ISeB (2006) 4: 399–419
DOI 10.1007/s10257-005-0012-7

ORIGINAL ARTICLE

Ahmad T. Al-Hammouri · Wenhui Zhang
Robert F. Buchheit · Vincenzo Liberatore
Panos K. Chrysanthis · Kirk Pruhs

Network awareness and application adaptability

Published online: 9 June 2006

Abstract In this paper, we argue that pervasive applications need to be aware of and adaptive to network conditions. We begin with an overview of three software projects in which we are currently involved, and highlight network awareness and application adaptability as a common thread among them. We argue that these features stem from the fundamental architectural principles of the Internet. We generalize our experience and elaborate on the principles for developing network awareness and adaptable applications.

1 Introduction

Pervasive computing enables us to access information from anywhere and to affect local and remote physical environments. The Internet is a powerful medium on which pervasive applications can be developed and deployed.

This work has been supported in part under NSF grants ANI-0123929, CCR-0098752, and CCR-0329910.

A. T. Al-Hammouri (✉) · W. Zhang · R. F. Buchheit · V. Liberatore
Department of Electrical Engineering and Computer Science,
Case Western Reserve University, 10900 Euclid Av.,
Cleveland, OH, 44106, USA
E-mail: ata5@case.edu
E-mail: vincenzo.liberatore@case.edu

P. K. Chrysanthis · K. Pruhs
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, 15260, USA
E-mail: panos@cs.pitt.edu
E-mail: kirk@cs.pitt.edu

However, most of the current Internet lack features that are critical to pervasive applications, such as multicast or Quality-of-Service (QoS) provisioning. As a result, applications need to exhibit particular intelligence when deployed over IP (Internet Protocol) networks. In this paper, we argue that network-enabled software needs to become *network-aware*, i.e., knowledgeable about the technical details of the network, and *network-adaptive*, i.e., responsive to the nature and behavior of the network. We expound the principles that are critical in developing network awareness and adaptable applications.

The importance of network awareness and adaptability can be examined in many pervasive applications. For example, VoIP (Arora 1999) enables telephony using Internet connectivity instead of plain old phone lines. When two people have a phone conversation, voice data should be transmitted to them in real time. Being aware that the Internet cannot provide real-time communication service, developers implemented some protocols, such as RTP (Schulzrinne et al. 1996), RSVP (Braden et al. 1997), to achieve real-time transmission so that VoIP applications can adapt to the Internet environment. In this paper, we report on three software projects, in which we are involved, that are network aware and are adaptable. These three projects are a scalable data dissemination middleware, an agent-based robotic control framework, and a distributed simulation platform. Although these applications differ greatly in objectives and functionalities, they share the common trait that their design acknowledges explicitly the network architectural principles, functionality, and missing features. In other words, these applications are network-aware, network-adaptive, or both. We will argue that network awareness and network adaptability are motivated by the fundamental design principle of the Internet—they are not a transient phenomenon. Based on the generalization of our experience, we will argue that network environment investigation should be an additional step within the existing software development methodologies (Pfleeger 1998; Schach 1996) and pervasive applications should be configurable to enhance adaptability.

The paper is organized as follows. In Sect. 2, we report on our implementation of the data dissemination software. In Sect. 3, we describe our agent-based architecture for remote robotic control, and our framework for distributed and physically realistic simulations. Section 4 puts awareness and adaptability within the broader perspective of the Internet design principles. In Sect. 5, we give a highlight to the principles that are crucial in developing effective pervasive applications with network awareness and adaptability. Section 6 discusses a common approach to achieve self-awareness and self-adaptability at runtime. Finally, Sect. 7 concludes the paper.

