Timely Autonomic Adaptation of Publish/Subscribe Middleware in Dynamic Environments

Joe Hoffert, Aniruddha Gokhale, and Douglas C. Schmidt
Vanderbilt University, Nashville, TN, USA

ABSTRACT

Quality-of-service -enabled publish/subscribe (pub/sub) middleware provides powerful support for scalable data dissemination. It is hard, however, to maintain key quality of service properties (such as reliability and latency) in dynamic environments for *distributed real-time and embedded* systems (such as disaster relief operations or power grids). Managing quality of service manually is often not feasible in dynamic environments due to slow response times, the complexity of managing multiple interrelated quality of service settings, and the scale of the systems being managed. For certain domains, distributed real-time and embedded systems must be able to reflect on the conditions of their environment and adapt accordingly in a bounded amount of time.

This paper describes our architecture of quality of service-enabled middleware and corresponding algorithms to support specified quality of service in dynamic environments. The results of our work show that the middleware can monitor and adjust to a changing environment while addressing timeliness concerns of distributed real-time and embedded systems. In particular, the middleware can autonomically determine and implement appropriate adaptations in microseconds.

Keywords: Publish/Subscribe, Middleware, Autonomic Adaptation, Quality of Service, Dynamic Environments

INTRODUCTION

Emerging trends and challenges. The use of publish/subscribe (pub/sub) technologies for distributed real-time and embedded (DRE) systems has grown in recent years due to the advantages of performance, cost, and scale as compared to single computers (Huang, 2006; Tarkoma, 2006). In particular, pub/sub middleware has been leveraged to ease the complexities of data dissemination for DRE systems. Examples of pub/sub middleware include the CORBA Notification Service (Ramani, 2001), the *Java Message Service* (JMS) (Monson-Haefel, 2000), Web Services Brokered Notification (Niblett, 2005), and the *Data Distribution Service* (DDS) (Pardo-Castellote, 2003). These technologies support the propagation of data and events throughout a system using an anonymous publication and subscription model that decouples event suppliers and consumers.

Pub/sub middleware is used across a wide variety of application domains, ranging from shipboard computing environments to cloud computing to stock trading. Moreover, the middleware provides policies that affect the end-to-end quality of service (QoS) of applications running in

DRE systems. Policies that are common across various middleware technologies include grouped data transfer (*i.e.*, transmitting a group of data atomically), durability (*i.e.*, saving data for subsequent subscribers), and persistence (*i.e.*, saving data for current subscribers).

Even though tunable policies provide fine-grained control of system QoS, several challenges emerge when developing pub/sub systems deployed in dynamic environments. Middleware mechanisms used to ensure certain QoS properties for one environment configuration may be ineffective for different configurations. For example, a simple unicast protocol, such as the *User Datagram Protocol* (UDP), may address the specified latency QoS when a publisher sends to a small number of subscribers. UDP could incur too much latency, however, when used for a large number of subscribers due to its point-to-point property, leaving the publisher to manage the sending of data to each subscriber.

Challenges also arise when considering multiple QoS policies that interact with each other. For example, a system might need low latency QoS and high reliability QoS, which can affect latency due to data loss discovery and recovery. Certain transport protocols, such as UDP, provide low overhead but no end-to-end reliability. Other protocols, such as the Transmission Control Protocol (TCP), provide reliability but unbounded latencies due to acknowledgment-based retransmissions. Still other protocols, such as lateral error correction protocols (Balakrishnan, 2005), manage the potentially conflicting QoS properties of reliability and low latency, but only provide benefits over other protocols in specific environment configurations.

It is hard to determine when to switch from one transport protocol to another or modify parameters of a particular transport protocol so that desired QoS is maintained. Moreover, manual intervention is often not responsive enough for the timeliness requirements of the system. DRE systems operate within strict timing requirements that must be met for the systems to function appropriately. The problem of timely response is exacerbated as the scale of the system grows, *e.g.*, as the number of publishers or subscribers increases.

This article describes how our work (1) monitors environment changes that affect QoS, (2) determines in a timely manner which appropriate transport protocol changes are needed in response to environment changes, (3) integrates the use of multiple supervised machine learning techniques to increase accuracy, and (4) autonomically adapts the network protocols used to support the desired QoS. We have prototyped this approach in the *ADAptive Middleware And Network Transports* (ADAMANT) platform (as briefly outlined previously (Hoffert, 2009-A)) that supports environment monitoring and provides timely autonomic adaptation of the middleware. ADAMANT provides the following contributions to research on autonomic configuration of pub/sub middleware in dynamic environments:

- Leveraging anonymous publish and subscribe middleware based on the DDS specification. DDS defines topic-based high-performance pub/sub middleware to support DRE systems. ADAMANT leverages the middleware to provide environment monitoring information that is disseminated throughout the DRE system (*e.g.*, change in sending rate, change in network percentage loss) to provide updates occurring in the operating environment.

- Multiple *supervised machine learning* (SML) techniques as a knowledge base to provide fast and predictable adaptation guidance in dynamic environments. ADAMANT provides *timely integrated machine learning* (TIML), a novel approach to provide high accuracy and timely determination of which SML technique to use for a given operating environment.

- Configuration of DRE pub/sub middleware based on guidance from supervised machine learning. Our ADAMANT middleware uses the *adaptive network transports* (ANT) framework (Hoffert, 2009-B) to select the transport protocol(s) that best addresses multiple QoS concerns for given computing resources. ANT provides an infrastructure for composing and configuring transport protocols using modules that provide base functionality (*e.g.*, an IP multicast module that handles multicasting the data to the network). Supported protocols include Ricochet, which uses a variation of forward error correction called lateral error correction that exchanges error correction information among receivers (Balakrishnan, 2007), and NAKcast, which uses negative acknowledgments (NAKs) to provide reliability. These protocols enable trade-offs between latency and reliability to support middleware for enterprise DRE pub/sub systems.

This paper extends our prior work (Hoffert, 2010-A, 2010-B) on ADAMANT by exploring the architecture and control flow for autonomic adaptation of QoS-enabled pub/sub DRE systems. Moreover, this paper (1) empirically evaluates TIML as an approach to increase accuracy and maintain constant-time complexity, (2) leverages the DDS pub/sub infrastructure to disseminate environment changes to all data senders and receivers, (3) empirically evaluates the bounded response times of the ANT framework when adapting transport protocols, and (4) provides an autonomic controller that manages the adaptation of transport protocols to support QoS as the environment changes and details how the controller manages the adaptations. The paper is organized as follows. We start by describing a motivating example and highlighting the challenges. Next, we present the structure and functionality of the ADAMANT framework. We then detail and analyze our experimental results. We compare our work with related research in autonomic adaptation. Finally, we conclude with lessons learned.

MOTIVATING EXAMPLE: AMBIENT ASSISTED LIVING IN A SMART CITY ENVIRONMENT

This section describes *Smart City Ambient Assisted Living* (SCAAL) applications, which combine *Ambient Assisted Living* (AAL) in the context of a smart city. It also presents research challenges associated with SCAAL applications. SCAAL applications help motivate the need for managing QoS interactions and providing timely adjustments of transport protocols for QoS-enabled pub/sub middleware deployed in dynamic environments. The objective for smart cities is to meld computational infrastructure into the surrounding environment and establish ubiquitous, context-aware services in a metropolitan area (Chandy, 2007). The purpose of AAL is to increase the independence and quality of life for elderly people, while decreasing the need for direct interaction of healthcare workers so they are freed up for other concerns.

As an example SCAAL scenario depicted in Figure 1, imagine an elderly person is navigating a large metropolitan area equipped with multiple technological devices. These devices aid in various aspects of the person's ability to be aware of her environment, *e.g.*, mobility, sensory enhancement, communication, and monitoring devices. In particular, the elderly person has a history of heart disease and 3-dimensional high-resolution heart monitoring equipment is periodically transmitting data. A personal datacenter publishes and subscribes to the data being managed by the personal devices including the heart monitoring data, and interfaces with the smart city by publishing and subscribing to data from the ambient environment. More specifically, health care workers, hospitals, and emergency medical services specialists are subscribing to the heart monitoring information that is being published.

