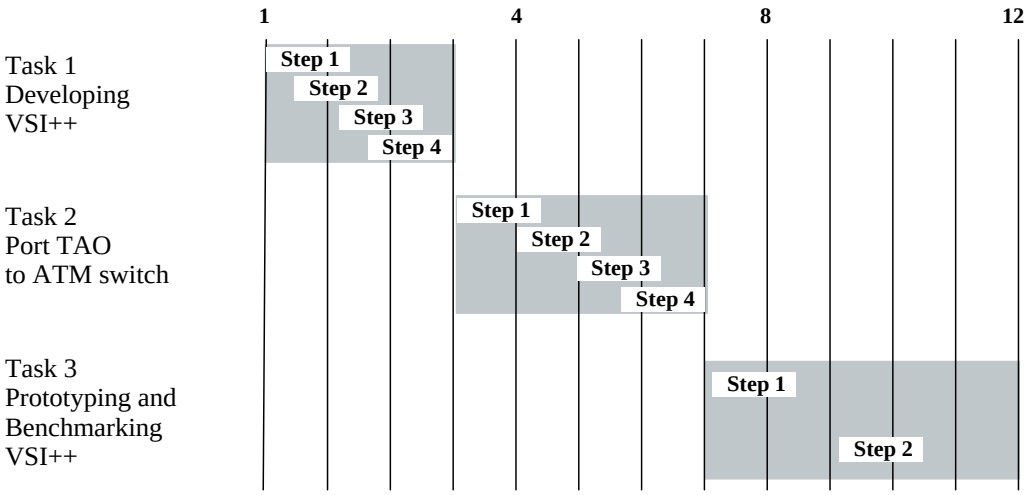

Figure 6: Timeline for Completion of Tasks.

specification. In fact the VSI is defined to avoid limiting the signaling protocols used and allows for multiple independent controllers, such as PNNI and MPLS, to simultaneously control the same switch. Thus using VSI a switch can support traffic for native ATM applications, IP over ATM and MPLS with each controller maintaining a separate view of the switch and responsible for its own resources.

B Personnel, Schedule, and Budget

The participants in this effort include the following personnel:

1. Faculty member (*i.e.*, Douglas C. Schmidt – Ph.D., Associate Professor, Washington University) at 10% during the period of performance.
2. Senior research associate (*i.e.*, Fred Kuhns, M.S.) at 50% during the period of performance.
3. Graduate student (*i.e.*, Vishal Kachroo, M.S. candidate Washington University) at 100% during the period of performance.

The time-table for completion of these task in the 12 month period of performance is shown in Figure 6. The total cost of the proposed effort is **$117,468**, split 50/50 between Motorola and Cisco. The following table provides a cost breakdown for this project.

Description	**Amount**
50% full time staff (salary & fringe)	40,000
1 graduate research assistant	25,300
10% Schmidt (salary & fringe)	10,000
overhead at 56%	42,168
total budget	117,468

References

[1] ATD. Advanced Technology Demonstration Network. http://www.atd.net/.

[2] Center for Distributed Object Computing. TAO: A High-performance, Real-time Object Request Broker (ORB). www.cs.wustl.edu/∼schmidt/TAO.html, Washington University.

[3] Constantin M. Adam, Aurel A. Lazar, and Mahesan Nandikesan. *Draft Technology Submission Working Document, Programming Interfaces for Networks*. Princeton, January 1999.

[4] Aniruddha Gokhale and Douglas C. Schmidt. Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems. *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, to appear, 1999.

[5] IEEE. IEEE P1520, Proposed IEEE Standard for Application Programming Interfaces for Networks. http://www.ieee-pin.org/.

[6] Fred Kuhns, Douglas C. Schmidt, and David L. Levine. The Design and Performance of a Real-time I/O Subsystem. In *Proceedings of the 5^{th} IEEE Real-Time Technology and Applications Symposium*, Vancouver, British Columbia, Canada, June 1999. IEEE.

[7] MSF. Multiservice Switching Forum. http://www.msforum.org/.

[8] Object Management Group. *Minimum CORBA - Joint Revised Submission*, OMG Document orbos/98-08-04 edition, August 1998.

[9] Object Management Group. *Realtime CORBA 1.0 Joint Submission*, OMG Document orbos/98-12-05 edition, December 1998.

[10] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 2.2 edition, February 1998.

[11] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. Ipsilon's General Switch Management Protocol Specification Version 1.1. Standards Track RFC 1987, Network Working Group, August 1996.

[12] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. Ipsilon's General Switch Management Protocol Specification Version 2.0. Standards Track RFC 2297, Network Working Group, March 1998.

[13] Guru Parulkar, Douglas C. Schmidt, and Jonathan S. Turner. $\mathrm{a^I t^P m}$: a Strategy for Integrating IP with ATM. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*. ACM, September 1995.

[14] Irfan Pyarali, Carlos O'Ryan, Douglas C. Schmidt, Nanbor Wang, Vishal Kachroo, and Aniruddha Gokhale. Applying Optimization Patterns to the Design of Real-time ORBs. In *Proceedings of the* 5^{th} *Conference on Object-Oriented Technologies and Systems*, San Diego, CA, May 1999. USENIX.

[15] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design and Performance of Real-Time Object Request Brokers. *Computer Communications*, 21(4):294–324, April 1998.

[16] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale. Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers. *Journal of Real-time Systems*, To appear 1999.

[17] VSI/1.0. Virtual Switch Interface (VSI) Specification, version 1.0. http://www.msforum.org/switch_control.pdf.