2 Scalable data dissemination

An HealthCare ALert (HCAL) system is an application that collects data from health operators, assesses whether there is a widespread health conditions, and, when a disease outbreak is detected, disseminates critical information to a large number of receivers. Examples of HCAL is the RODS system (Tsui et al. 2003; Wagner et al. 2003) and the analytic framework

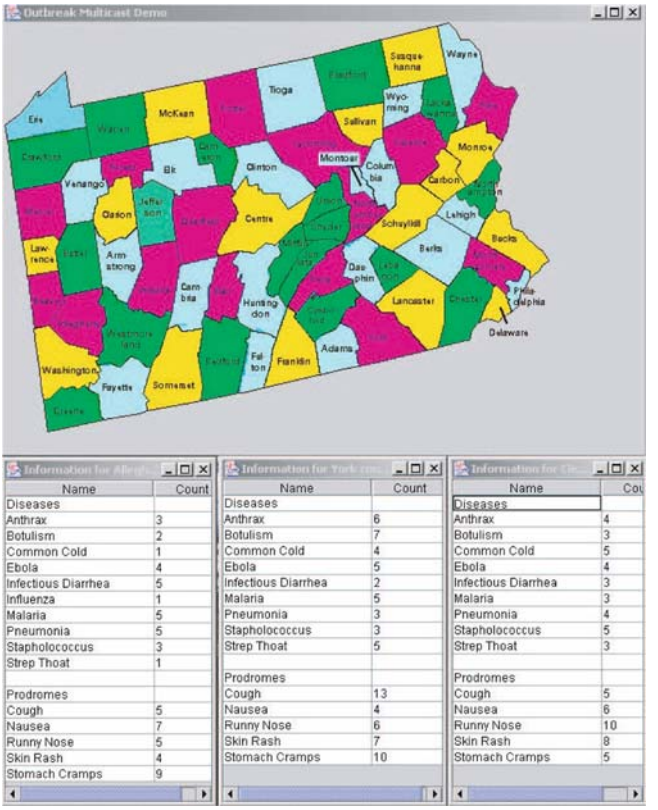

Fig. 1 A graphical user interface of the RODS HCAL system showing a map of the state of Pennsylvania, the number of diseases and prodromes detected in three selected counties during a run of the software on fictitious data

described in Buckeridge et al. (2003). Figure 1 shows a Graphical User Interface from the RODS HCAL at the client side (Li et al. 2003). The user is presented with a map of Pennsylvania broken down by counties. The purpose of this interface is to convey information to health operators and to the public regarding current disease outbreaks in the state. By clicking on a county, the user can get current statistics of diseases and prodromes for that county. The information is automatically refreshed when the client receives updated data from the server.

The data dissemination component is particularly needed when severe outbreaks are detected and health operators or the public urgently demand current information. In these circumstances, the demand for information would vastly exceed the computational capabilities of any single Web server and of most interconnection links. Therefore, two fundamental issues are:

1. The preparedness of the software to work in emergency scenarios,
2. Its scalability, i.e., its ability to serve a large number of potential requests for information.

Hence, the software developer should be aware of the network conditions and employ advanced technologies to improve system scalability. Lack of network awareness can cause a failure in developing an effective HCAL.

These objectives can be achieved through multicast communication, whereby a single piece of information is delivered simultaneously to all parties interested in monitoring health conditions (and only to them). Internet multicast is in principle similar to a television broadcast, which is aired by one station and received simultaneously and independently by a multitude of television sets. Multicast brings similar ideas to the Internet. However, allowing multicast in the network introduces non-trivial data management problems such as caching, consistency, and scheduling. We have built a middleware (MSMDD 2004; Chrysanthis et al. 2003; Li et al. 2003; Mahmoud 2004) which integrates state-of-the-art data dissemination and multicast communication techniques into a software distribution. Although the user interface appears on the surface as no more than a regular Web page, the system incorporates multicast through a middleware library and is consequently prepared for scalable use. For a better understanding of how the system works, now we give a description of middleware support for multicast-based data dissemination (MBDD) (Chrysanthis et al. 2003).

2.1 Data dissemination methods

The middleware supports three data dissemination methods, i.e., multicast push, multicast pull and unicast pull. The middleware can use any combination of these three multicast methods at run time. Applications cannot distinguish which method(s) are used because the details of the middleware are concealed from them. In multicast push, the server periodically multicasts data to clients. Clients join the multicast channel to retrieve data without explicitly sending requests to the server. In this way, the load on the server is reduced and the system scalability can be enhanced significantly. Multicast push is a good fit for disseminating hot data, e.g., the most up-to-date information requested by most clients of an HCAL during a disease outbreak. In multicast pull, the client sends an explicit request to the server when it needs some data. The server multicasts the requested data to all members of the multicast group. When there are multiple clients requesting the same data within a short time, the server can aggregate the requests and multicast the information only once. Multicast pull is used to disseminate warm data for which multicast push cannot be justified, while there is an advantage in aggregating concurrent client requests. Cold data are disseminated by traditional unicast pull. For instance, in an HCAL, some history records may be requested sporadically and consequently, unicast is suitable for delivering these data to the clients.