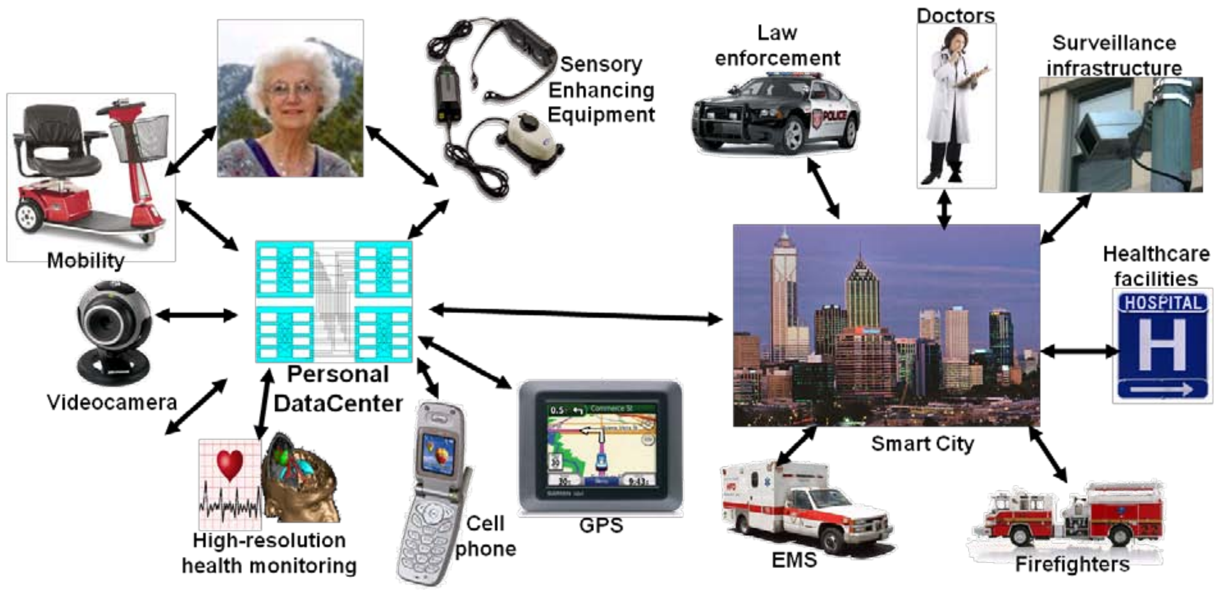


Figure 1: Smart City Ambient Assisted Living (SCAAL) Example

The personal datacenter operates in a dynamic environment since (1) the elderly person moves through space in the smart city and updates personal information in time and (2) the smart city enhances and updates the amount and kind of data that it provides as it moves through time. Our research focuses on (1) composite metrics to evaluate transport protocols in support of multiple QoS concerns (such as reliability and low latency for high-resolution 3D heart monitoring information), (2) evaluations of multiple transport protocols in different operating environments using the composite metrics, (3) support for monitoring the environment, (4) supervised machine learning techniques to determine transport protocols that best support the QoS that a personal datacenter device must manage in a SCAAL application, and (5) autonomically adapting the transport protocols to provide the best QoS given the changes in the environment. Supporting autonomic adaptation of the personal datacenter presents the following challenges:

Challenge 1: Managing interacting QoS requirements. The personal datacenter must manage multiple interacting QoS requirements, *e.g.*, data reliability so enough data is received and low latency and jitter for soft real-time data so that detailed 3-dimensional heart monitoring data arrive before they are needed. For example, the streamed data must be received soon enough so that successive dependent data can also be used, such as dependent MPEG B and P frame data being received before the next I frame makes them obsolete. Moreover, the personal datacenter must balance the interacting QoS requirements with an environment that varies dynamically, *e.g.*, number of data senders and receivers, network bandwidth, network packet loss. Section *Addressing Challenge 1: Managing Interacting QoS Requirements* describes how we address this challenge by supporting runtime migration and reconfiguration in bounded time of transport protocols used as the QoS mechanisms to provide needed QoS.

Challenge 2: Accurate Adaptation. The personal datacenter must be able to adjust to changes in the environment accurately. As changes in environment occur (*e.g.*, increases in heart data updates, decreases in networking capability, requests for data from additional senders and receivers), the personal datacenter must accommodate data needs for data producers and consumers, take advantage of additional resources, or provide access to additional data producers and consumers while maintaining QoS. For a given environment configuration, the SCAAL application must accurately implement adjustments that are appropriate to the operating environment. If the personal datacenter cannot make accurate adjustments as the environment changes then situation awareness and critical health information could be lost or delayed causing loss of orientation or injury to the elderly person. Section *Addressing Challenge 2: Accurate Adaptation* describes how we address this challenge by leveraging DDS to disseminate the environment monitoring information needed to determine an accurate adaptation and TIML to accurately determine the appropriate transport protocol.

Challenge 3: Timely Adaptation. Due to timeliness concerns of DRE systems such as SCAAL applications, the personal datacenter must adjust in a timely manner as the environment changes. If the personal datacenter cannot adjust quickly enough it will fail to perform adequately and critical data such as 3-D heart information will not be received in time. As the amount of data relevant to the SCAAL application fluctuates and the demand for information varies with a corresponding change in the data update rate, the personal datacenter must be configured to accommodate these changes with appropriate responsiveness to maintain the specified quality of service. Configuration changes must not only be timely in general but they must also be bounded – and ideally constant time – so that critical information updates (such as health monitoring) are not lost or received too late to be of use. Section *Addressing Challenge 3: Timely Adaptation* describes how we address this challenge by using constant-time complexity machine learning techniques, constant-time integration of these techniques, and constant-time migration of transport protocols.

Challenge 4: Reducing development complexity. Many elderly people can use a personal datacenter to improve their independence. Likewise, the health care industry can benefit from the decreased workload for health care providers. A personal datacenter that is developed for one particular elderly individual in a particular operating environment, however, might not work well for a different elderly individual in a different operating environment with different personal equipment. For example, one elderly person might have a cell phone, visual enhancing equipment, and a heart monitor in a smart metropolitan area where police officers, firefighters, and healthcare workers can subscribe to and publish information relevant to the city. Another elderly person might have a PDA, a GPS device, a video camera, and a personal monitoring screen that is able to display video of what is behind the person in a smart metropolitan area where ambulances, hospitals, and traffic monitoring cameras can subscribe to and publish information relevant to the city. These two different scenarios impose different pub/sub demands (*e.g.,* the number of publishers and subscribers, the data sending rates)

The personal datacenters used to manage the information between the elderly person and the smart city therefore need to be developed and configured readily for any one particular operating environment accounting for different metropolitan areas, differences in personal equipment, and differences in the data needs of various individuals to leverage the personal datacenters across a wide range of individuals and locales. Section *Addressing Challenge 4: Reducing Development Complexity* describes how we address this challenge by leveraging DDS to disseminate environment updates, and using machine learning to map environment configurations to the appropriate transport protocols.

STRUCTURE AND FUNCTIONALITY OF ADAMANT

This section presents the structure and functionality of the ADAMANT middleware platform, focusing on its software architecture and control flow. It also describes how ADAMANT addresses the challenges of SCAAL applications presented in section *Motivating Example: Ambient Assisted Living in a Smart City Environment*.

Architecture of ADAMANT

Figure 2 shows ADAMANT's control flow and logical architecture. This section details the architecture of ADAMANT while the following section *Control Flow of ADAMANT* describes how autonomic adaptation is manifested in ADAMANT in each one of the steps illustrated in Figure 2. ADAMANT integrates and enhances the following technologies and innovative techniques to provide autonomic adaptation of DRE pub/sub middleware in dynamic environments and address the challenges listed in the motivating example section:

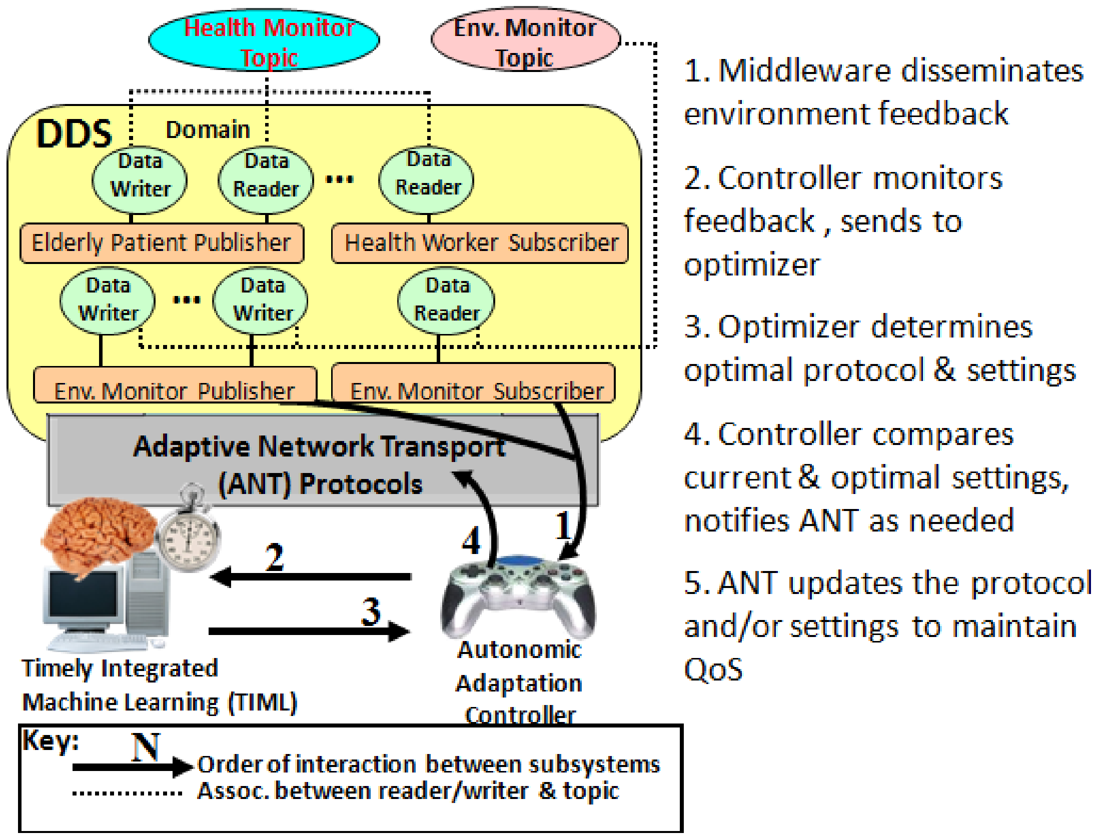

Health Monitor Topic
Env. Monitor Topic
DDS
Domain
Data Writer
Data Reader
Data Reader
Elderly Patient Publisher
Health Worker Subscriber
Data Writer
Data Writer
Data Reader
Env. Monitor Publisher
Env. Monitor Subscriber
Adaptive Network Transport (ANT) Protocols
1
2
3
4
Timely Integrated Machine Learning (TIML)
Autonomic Adaptation Controller
1. Middleware disseminates environment feedback
2. Controller monitors feedback , sends to optimizer
3. Optimizer determines optimal protocol & settings
4. Controller compares current & optimal settings, notifies ANT as needed
5. ANT updates the protocol and/or settings to maintain QoS
Key:
N
Order of interaction between subsystems
Assoc. between reader/writer & topic

Figure 2: ADAMANT Architecture and Control Flow

- The OMG Data Distribution Service (DDS) is standards-based QoS-enabled pub/sub middleware for exchanging data in event-based DRE systems. It provides a global data store in which publishers and subscribers write and read data, respectively. ADAMANT uses DDS to provide the infrastructure for disseminating environment monitoring information needed to determine accurate adaptations, as well as normal application data, such as the health monitoring information in SCAAL applications. DDS enables applications to communicate by publishing information they have and subscribing to information they need in real time.

  DDS enables *flexibility* and *modular structure* by decoupling: *location*, via anonymous publish/subscribe; *redundancy*, by allowing any numbers of readers and writers; *time*, by providing asynchronous, time-independent data distribution; and *platform*, by supporting a platform-independent model that can be mapped to different languages (*e.g.*, Java and C++).

The DDS architecture consists of two layers: (1) the *data-centric pub/sub* (DCPS) layer that provides APIs to exchange topic data based on chosen QoS policies and (2) the *data local reconstruction layer* (DLRL) that makes topic data appear local. Our work focuses on DCPS since it is more broadly supported than the DLRL. Moreover, DCPS provides finer grained control of QoS.

The DCPS entities in DDS include *topics*, which describe the type of data to write or read; *data readers*, which subscribe to the values or instances of particular topics; and *data writers*, which publish values or instances for particular topics. Moreover, *publishers* manage groups of data writers and *subscribers* manage groups of data readers. Various properties of these entities can be configured using combinations of the 22 DDS QoS policies .

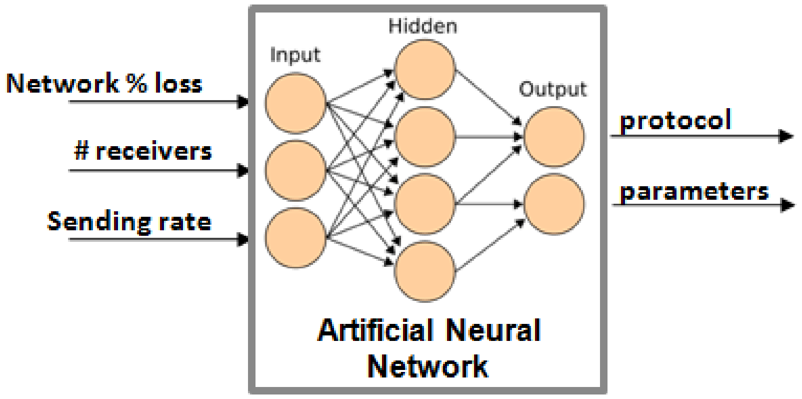


Figure 3: Artificial Neural Network for Determining Appropriate Transport Protocol

DDS’ rich support for QoS can be applied for application data and for the environment monitoring topic that ADAMANT provides (*e.g.*, prioritization for transporting and managing the operating environment updates as well as the application data).

- TIML provides a novel integration of multiple supervised machine learning techniques as a knowledge base. This knowledge base, in turn, provides fast and predictable adaptation guidance in dynamic environments. TIML also uses machine learning techniques to manage the inherent complexity of providing the appropriate transport protocol recommendation for a given operating environment. TIML utilizes perfect hashing (Brodnik, 1994) on the mapping of environment configurations to transport protocols to provide constant-time determination of which supervised machine learning technique to use for a given environment configuration. In particular, TIML utilizes the GPERF (Schmidt, 2000-A) open-source implementation of perfect hashing.

For our ADAMANT prototype TIML uses several supervised machine learning techniques, including *Artificial Neural Networks* (ANNs) (Patterson, 1998) to determine in a timely manner the appropriate transport protocol for the QoS-enabled pub/sub middleware platform given an environment configuration that is known *a priori* (*i.e.*, used for training). It also uses *Support Vector Machines* (SVMs) (Meyer, 2003) to determine in a timely manner the appropriate transport protocol for an environment configuration unknown until run-time (*i.e.*, not used for training).

An ANN is a supervised machine learning technique modeled on neuron interactions in the human brain. As shown in Figure 3, an ANN has an input layer for aspects of the operating environment, *e.g.*, percent network loss and sending rate. An output layer represents the solution generated based on the input. A hidden layer connects the input and output layers. As the ANN is trained on inputs and correspondingly correct outputs, it strengthens or weakens connections between layers to generalize based on inputs and outputs.

Figure 3 also shows how an ANN can be configured statically in the number of hidden layers and the number of nodes in each layer that directly affects the processing time complexity between the input of operating environment conditions and the output of an appropriate transport protocol and settings. This static configuration structure supports bounded response times.

SVMs are supervised learning techniques used for classification and prediction. An SVM is first given a set of training examples where each example is denoted as belonging to a particular class or grouping. An SVM next builds a model that predicts into which grouping a new example should be categorized. As shown in Figure 4, the SVM creates classification boundaries between the different classification groupings to maximize the differences between the groupings. Leveraging the heuristic of locality, this maximization helps to correctly classify new examples that have not been used in training the SVM model, *i.e.*, examples in the same group are deemed fairly close to each other in the classification space. Like ANNs, SVMs are configured offline during training to exhibit bounded response times at runtime.

- ADAMANT uses the ANT framework to select the transport protocol(s) that best address multiple QoS concerns for a given operating environment. ANT provides infrastructure for composing and configuring transport protocols via base modules, such as the `IPMulticastModule` that supports sending out and receiving data using IP Multicast. These modules can flexibly and dynamically be connected together by publishing and subscribing to event types (*e.g.*, `SEND_PACKET_EVENT`, `GOT_PACKET_EVENT`, `SEND_NAK_EVENT`, and `GOT_NAK_EVENT`).

  ANT supports transport protocols that balance the need for reliability and low latency. For example, Ricochet enables trade-offs between latency and reliability to support middleware for DRE pub/sub systems involved with dissemination of multimedia data. The ANT framework allows ADAMANT to change and reconfigure transport protocols (including protocol parameters) while an application is running. The time complexity for ANT to reconfigure protocols is bounded as needed for DRE systems.

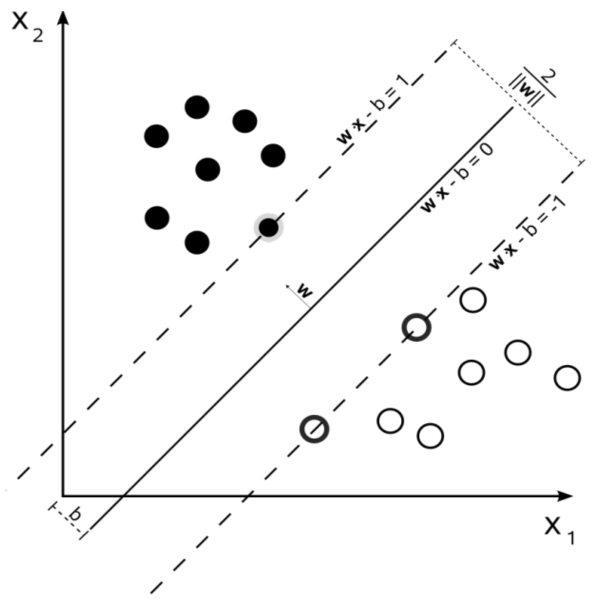

X_2
X_1
$\mathbf{w} \cdot \mathbf{x} - b = 1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w}$
b

Figure 4: Maximizing Grouping Differences in a Support Vector Machine

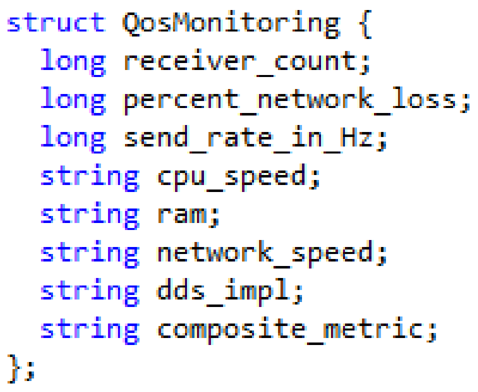
- A QoS monitoring topic defines the data for environment information relevant to adapting transport protocols. ADAMANT leverages DDS to provide this topic dedicated to describing the operating environment of an application. This environment information is used to determine appropriate adaptation of the QoS mechanisms in ADAMANT, namely, the transport protocols. Moreover, since ADAMANT leverages DDS to create the environment monitoring topic, DDS QoS policies can also be applied to the dissemination of this topic data providing fine-grained control as to when and how environment configuration updates are propagated in the SCAAL application (*e.g.*, applying DDS' transport priority QoS policy to health monitoring data to ensure the data has priority over other data on the network).
- Autonomic control manages the adaptation process. ADAMANT provides an autonomic controller that responds to changes in the operating environment. Whenever environment changes are communicated via ADAMANT's environment monitoring topic, the controller passes the changes to TIML to determine the appropriate response. The controller then passes TIML's recommended adaptation to the ANT framework to change the transport protocols while the system is running.