2.2 Architecture

Figure 2 shows the architecture of the middleware and its relationship with the transport and application layer. The architecture is flexible and extensible

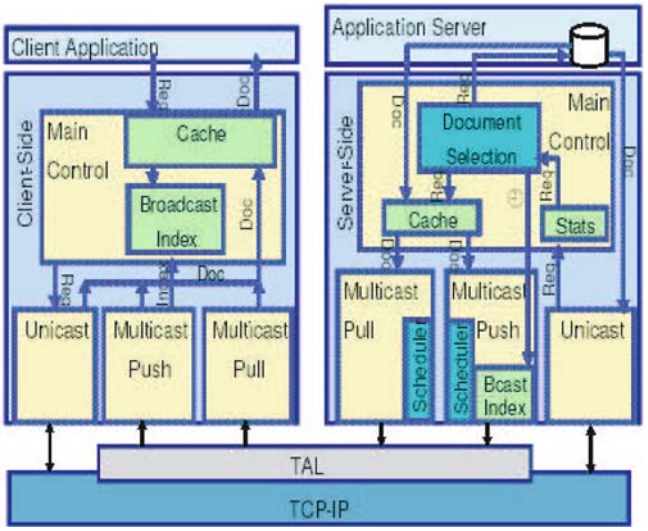

Fig. 2 Middleware data dissemination architecture and its relationship with the transport and application layers

and its components can be selected or replaced depending on the underlying multicast transport mechanism or on the application needs. The document selection component gathers statistics on document (a data item identified by identifiers such as URLs) popularity, which is the probability that clients request a document. The component uses this statistic to classify the documents as hot, warm or cold so that they can be disseminated respectively through multicast push, multicast pull and unicast pull. An index of sorted URIs or URI digests is broadcast by the server, which enables clients to quickly decide if a document is multicast by the push channel. The multicast push scheduling component determines the frequency and order by which hot documents are multicast. The multicast pull scheduling component resolves contention among client requests to use the multicast pull channel and arranges the order in which warm documents are disseminated. The multicast documents on the server are loaded into the cache before being disseminated to clients. Caching is also available at the client side. Advanced caching policies are incorporated into the caching component. Interaction between the middleware and the underlying transport layer are enabled by the transport adaptation layer (TAL) using a uniform interface.

2.3 Multicast schemes

The transport layer provides various multicast schemes for multicast communication. A multicast scheme can have specific requirements on the network where it works. This requires developers have a good knowledge of the network where the multicast system is deployed.

There are multiple multicast schemes that have been developed, ranging from IP multicast and end-to-end multicast schemes. In IP multicast, the server sends a single data copy to the network and packets are replicated and forwarded to receivers by multicast-enabled routers. Therefore, the server can disseminate data to a large number of receivers by using small bandwidth. Based on IP multicast, more sophisticated schemes are obtained by adding higher level features, such as reliability, congestion control and security. However, IP multicast has not been widely deployed because of a number of challenging issues, such as the management of globally unique addresses, maintaining per group state at routers, defending from flooding attacks and needs for changes at the infrastructure level. In practice, IP multicast is often confined within local-area networks.

Implemented at the application layer, end-to-end multicast schemes give an effective solution to deploy multicast in the Internet. In end-to-end multicast, applications are strategically placed in a logical overlay network, which is a network abstraction on top of the substrate network. Communication between the applications is along overlay edges using unicast. End-to-end multicast can utilize features such as flow control, congestion control and reliable delivery that are available for point-to-point connections. However, this approach can increase server-to-client delays because packets are relayed by applications. End-to-end multicast also requires more bandwidth than IP multicast since the same data copies can be transmitted over the same physical link more than once.

The selection of multicast scheme for a multicast system is determined by a number of factors: the size of the multicast group, the performance requirements, the network environments where the system is deployed, and other requirements on security, reliability. Network awareness can play an important role in obtaining the correct choice of a multicast method. For example, because HCALs must be deployed across heterogeneous Internet links, an end-to-end multicast scheme should be chosen by the application.