Control Flow of ADAMANT

ADAMANT supports the *Monitor, Analyze, Plan, Execute – Knowledge* approach (Kephart, 2003; Huebscher, 2008), which abstracts the management architecture into the four needed functions of collecting data, analyzing the data, creating a plan of action based on the updated data and corresponding analysis, and executing the plan. Undergirding these functions is knowledge extraction which raises the level of abstraction from data collection and analysis to determining the behavior and state of the system along with the appropriate high level adaptations. ADAMANT components are physically distributed across the computing platforms in the system, *e.g.*, each computing platform has its own identical instantiation of TIML and the autonomic adaptation controller. Since environment configuration changes are published to all subscribers via DDS, all local ADAMANT components receive the same updates. Since components are deterministic, they generate the same transport protocol to use and initiate the same protocol modifications. This distributed architecture enables scalability in the number of publishers, subscribers, and computing platforms.

The first step in ADAMANT's control flow (shown as Step 1 in Figure 2) is receiving changes to the environment configuration. ADAMANT creates and supports an environment monitoring topic to which various application data senders and receivers can publish and subscribe, respectively. For example, the heart monitoring portion of the SCAAL application can publish changes to the environment monitor topic when it adjusts its data sending rate based on requests from health workers subscribing to the data. Likewise, data subscribers can query the environment monitor topic for the periodic sending rate of the data and then calculate the loss percentage in the network by dividing the expected number of data updates for a given period with the actual number of updates received.

Figure 5 shows the data described in environment monitor topic. The data is described in the platform-independent *interface definition language* (IDL) as defined by the OMG. Our prototype is interested in the attributes of the environment information shown in Table 2 since the data values for these aspects are used to determine the most appropriate transport protocol.



```
struct QosMonitoring {
  long receiver_count;
  long percent_network_loss;
  long send_rate_in_Hz;
  string cpu_speed;
  string ram;
  string network_speed;
  string dds_impl;
  string composite_metric;
};
```

Figure 5: Environment Monitor Topic

Table 2:QoSMonitoring Attributes

Attribute	*Description*
receiver_count	The number of receivers currently receiving application data (*e.g.*, value 5 indicates that 5 receivers are receiving application data).
percent_network_loss	The percent packet loss in the network (*e.g.*, value 3 indicates that a 3% loss of packets occurs in the network).
send_rate_in_Hz	The data sending rate for the heart monitoring data in Hz (*e.g.*, value 50 indicates a sending rate of 50 Hz, *i.e.*, 50 times a second).
cpu_speed	The speed of the CPU being used in MHz (*e.g.*, "2992.883" indicates a CPU speed of 2.992883 GHz). For clarity and simplicity, the ADAMANT prototype assumes common CPU speeds for all machines used.
RAM	The amount of random-access memory available on the machines being used in kilobytes (*e.g.*, "2062172" indicates 2 GB of RAM). Again, for clarity and simplicity, the ADAMANT prototype assumes common amount of RAM for all machines used.
network_speed	The speed of the network being used in Mb/sec (*e.g.*, "1000" indicates a 1 Gb/sec network).
dds_impl	The DDS implementation being used (*e.g.*, value "OpenSplice" indicates the use of PrismTech's OpenSplice DDS implementation). For simplicity as a proof of concept, the ADAMANT prototype only currently supports the OpenSplice DDS implementation, though support for the other DDS implementations (*e.g.*, OpenDDS or RTI DDS) can easily be added.
composite_metric	The composite metric of interest to the application, *e.g.*, "ReLate2". The ReLate2 composite metrics quantitatively evaluate multiple QoS properties. For example, the ReLate2 metric combines data reliability and latency to produce a single value used for quantitative comparison. Other composite metrics include ReLate2Jit that quantitatively evaluates data reliability, latency, and jitter; ReLate2Net that evaluates reliability, latency, and network bandwidth usage; and ReLate2Burst that evaluates reliability, latency, and network data burstiness (Hoffert, 2010-A).

After updates have been made to the environment monitor topic, the autonomic controller receives the updated environment configuration (outlined as Step 2 in Figure 2). The autonomic controller then compares the new and previous environment configurations. If the configurations are different the controller invokes TIML to determine which transport protocol and parameter values best support the desired QoS. If the configurations do not differ the autonomic controller simply returns since no adaptation is needed.

Step 3 in Figure 2 shows how TIML receives the new environment configuration and determines if the configuration is one on which the machine learning techniques have been trained. If the machine learning techniques have previously been trained off-line using the configuration, TIML uses an ANN to determine the appropriate transport protocol and parameter settings. Since we overfitted[1] the ANN to our experimental training data, the ANN produces 100% accurate determinations for these known environment configurations as illustrated in our previous work (Hoffert, 2010-B; Hoffert, 2010-C).

If machine learning techniques have not previously been trained on an environment configuration, however, TIML uses an SVM to determine the appropriate transport protocol and parameter settings. Our prior work (Hoffert, 2010-C) shows how an SVM will provide higher accuracy for determining the appropriate protocol and parameters than an ANN when the input environment configuration was not used during off-line training (*i.e.*, unknown until run-time). The overall accuracy of ADAMANT is enhanced by combining the 100% accuracy of an overfitted ANN for environment configurations known *a priori* with the higher accuracy of an SVM for environment configurations unknown until run-time. In particular, we see an increase in accuracy of 8.6% combining both an ANN and an SVM, compared to only using an ANN (*i.e.*, 77.69% average ANN accuracy for environments unknown until run-time compared to 86.29% for the SVM).

Both ANNs and SVM provide constant-time complexity for determining protocols and parameters. The mechanism used to determine if the environment configuration have been known *a priori* must therefore also provide constant-time complexity to maintain this time complexity for the entire protocol optimization process. TIML utilizes perfect hashing for the environment configurations to determine in constant time whether or not an environment configuration is known *a priori* (*i.e.*, used for training) or unknown until run-time. Since the environment configurations used for training the ANN are known *a priori* TIML uses this information to construct the perfect hash which can be thought of as a constant-time table lookup. TIML provides the environment configurations on which the ANN has been trained as keys to the perfect hashing to map to the corresponding scaled environment configuration data. If the key is found via the perfect hash the TIML knows that the environment configuration has been seen before in off-line training and then uses the ANN since it will provide perfect accuracy. If the key is not mapped, then TIML will use the SVM since it provides the highest accuracy for environment configurations that are unknown until runtime.

Once the appropriate transport protocol has been decided, TIML returns this result to the autonomic controller (Step 4 in Figure 2). The controller then compares the recommended transport protocol and protocol parameters with the current transport protocol and protocol parameters. If there is no difference, the controller need not take any further action. If there are differences between the current protocol and the recommended protocol, the controller passes the new protocol settings to ANT to make the needed adaptation.

[1] Overfitted ANNs are specialized for the environments they have seen—and on which they have been trained—which reduces development complexity and increases accuracy (Dietterich, 1995).

Our ADAMANT prototype uses the OpenSplice DDS implementation, which uses a networking daemon on each machine to send and receive data across machine boundaries. The ANT framework resides in the networking daemon since the ANT protocols are used to disseminate the application data across the network. The autonomic controller resides in the application executable since it needs to respond to updates in the environment as facilitated by the environment monitor topic. For a single computer platform, OpenSplice uses shared memory to communicate between the SCAAL application executable and the OpenSplice daemon. Since the daemon runs as a separate process from the application executable, some form of interprocess communication (IPC) is needed to have the controller inform ANT of the needed protocol changes.

The form of IPC used when communicating between the autonomic controller and ANT can vary depending upon the needs of the application and the IPC mechanisms supported by the operating system. In our ADAMANT prototype the autonomic controller residing in the application executable sends a signal to ANT residing in the networking daemon. The OpenSplice networking daemon is enhanced to include a signal handler. In particular, when the controller determines the transport protocol must be modified it sends a signal (e.g., SIGUSR1) to the networking daemon. When the networking daemon processes the SIGHUP signal, the daemon invokes ANT to reconfigure. ADAMANT utilizes the Component Configurator pattern (Schmidt, 2000-B) for ANT to reconfigure itself by constructing the appropriate configuration file and then signaling ANT to reconfigure.

The need for IPC depends upon the DDS implementation. For example, rather than using a network daemon, the OpenDDS DDS implementation supports direct point-to-point network connectivity between application executables residing on different machines. For ADAMANT using OpenDDS, intra-process communication would be needed rather than IPC. ADAMANT would set a variable accessible across threads using appropriate locking mechanisms. ANT would then wait until the variable was set (*e.g.*, using a condition variable) and reconfigure the transport protocol as needed.

After ANT receives the signal to reconfigure (Step 5 in Figure 2) it determines whether to modify an existing transport protocol or switch to a new protocol. ANT keeps track of the current transport protocol being used for comparison. If the current protocol must be modified then ANT invokes the appropriate methods on the relevant protocol modules to change the protocol parameters. If a new protocol must be used ANT first disables the existing protocol and enables the new protocol.

The modules in the ANT framework use pub/sub communication to consume and supply events of interest. This approach allows for flexibility in the way modules are connected together to create the functionality needed for a particular transport protocol. This approach also allows the enabling/disabling of transport protocols simply by registering and unregistering for particular events. ANT thus unregisters events for the old protocol to disable the old protocol and registers events for the new protocol to enable the new protocol.

Addressing Challenges of SCAAL Applications

This section describes how ADAMANT addresses the challenges of SCAAL applications presented in the section *Motivating Example: Ambient Assisted Living in a Smart City Environment*.

Addressing Challenge 1: Managing interacting QoS requirements

ADAMANT addresses the challenge of managing interacting QoS requirements by using the transport protocols provided by the ANT framework. ANT supports transport protocols that address interacting QoS requirements. In particular, it provides the NAKcast and Ricochet transport protocols that balance the contentious QoS requirements of data reliability and low latency. As shown in previous work (Hoffert, 2009-B), these protocols ameliorate the loss of network data packets while imposing low latency overhead. In particular, the NAKcast protocol uses negative acknowledgments (that is, NAKs) that the receiver sends to the sender for notification of lost data packets. NAKcast provides a tunable timeout parameter to determine when NAKs should be sent. The Ricochet protocol supports error correction information that the receivers send to each other to recover from lost data packets. Ricochet provides a tunable parameter to determine how many data packets need to be received before error correction is sent out. Ricochet also provides a tunable parameter to determine how many other receivers receive the error correction information from a single receiver.

Addressing Challenge 2: Accurate adaptation

ADAMANT addresses the challenge of accurate adaptation in several ways. First, it leverages the use of DDS to provide the infrastructure to disseminate the environment monitoring information needed to determine an accurate adaptation. Second, it uses TIML to provide an integration of multiple supervised machine learning techniques to provide high accuracy for both operating environments known *a priori* and operating environments unknown until runtime. TIML supports accurate adaptation guidance in dynamic environments by using the most accurate machine learning technique for operating environments known *a priori* (*i.e.,* ANNs) integrated with the most accurate technique for operating environments unknown until runtime (*i.e.*, SVMs). Our previous work presents experimental results illustrating the accuracy of ANNs and SVMs (Hoffert, 2010-C). In particular, the SVM accuracy results use *n*-fold cross validation where *n* is the number of mutually exclusive training and testing data sets (Liu, 2006). Third, ADAMANT's autonomic controller ensures accuracy by managing the adaptation process of receiving environment updates, delegating this information to TIML to provide guidance, and passing the recommended transitions to ANT.

Addressing Challenge 3: Timely adaptation

ADAMANT addresses the challenge of timely adaptation in several ways. First, as already mentioned for Challenge 2, it uses DDS to disseminate the environment monitoring information needed to determine an accurate adaptation. Second, since the monitoring information is realized as a DDS topic, the DDS QoS policies can be applied to the topic and the applicable entities involved with the topic (*e.g.*, data readers, data writers). For example, the transport priority QoS policy can be applied to the environment monitoring data to ensure the environment updates have priority over other data on the network.

ADAMANT supports constant-time runtime transition and reconfiguration of transport protocols used as the QoS mechanisms to provide needed QoS, as discussed in the section *Experimental Results and Analysis*. In particular, TIML utilizes an ANN to provide adaptation guidance in constant time for operating environments known *a priori*. TIML uses an SVM to guide adaptation in constant time for operating environments unknown until runtime. Moreover, TIML uses constant-time perfect hashing to integrate the machine learning techniques and determine the appropriate technique to use.

Addressing Challenge 4: Reducing development complexity

ADAMANT addresses the challenge of reducing development complexity by using machine learning techniques that manage the inherent complexity of providing the appropriate transport protocol recommendation for a given operating environment. The machine learning techniques can also be used directly in the ADAMANT implementation. These techniques thus reduce development complexity by eliminating the accidental complexity of transforming the mapping of environments to protocols from design to implementation (Hoffert, 2010-A). Moreover, ADAMANT provides an environment monitoring topic that disseminates and handles the environment information updates relevant to adapting the QoS mechanisms of transport protocols.

EXPERIMENTAL RESULTS AND ANALYSIS

This section describes the setup, design, and analysis of results from experiments we conducted to identify the need for autonomic adaptation of transport protocols and evaluate the timeliness of the adaptations in dynamic environments representative of the SCAAL applications presented in the section *Motivating Example: Ambient Assisted Living in a Smart City Environment*. These results quantify (1) the effect of changes in the operating environment on the QoS provided by ADAMANT as measured by the composite QoS metrics defined below, (2) the timeliness of TIML's determination of an appropriate transport protocol, and (3) the timeliness of ADAMANT's adaptation of transport protocols via the ANT framework.

Experimental Setup

We conducted our experiments using the Emulab testbed (www.emulab.net) at the University of Utah. Emulab allows the configuration of various types of computing and networking platforms. For our experiments highlighting the need for adaptation, we held the computing and networking platform constant (*i.e.*, 3 GHz CPU, 1 Gbps LAN). We used the Redhat Fedora Core release 6 OS with real-time patches across all the computing nodes.

The points of variability for the experiments were indicative of dynamic environments. In particular, we varied the number of data receivers, the percent loss in the network, and the data sending rate as outlined in section *Control Flow of ADAMANT*. By adjusting these variables we were able to highlight scenarios where changes in the environment mandated changes to the transport protocols being used to provide the highest level of QoS for the multiple QoS properties involved.

Composite QoS Metrics for Reliability and Timeliness

Our previous work on QoS-enabled pub/sub middleware performance (Hoffert, 2009-B, 2010-A) showed that some transport protocols provide better reliability (as measured by the number of network packets received divided by the number sent) and lower data latency for particular environments while other protocols provide better reliability and lower latency for other environments. We therefore developed several composite QoS metrics to quantitatively evaluate multiple QoS aspects simultaneously. These composite metrics provide a uniform and quantitative evaluation of ADAMANT in dynamic environments. Our family of composite metrics are based on the QoS concerns of reliability and average latency and optionally include the QoS aspects of (1) jitter (*i.e.*, standard deviation of the latency of network packets), (2) network bandwidth usage, and (3) burstiness (*i.e.*, the standard deviation of average bandwidth usage per second of time).

In particular, we defined the ReLate2 family of composite QoS metrics. The ReLate2 metric is defined by the product of the average data packet latency and the percent loss that the transport protocol provides + 1 (to account for 0% loss) which implies an order of magnitude increase for 9% loss. Based on previous research which evaluated percent data loss for multimedia applications (Bai, 2006, 2007; Ngatman, 2008), this adjustment is relevant for multimedia data such as the high-resolution 3-D health data in our SCAAL example. For example, if for a given protocol the average packet latency is 1,000 µs and the percent loss is 0 (*i.e.*, no packets lost) then the ReLate2 value is 1,000. Having 9% and 19% loss with the same average latency produces the ReLate2 values of 10,000 and 20,000, respectively. ReLate2Jit is a product of the ReLate2 value and the jitter of the data packets to quantify the multiple QoS concerns of jitter, reliability, and average latency. Each of these composite metrics provide a single quantitative value that can be objectively compared when using different transport protocols in various operating environments.

Experiments Highlighting Need for Autonomic Adaptation

We now present the results of experiments for autonomic adaptation of the QoS mechanisms of transport protocols. We apply the composite metrics defined in the previous section to several different operating environments to highlight how differences in the environment trigger differences in the transport protocols used to support QoS. Figure 6 shows a change in the sending rate corresponds to a change in the protocol that provides the best QoS.

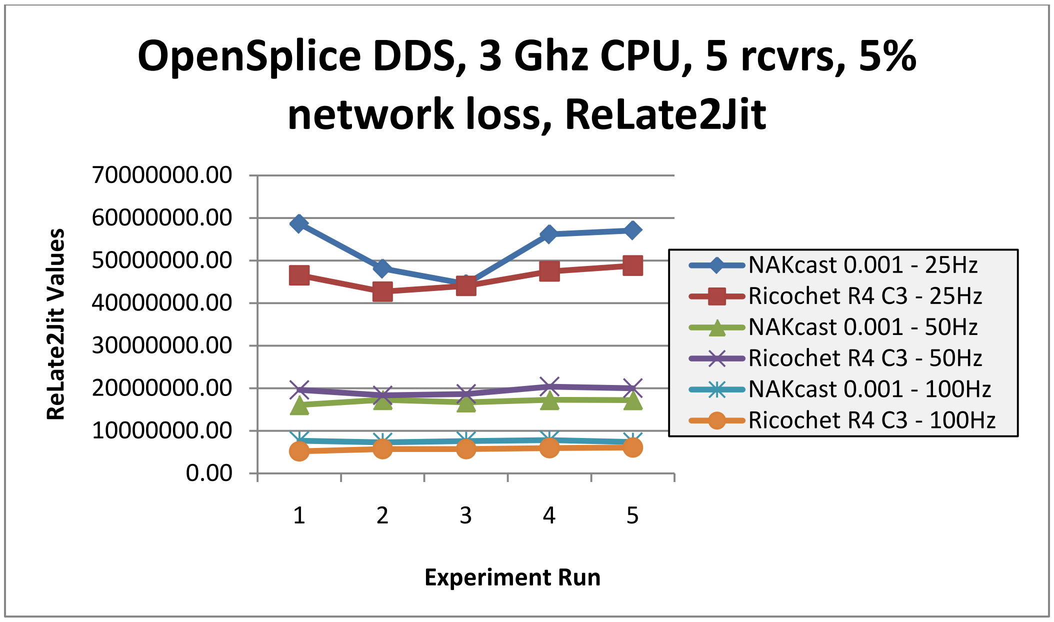


Figure 6: Changing Data Sending Rate

In particular, for an operating environment using the OpenSplice DDS implementation, machines with 3 GHz CPUs, 5 data receivers, and 5% network packet loss, we see that for a data sending rate of 25Hz, the NAKcast protocol (with a timeout parameter to determine NAK transmissions of 0.001 seconds) performs better (*i.e.*, has lower ReLate2Jit values) than Ricochet (with an *R* value of 4 and a *C* value of 3).