2.4 Performance

The objective of MBDD is to provide a scalable data management layer to applications. Therefore, it is important to optimize the application-perceived level of service. The application level performance often cannot be derived directly from the network level metrics. For example, a small loss rate on the links may not largely reduce the bandwidth available to clients, but it can substantially increase the latency to satisfy a client's request (Zhang et al. 2004). However, we can still take advantage of network awareness. If we learn that a network can experience severe loss of packets, we can selectively incorporate coding techniques (Luby et al. 1997) into applications to avoid remarkable deterioration of application level performance due to packet loss. In this way, applications can be more adaptive to networks with higher loss rates.

3 Remote control

In the case of HCALs, we have argued that network awareness allows software developers to identify what functionality the network lacks and what corresponding technologies should be adopted by applications to adapt to the network environment. We continue our argument on the importance of adaptability and network awareness by the discussion of remote control. We first describe the difficulties we face in the network, then we give two examples: Internet robotics and distributed simulations.

3.1 Quality-of-Service

A network can introduce unreliable and time-dependent levels of service in terms of, for example, delays, jitter, or losses. QoS can ameliorate the real-time network behavior, but the network behavior is still subject to interference (especially in wireless media), to routing transients, and to aggressive flows. In turn, network vagaries can jeopardize the stability, safety, and performance of controlled units in a physical environment. Therefore, a primary objective is to devise algorithms to compensate for the vagaries of network service. Such strategies are targeted toward the application layer and their objective is to deal with packet losses, delays, jitter, and network unreliability.

3.2 Internet robotics

The broad vision of Internet robotics is to enable end-users to affect a remote physical environment through network connectivity and action units. Applications range from distributed instrumentation (Al-Hammouri et al. 2003) to home robotics (Ngai et al. 2002). We have implemented a distributed system to facilitate human-robot remote interaction. Our approach is based on three concepts, all of which involve the system awareness of and adaptiveness to network conditions.

First, the system employs software components (agents) that can move among hosts in response to changes in the robotic tasks and in network conditions. Consider the snapshot shown in the upper part of Fig. 3 where agents A_1 and A_2 start communicating while residing in two different machines, A and B. Suppose at the beginning the two agents are satisfied with the communication quality. If, at some time later, network dynamics—such as congestion and routing paths-change, agents A_1 and A_2 may not be able to communicate in real time. However, agent A_1 can migrate to another machine where it can communicate effectively with agent A_2. On the extreme, A_1 will move to machine B, where A_2 resides as shown in the lower part of Fig. 3. Then, the two agents will execute as two threads and will communicate using inter-thread communication, which is far faster than communication over network. This extreme solution is not always feasible due to the computational requirements on host B, but in general, agent mobility

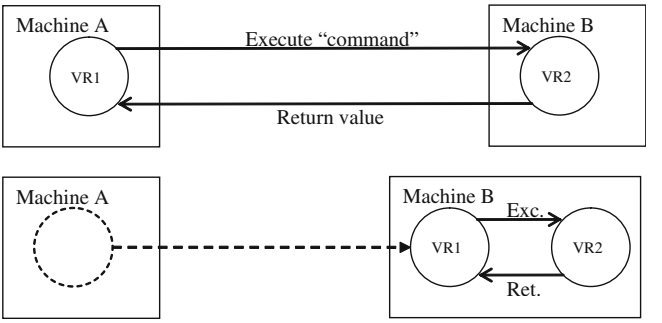

Fig. 3 Agent mobility. (*Up*) agent A_1 is in machine A; agent A_2 is in machine B. (*Down*) Both agents, A_1 and A_2, are in machine B

implements a high degree of awareness of and adaptability to end-to-end communication performance.

One of the core aspects of network awareness and adaptability is the detection and the recovery from partial failures. Agents' mobility can deal with planned disconnections. For example, if an agent realizes that its current machine is shutting down or the available computation resources are stepping down, it can move to another machine in the network with suitable resources. As for unexpected failures, we adopted the leases strategy from Jini (2004), which is based on the time-out idea. Agents join the system with a fixed lease duration. While agents are operational, they maintain their leases by renewing them. When an agent becomes unavailable because, for instance its network connection goes down, it will be unable to renew the lease and so the lease expires after a timeout. The system then cleans previous references of the unavailable agent and notifies other components via event notifications. By this process, the system arrives to a safe and predictable state and it continues to function correctly after failures.