Ricochet's *R* value determines how many data packets are received before error correction data is sent (*e.g.*, 4 packets received before one error correction packet is sent) and Ricochet's *C* value determines how many other receivers this receiver sends error correct data (*e.g.*, 3 receivers receive error correction data from any one receiver). When the sending rate is changed to

50Hz, however, Ricochet performs better. Finally, when the sending rate is further increased to 100Hz NAKcast again performs better (*i.e.*, has lower ReLate2Jit values).

Timeliness of TIML

We next describe the timeliness of TIML as it decides the most appropriate transport protocol for a given environment configuration. As described in Challenge 2 (timely adaptation), the personal datacenter for the SCAAL application needs to have timely adaptations. We now provide timing information based on the responsiveness of TIML when queried for an optimal transport protocol. We used the Emulab configuration as described in the section *Experimental Setup*. A high resolution timestamp was taken right before and right after each call was made to TIML.

TIML combines and integrates the use of ANN and SVM machine learning techniques. These techniques present different response times (although the times for each technique remain constant). We therefore conducted experiments with operating environment configurations that would use the ANN (*i.e.*, the configurations that were known *a priori*) and configurations that would use the SVM (*i.e.*, the configurations that were unknown until run-time). Since these techniques provide constant-time performance, their compute times are invariant to the specific environment configuration, so we did not run timing test for all different environment configurations.

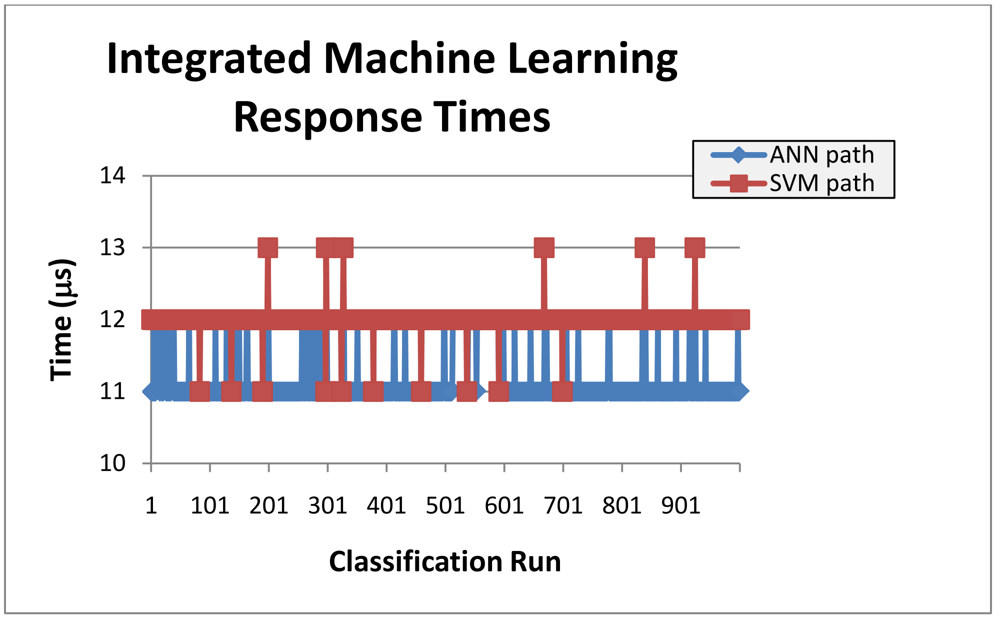


Figure 7: Integrated Supervised Machine Learning Response Times

Figure 7 presents the response times for TIML in ADAMANT for 1,000 iterations when TIML selects and uses either an ANN or the SVM. The figure highlights the times used within the integrated machine learning techniques when the environment configuration is (1) known *a priori* and thus triggers the use of an ANN and (2) unknown until run-time triggering the use of an SVM. On average TIML when using the ANN presents the lower response time of 11.161 µs while TIML using the SVM presents an average response time of 11.996 µs. The bound on TIML is then the maximum between the two (*i.e.*, 11.996 µs). The figure also appears to show that TIML using the ANN has more jitter than TIML using the SVM. The jitter is within the resolution of the timers (i.e., 1 µs) used for collecting the times, however, since the times only vary by +/- 1 µs from the median values (*i.e.*, 11 µs for the TIML when the ANN is used and 12 µs when the SVM is used).

Timeliness of ANT Reconfiguration

We now describe the experiments we conducted to show the timeliness of the ANT framework as it transitions from one transport protocol to another. As described in Challenge 2 (timely adaptation), the personal datacenter for the SCAAL application needs to have timely adaptations. In the previous section we presented timing results for determining the appropriate transport protocol. In this section we provide timing information on the reconfiguration of transport protocols supported in the ANT framework portion of ADAMANT. We used the same experimental environment as described in the section *Experimental Setup*. A high resolution timestamp was taken right before and right after each call made to ANT to reconfigure transport protocols.

Figure 8 shows the times taken for transport protocol reconfiguration across 1000 iterations. The figure includes times for three different scenarios. Two of the scenarios are most relevant for the transport protocols that best handle reliability and latency (*i.e.*, the NAKcast and Ricochet protocols). The third scenario presents a baseline when checks are performed to determine if a protocol transition is needed but no transition is needed.

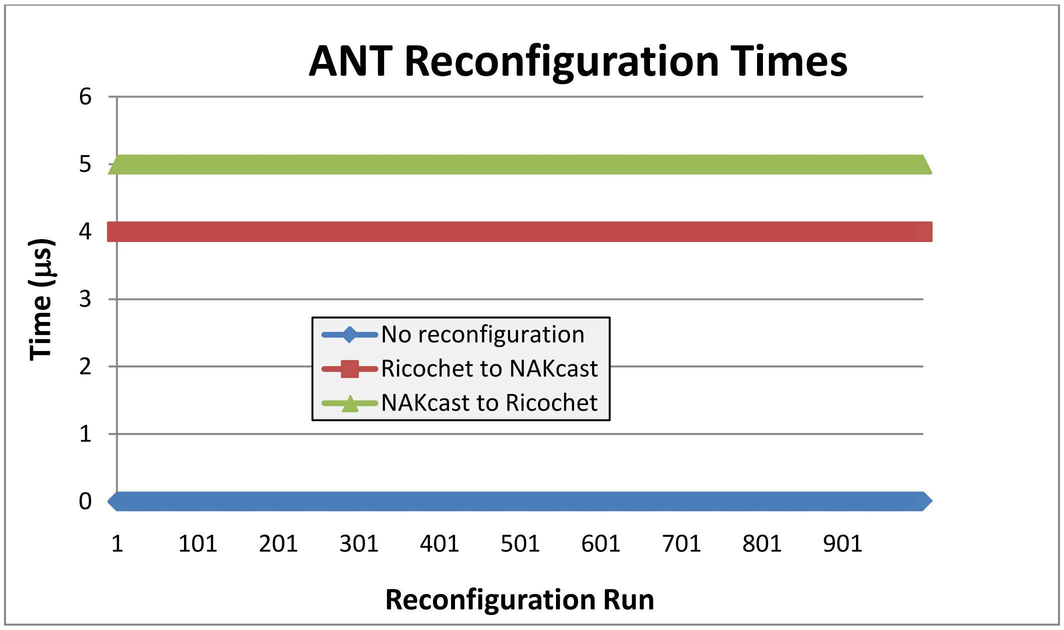


Figure 8: Transport Protocol Reconfiguration Times within ANT

The baseline times for no reconfiguration shows 0 µs taken to determine that no protocol reconfiguration is needed. Obviously, some time is taken to make the determination that no reconfiguration is needed but this time is smaller than the resolution of the timestamps taken (*i.e.*, < 1 µs). These times provide an idea of the overhead required in making any protocol reconfiguration.

The remaining two scenarios are when (1) the NAKcast protocol is running and a transition is made to the Ricochet protocol and (2) when the Ricochet protocol is running and a transition is made to the NAKcast protocol. The times for these transitions should be constant since, when reconfiguring, ANT registers a constant number of events and event handlers for the new protocol and unregisters a constant number of events and event handlers for the old protocol. The number of event and event handlers is known *a priori* at development time. Registering and unregistering events and event handlers correspond to inserting and removing items from a queue which are constant time operations.

It should be noted that the transport protocols that are a focus of this research do not have state that needs to be transferred from one protocol to another. Therefore, transitioning from one protocol to another did not require quiescence of the system for transferring state. There are trade-offs between having either both transport protocols running during the transition (*i.e.*, when transitioning from the NAKcast protocol to the Ricochet protocol, start up the Ricochet protocol before stopping the NAKcast protocol) or having neither protocol running during the transition (*i.e.*, when transitioning from the NAKcast protocol to the Ricochet protocol, stop the NAKcast protocol before starting up the Ricochet protocol. Running both protocols during the transition will temporarily increase reliability and processing while running neither protocol will decrease reliability and processing. Evaluating these trade-offs in detail is left as future work.