Our system also relies on Jini's distributed events—the extension of local events over the network. Distributed events provide flexible asynchronous-communications. The necessity of deploying this stems from the asynchronous nature of these network applications. By registering to receive notifications about events that occur in the system, system entities will be kept updated with recent system conditions without constantly polling each other. Thus, reducing the amount of communication greatly.

Figure 4 shows a possible configuration of system components. In the figure, Supervisor is the user who interacts with the robot; VS (Virtual Supervisor) is a GUI that facilitates interaction with other system components; Virtual Robots (VRx's) are the mobile agents; Robot Proxy (RP) is a thin legacy layer between diverse robot implementations and our distributed system; and Lookup Services (LUs), which are adopted from Jini, are directories of available VRs and RPs in the system. Such tree structure forms an overlay topology that must be reconfigurable, as we discuss next. On one hand, the logical interconnection between components are defined to be specialized, for example VR3 can only communicate with VR4, VR6 and RP4 but not with, say, VR7. On the other hand, the overlay nodes (the

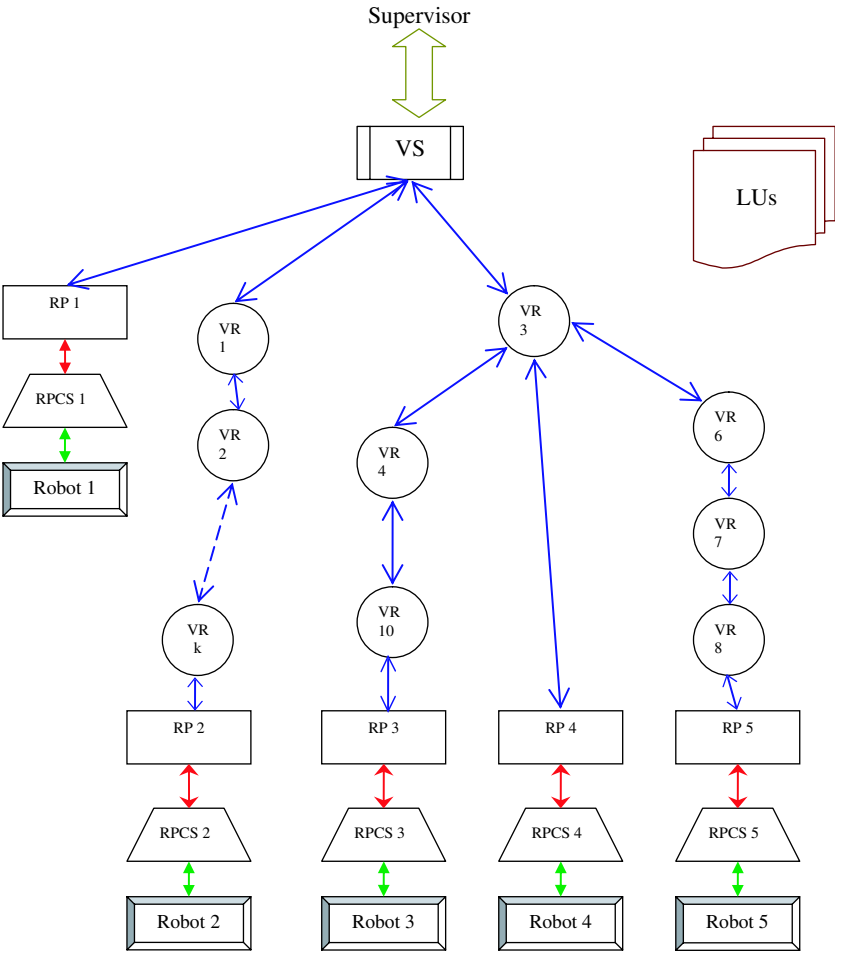

Fig. 4 An example of logical interconnection among software agents

software components) are distributed throughout the machines in the network, and the assignment of these nodes to the underlying physical network devices is reconfigurable, that is, it can change according to network conditions. These software components can be in different machines, all in one machine, or any intermediate configuration between these two extreme cases. In our ongoing research, we are investigating how to make the software components be aware of the communication quality and adaptable to network conditions by reconfiguration of the logical interconnection topology. For instance, if it happens that VR3 and VR8, in the topology of Fig. 4, operate in very close machines while VR6 and VR7 are in distant machines, VR6 and VR7 must become aware of this non-optimal physical placement and they must react accordingly, i.e., move very close to VR3 and VR8, if possible.