In particular, for the NAKcast and Ricochet protocols we know *a priori* the number and kinds of events and event handlers that each protocol uses. ADAMANT first unregisters all the relevant events and event handlers for an old protocol and then registers all the relevant events and event handlers for the new protocol. Since ADAMANT controls how and in what order events and event handlers are registered and unregistered in ANT, managing the lists for these events and event handlers can be done in constant time. The Ricochet to NAKcast transition consistently takes 4 μs while the NAKcast to Ricochet transition consistently takes 5 μs. For the ADAMANT prototype using the OpenSplice DDS implementation, these transitions are happening within the single network daemon per computing platform. As noted in section *Control Flow of ADAMANT*, ANT's transitions are deterministic with the DDS middleware ensuring that all the computing platform see the same updates and therefore make the same transitions. These empirical transition times verify that ANT protocol transitions are made in a constant amount of time.

Summary of Results

The results of experiments presented in this section show that there are scenarios where a change in the operating environment requires a change in the QoS mechanisms (*e.g.*, transport protocols) that ADAMANT is utilizing. Based on this information, the experiments show that ADAMANT delivers constant-time decision-making regarding the appropriate transport protocol to use as well as constant-time transitioning from one transport protocol to another. For QoS-enabled DRE pub/sub applications ADAMANT provides the constant-time complexity needed for detecting environment changes, determining the appropriate course of action, and executing that plan.

RELATED WORK

This section compares our work on ADAMANT with related work.

Support for adaptive middleware.

The Mobility Support Service (MSS) (Caporuscio, 2003) provides a software layer on top of pub/sub middleware to enable end host mobility. The purpose of MSS is to support the movement of clients between access points of a system using pub/sub middleware. In this sense, MSS adapts the pub/sub middleware used in a mobile environment. Mobile clients notify MSS when mobility starts and ends. MSS buffers messages and manages connections while the client moves to a different access point. MSS is designed to support multiple pub/sub technologies, *e.g.*, implementations of JMS, and adapt to the technology-specific characteristics.

MSS is solely focused on supporting mobility of pub/sub, however, and therefore does not address Challenge 1 (managing interacting QoS). Moreover, MSS is not focused on DRE systems and therefore does not address Challenge 2 (timely adaptation). Finally, MSS does not address operating environments where aspects of the environment change and impact QoS.

Gridkit (Grace, 2005) is a middleware framework that supports reconfigurability of applications dependent upon the condition of the environment and the functionality of registered components. Gridkit focuses on grid applications which are highly heterogeneous in nature. For example, these applications will run on many types of computing devices and across different types of networks.

To register components, application developers use Gridkit's API, which is based on binding contracts. Gridkit then uses the contract information along with a context engine to determine which components to include in the application. The context engine takes into account the context of the host machines, *e.g.*, battery life, network connectivity.

Gridkit focuses on reconfiguration for installing an application and does not address Challenge 2 (accurate adaptation) or Challenge 3 (timely adaptation) for systems operating in dynamic environments. Within Gridkit no consideration is given to making timely and accurate adaptations based on the environment changing for a single application installation. Moreover, Gridkit fails to address Challenge 1 (managing interacting QoS) as it does not address QoS concerns.

David and Ledoux have developed SAFRAN (David, 2006) to enable applications to become context-aware themselves so that they can adapt to their contexts. SAFRAN provides reactive adaptation policy infrastructure for components using an aspect-oriented approach. SAFRAN follows the structure of a generic AOP system by supporting (1) a base program which corresponds to a configuration of components, (2) point-cuts which are invoked in response to internal events (*e.g.*, invocations on interfaces) and external events (*e.g.*, change in system resources), (3) advices which define functionality to be executed for point-cuts, and (4) adaptation which uses adaptation policies to link join points to advices.

The SAFRAN component framework, however, only provides development support of maintaining specified QoS. The adaptive policies and component implementation are the responsibility of the application developer. Therefore, SAFRAN does not address Challenge 2 (accurate adaptation) and Challenge 3 (timely adaptation).

Machine learning in support of autonomic adaptation.

Vienne and Sourrouille (Vienne, 2005) present the Dynamic Control of Behavior based on Learning (DCBL) middleware that incorporates reinforcement machine learning in support of autonomic control for QoS management. Reinforcement machine learning not only allows DCBL to handle unexpected changes but also reduces the overall system knowledge required by the system developers. System developers provide an XML description of the system, which DCBL then uses together with an internal representation of the managed system to select appropriate QoS dynamically.

DCBL's use of reinforcement learning, however, does not address Challenge 3 (timely adaptation) as reinforcement learning is unbounded in its time complexity. DCBL also focuses on single computers rather than addressing scalable DRE systems and therefore does not address Challenge 4 (development complexity) for DRE systems. Moreover, DCBL requires developers to specify in an XML file the selection of operating modes given a QoS level along with execution paths, which leaves handling Challenge 1 (managing interacting QoS) to developers.

Tock *et al.* (Tock, 2005) utilize machine learning for data dissemination in their work on Multicast Mapping (MCM). MCM hierarchically clusters data flows so that multiple topics are mapped onto a single session and multiple sessions are mapped onto a single reliable multicast group. MCM's approach manages the scarce availability of multicast addresses in large-scale systems. MCM leverages machine learning to adapt as user interest and message rate change during the day. MCM is designed only to address the scarce resource of IP multicast addresses in large-scale systems, however, rather than Challenge 1 (managing interacting QoS).

Autonomic adaptation of service level agreements.

Herssens *et al.* (Herssens, 2008) describe work that centers on autonomically adapting service level agreements (SLAs) when the context of the specified service changes. This work acknowledges that both offered and the requested QoS for Web services might vary over the course of the interaction and accordingly modifies the SLA between the client and the server as appropriate. However, this work does not address Challenge 2 (accurate adaptation) or Challenge 3 (timely adaptation) in dynamic environments but rather negotiates the QoS agreement to fit the dynamic environment.

Autonomic adaption of networks.

The Autonomic Real-time Multicast Distribution System (ARMDS) (Brynjulfsson, 2006) is a framework that focuses on decreasing excessive variance in service quality for multicast data across the Internet. The framework supports the autonomic adaptation of the network nodes forming the multicast graph so that the consistency of service delivery is enhanced. However, ARMDS does not address Challenge 1 (managing interacting QoS).

CONCLUDING REMARKS

Developers of systems that utilize DRE pub/sub middleware face a number of challenges when developing and deploying their systems in dynamic environments. To address these challenges, we developed ADAMANT to integrate and enhance (1) QoS-enabled pub/sub middleware, (2) an environment monitoring topic, (3) a flexible transport protocol framework, (4) a novel integration of supervised machine learning techniques, and (5) an autonomic controller to provide fast and predictable reconfiguration of middleware and transport protocols for enterprise DRE pub/sub systems. This paper presents the results of experiments that show how ADAMANT can adapt autonomically to changing conditions in operating environments to support QoS in a fast, constant-time, and accurate manner.

The following is a summary of lessons learned from our experience evaluating ADAMANT's autonomic adaptation performance in various operating environments:

- **Several trade-offs exist when using machine learning in dynamic environments.** There are several trade-offs between having machine learning that (1) is completely accurate for environments known at training time, (2) highly accurate for environments unknown until run-time, (3) can accommodate new data on which to train as the system is running, and (4) can expend the appropriate amount of time interactively training machine learning tools while the system is running. Since overfitting an ANN to environment configurations known *a priori* provides perfect accuracy and low response times, it is preferable to incorporate new operating environment configurations unknown until runtime into the ANN training set while the system is running. A low-priority thread could be used to constantly re-train the ANN and swap in the updated ANN at appropriate times. While this approach would incorporate new environment configurations, our future work is addressing trade-offs between when to migrate to using the updated ANN versus how to determine the importance of the low-priority training thread so it will not be starved.

- **Preparing environment information for use in machine learning tools is time consuming and tedious.** A large amount of data can be generated based on the number and types of environment variables (*e.g.*, number of receivers, data sending rate, percent loss in the network, CPU speed, QoS metric used). In addition to the raw data, the data can be scaled (*i.e.*, transformed to be within minimum and maximum values such as between -1 and 1). Some machine learning techniques provide better results when the data are scaled. For our experimental data, scaling the data produced the best results. When scaling the data produces the best results, the environment data received from ADAMANT's monitoring topic also needed to be scaled as well which adds some overhead and complexity. The scaling factors used on the data for training the machine learning tools need to be managed and applied to the data collected from the environment during runtime.
- **Multiple machine learning approaches can be integrated to handle configurations known *a priori* and environment configurations not known until runtime.** Some machine learning techniques provide higher accuracy than others for operating environments known *a priori*. In particular, ANNs can be overfitted to the data to provide 100% accuracy for these kinds of environments. Other techniques provide higher accuracy for environments unknown until runtime. An integration of multiple machine learning techniques can provide higher overall accuracy than can be provided by any single machine learning technique. If timeliness is a concern, then when integrating multiple techniques, care must be taken to ensure that the integration itself does not change the time complexity characteristics. ADAMANT incorporates TIML to increase its overall accuracy for both operating environments known *a priori* and environments unknown until runtime while also ensuring that the integration itself maintains the constant-time complexity needed by DRE systems.
- **Transport protocols need to be selectively used based on the QoS specified.** While several DDS implementations (*e.g.*, OpenDDS and OpenSplice) provide pluggable transport frameworks to leverage standard and custom transport protocols the properties of these protocols must be dictated by application-specified QoS policies. The focus on transport protocols is a starting point for research into classifying QoS mechanisms (which is a far richer space than just transport protocols) and mapping application specified QoS policies to these QoS mechanisms in an automated way. For example, in our work we wanted to specify that the environment monitoring topic information be sent and received reliably. DDS implementations, however, provide no infrastructure for mapping between the transport protocols (*e.g.*, Ricochet and NAKcast) used and the QoS properties (*e.g.*, reliable data communication, or "best-effort") specified. Our future work is therefore developing a transport protocol taxonomy that QoS-enabled middleware can leverage to determine which protocol to apply

based on QoS specified at the application level using DDS QoS policies. The properties that transport protocols provide can be used to classify the protocols with respect to QoS. The middleware can then select (1) the most appropriate transport protocol based on the QoS properties needed and (2) different transport protocols for different QoS properties.