Figure 5 shows a current prototype: the window on the bottom left corner is a live WebCam view of the robot work space. The top window enables an end-user to directly tele-operate the robot through mouse movements and keyboard strokes. The bottom right window enables the creation of new mobile agents by directly instantiating the definitions contained in appropriate Java class files. This GUI also allows the logical interconnection of agents in a hierarchal structure such as the one in Fig. 4.

3.3 Distributed simulations

The environment can be located in the physical world, but it can also be numerically simulated. The result is a distributed interactive simulation, with applications ranging from medical training to networked video-games. However, the realism and effectiveness of networked virtual environments suffers in most current networks due to the network's lack of QoS guarantees. For example, poor network QoS can render the simulation unresponsive to user input, with consequent loss of user attention and of the overall effectiveness of the simulation. In these networks, privileged users can obtain high QoS levels at a substantial cost and show impressive network-enabled demonstrations. However, most other users do not have access to those exceptional QoS levels even if they have broadband connectivity, so that the effectiveness of network applications is hampered. Consequently, the current state of affairs prevents the widespread adoption of high-fidelity networked simulations for the large number of potential users with typical broadband connectivity.

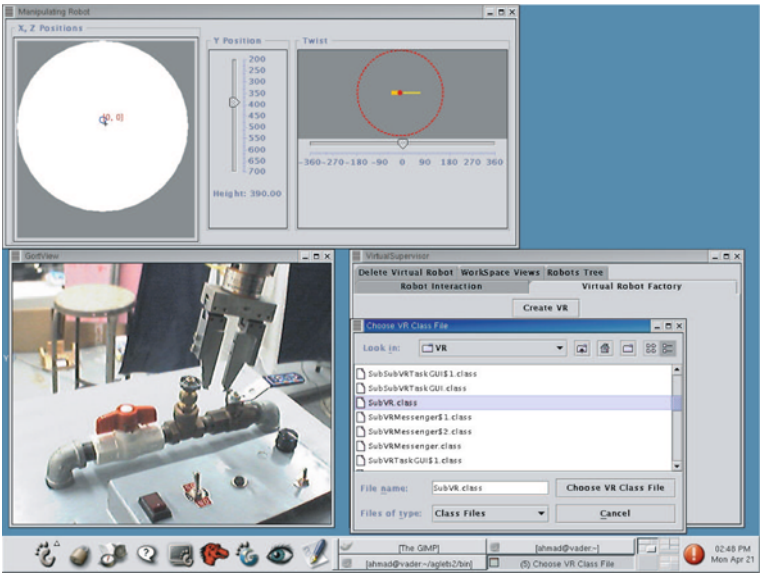

Fig. 5 VS GUI panel

Our project in this field seeks to explore the effectiveness of such applications through the development and implementation of an application-layer delay compensation strategy that must be aware of and adaptive to the lack of QoS. Figure 6 shows a network video-game that we have implemented. The game simulates a sailing boat and is designed to behave similarly as if the user is remotely operating an actual sailboat on the open ocean. The objective of the game is to sail toward the target buoy as quickly as possible given the environmental conditions (wind and current). The game is based on a client-server model. In addition to the control and update data stream pertaining to the game state, there is also a ping stream between the client and server. This is used by the client to maintain a running estimate of the latency between itself and the server. This estimate is then used to extrapolate and forward the game state by one RTT as in the strategy of dead reckoning (Smed et al. 2002b). To provide control feedback to the user, the short circuit mechanism is used (Smed et al. 2002a). Thus, in addition to sending the control inputs to the server they are also stored locally. The inputs are then compiled into a position offset for the vehicle and applied to the incoming state data before it is displayed to the user.

Each of the data points in Fig. 7 are based on the time performance of two identical automatic pilots on a fixed course consisting of a single buoy 1,000 feet distance to which the agents race. The data points each represent the average of 15 separate trials under each testing condition. In each case, one agent runs at the client and is subject to the effects of the latency and any delay compensation. The other agent runs at the server and experiences no latency effects. These results demonstrate the effectiveness of the delay compensation in allowing for feedback-based sail control even at very high latencies.