- **QoS-enabled middleware provides a fairly coarse-grained approach to reliability.** Utilizing transport protocols such as Ricochet and NAKcast allows QoS-enabled middleware to provide finer-grained reliability as well as considering latency. Reliability is typically only supported, however, as the binary choice of either best-effort reliability (which implies no reliability support) or *perfect reliability* (which implies reliability support for every individual network packet) with no consideration of highly probabilistic reliability (where the probability of receiving any one network packet is high but is not intended to be 100%) . Moreover, the semantics of combining multiple QoS aspects (*e.g.*, reliability and latency) are not clearly defined at the middleware level. Transport protocols, such as Ricochet and NAKcast, capture the finer-grained reliability property of high probability of reliability and low data latency which need to be specified, for example, in applications transmitting multimedia data.
- **High-level metrics are useful to quickly differentiate the performance of various configurations.** The use of metrics—even coarse-grained metrics—helps explore a large configuration space efficiently. Developing composite metrics (*e.g.*, ReLate and ReLate2) helps ameliorate navigating a configuration space with several points of variability.

All the source code and documentation for ADAMANT is available in open-source form at www.dre.vanderbilt.edu/~jhoffert/ADAMANT.

REFERENCES

Bai, Y., & Ito, M. (2007). A new technique for minimizing network loss from users’ perspective. *Journal of Network Computing Applications, 30*(2), 637–649.

Bai, Y., & Ito, M. (2006). A study for providing better quality of service to VoIP users. In *20th International Conference on Advanced Information Networking and Application* (pp. 799–804). Washington, D.C.: Lecture Notes in Computer Science.

Balakrishnan, M., Birman, K., Phanishayee, A., & Pleisch, S. (2007). Ricochet: Lateral error correction for time-critical multicast. In *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation* (pp. 73 – 86). New York: ACM Press.

Balakrishnan, M., Pleisch, S., & Birman, K. (2005). Slingshot: Time-critical multicast for clustered applications. In *Fourth IEEE International Symposium on Network Computing and Applications* (pp. 205-214). New York: ACM Press.

Brodnik, A., Munro, J. (1994). Membership in constant time and minimum space. In *Algorithms - ESA '94* (pp. 72 – 81). Berlin / Heidelberg: Springer LNCS.

Brynjulfsson, B., Hjalmtysson, G., Katrinis, K., & Plattner, B. (2006). Autonomic network-layer multicast service towards consistent service quality. In *20th International Conference on Advanced Information Networking and Applications* (pp. 494–498). Los Alamitos, CA: IEEE Computer Society.

Caporuscio, M., Carzaniga, A., & A. Wolf. (2003) Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering, 29*(12), 1059–1071.

Chandy, M., Etzion, O., von Ammon, R., & Niblett, P. (2007) “07191 summary – event processing,” presented at Event Processing, ser. Dagstuhl Seminar Proceedings, M. Chandy, O. Etzion, and R. von Ammon, Eds., no. 07191. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

David, P.-C., & Ledoux, T. (2006). An aspect-oriented approach for developing self-adaptive fractal components, In W. Löwe & M. Südholt (Eds.) *Software Composition* (pp. 82–97). Berlin/Heidelberg : Springer LNCS.

Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM Computing Surveys, 27*(3), 326 – 327.

Grace, P., Coulson, G., Blair, G., & Porter, B. (2005). Deep middleware for the divergent grid. In *Middleware ’05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware* (pp. 334–353). New York: Springer-Verlag New York, Inc.

Herssens, C., Faulkner, S., & Jureta, I. J. (2008). Context-driven autonomic adaptation of sla. In *Proceedings of the 6th International Conference on Service-Oriented Computing* (pp. 362–377). Berlin, Heidelberg : Springer-Verlag.

Hoffert, J., Schmidt, D., & Gokhale, A. (2010-A). Adapting and Evaluating Distributed Real-time and Embedded Systems in Dynamic Environments. In *1st International Workshop on Data Dissemination for Large scale Complex Critical Infrastructures* (pp. 23-28). New York: ACM Press.

Hoffert, J., Schmidt, D., & Gokhale, A. (2010-B). Adapting Distributed Real-time and Embedded Pub/Sub Middleware for Cloud Computing Environments. In I. Gupta & C. Mascolo (Eds.), *ACM/IFIP/USENIX 11th International Middleware Conference* (pp. 21-41). Berlin/Heidelberg : Springer LNCS.

Hoffert, J., Mack, D., & Schmidt, D. (2010-C). Integrating Machine Learning Techniques to Adapt Protocols for QoS-enabled Distributed Real-time and Embedded Publish/Subscribe Middleware. *Network Protocols and Algorithms, 2*(3), 37 – 69.

Hoffert, J. & Schmidt, D. (2009-A). Maintaining QoS for Publish/Subscribe Middleware in Dynamic Environments. In *3rd International Conference on Distributed Event-based Systems* (pp. 28:1-28:2). New York: ACM Press.

Hoffert, J., Gokhale, A., & Schmidt, D. (2009-B). Evaluating Transport Protocols for Real-time Event Stream Processing Middleware and Applications. In R. Meersman, T. Dillon, & P. Herrero (Eds.), *11th International Symposium on Distributed Objects, Middleware, and Applications*. Berlin/Heidelberg : Springer-Verlag.

Huang, Q., Freedman, D., Vigfusson, Y., Birman, K., & Peng, B. (2010). Kevlar: a flexible infrastructure for wide-area collaborative applications. In I. Gupta & C. Mascolo (Eds.), *ACM/IFIP/USENIX 11th International Middleware Conference* (pp. 148 – 168). Berlin: Springer-Verlag.

Huang, Y. & Gannon, D. (2006). A comparative study of web services-based event notification specifications. In *International Conference on Parallel Processing Workshops* (pp. 7–14). Los Alamitos : IEEE Computer Society.

Huebscher, M. & McCann, J. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys, 40*(3), 7:1 – 7:28.

Kephart, J. & Chess, D. (2003). The vision of autonomic computing. *Computer, 36*(1), 41 – 50.

Liu, Y. (2006). Create stable neural networks by cross-validation. In *International Joint Conference on Neural Networks* (pp. 3925 – 3928). Los Alamitos : IEEE Computer Society.

Meyer, D., Leisch, D., & Hornik, K. (2003). The support vector machine under test. *Neurocomputting, 55*(1-2), 169 – 186.

Monson-Haefel, R., & Chappell, D. (2000). *Java Message Service*. Sebastopol, CA: O'Reilly & Associates, Inc.

Niblett, P., & Graham, S. (2005). Events and service-oriented architecture: the OASIS web services notification specifications. *IBM Systems Journal, 44*(4), 869 – 886.

Ngatman, M., Ngadi, M., & Sharif, J. (2008). Comprehensive study of transmission techniques for reducing packet loss and delay in multimedia over ip. *International Journal of Computer Science and Network Security, 8*(3), 292–299.

Pardo-Castellote, G. (2003). OMG data-distribution service: architectural overview. In *23rd International Conference on Distributed Computing Systems* (pp. 200 – 206). Los Alamitos : IEEE Computer Society.

Patterson, D.W. (1998). *Artificial Neural Networks: Theory and Applications.* Upper Saddle River, NJ: Prentice Hall PTR.

Ramani, S., Trivedi, K.S., and Dasarathy, B. (2001). Performance analysis of the CORBA notification service. In *20th IEEE Symposium on Reliable Distributed Systems* (pp. 227 – 236). Los Alamitos : IEEE Computer Society.

Schmidt, D. C. (2000-A). GPERF: a perfect hash function generator. In R. Martin (Ed.), *More C++ Gems* (pp. 461 – 491). New York, NY: Cambridge University Press.

Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000-B). Pattern-oriented software architecture: patterns for concurrent and networked objects, Volume 2. New York, NY: Wiley & Sons.

Tarkoma, S., & Raatikainen, K. (2006). *State of the art review of distributed event systems* (Tech. Rep. C0-04). Helsinki: University of Helsinki.

Tock, Y., Naaman, N., Harpaz, A., & Gershinsky, G. (2005). Hierarchical clustering of message flows in a multicast data dissemination system. In S. Q. Zheng (Ed.), *International Conference on Parallel and Distributed Computing Systems* (pp. 320 – 326). Phoenix : IASTED/ACTA Press.

Vienne, P. & Sourrouille, J.-L. (2005). A middleware for autonomic QoS management based on learning. In *5th International Workshop on Software Engineering and Middleware* (pp. 1–8). New York : ACM Press.