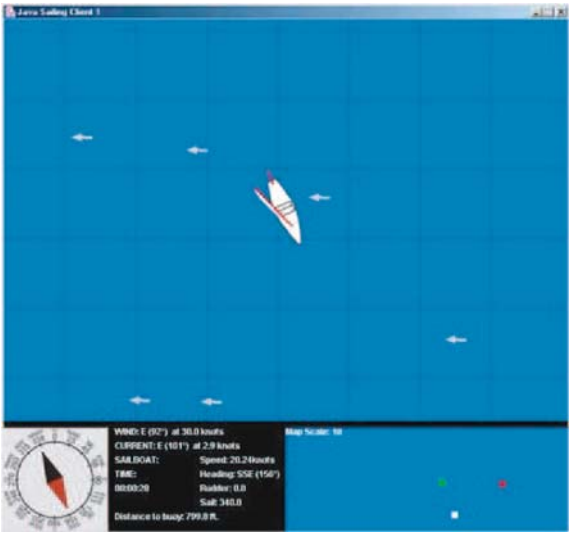

Fig. 6 Sailing game screenshot

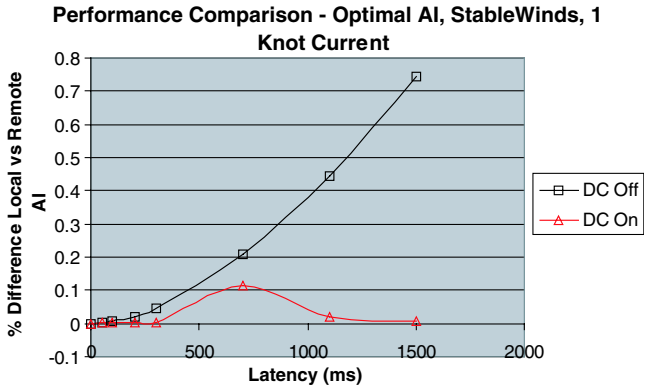

Fig. 7 Comparison of time performance of synthetic player with delay compensation and without

The framework developed for this game can provide a stable platform for further study of the field including potentially such other techniques as contingency control (Branicky et al. 2003) and server side latency reduction as well as improved game state convergence techniques.

4 Awareness and adaptability

Healthcare alert systems, Internet robotics, and distributed simulations are vastly different types of applications that share one common trait: they embed communication functionality that the network lacks. For example, an HCAL embeds multicast methods that are seldom supported in the Internet. The Internet robotics system embeds mobile agents for real-time communication. Similarly, distributed simulations embed methods to remediate poor network QoS. Multicast, real-time communication and QoS appropriately belong to the communication sphere and a software developer could reasonably expect that multicast, real-time communication and QoS should be provisioned within the network infrastructure, but the pragmatic reality is that such functions are not supported in the Internet. Therefore, the developer has the responsibility of knowing about these issues and of explicitly taking corresponding steps in his code. In this section, we will explore the causes of this fact.

4.1 The end-to-end argument

A fundamental design principle of the Internet is the end-to-end-argument. The argument states that a function should be pushed to the higher layers if possible, unless a lower layer implementation can achieve large performance benefits (Saltzer et al. 1984). The argument makes for a network that offers minimal functionality and expects additional components to be

implemented by the networked applications. For example, the argument implies that an HCAL should implement components for multicast. Similarly, Internet robotics systems should implement components for real-time communication and distributed simulation environments should implement components to tolerate low QoS. The end-to-end argument has been central to the success of the Internet: the network offers minimal functionality, but it also leaves developers free to create innovative applications. Internet capabilities are only limited by the inventiveness of the software developer. At the same time, however, the end-to-end argument requires the software developer to explicitly recognize the role of the network and to take steps to make his code network aware and adaptive. As such, awareness and adaptability are a permanent and structural component of Internet-enabled software. The end-to-end argument also makes network awareness extraneous to network research or to the infrastructure. As such, the argument implies that progress in network aware and adaptive technology would stem preferentially from end-applications rather than from core infrastructure research.

4.2 Awareness and adaptability

Lack of awareness and adaptability can lead to a "bubble" of inflated expectations and to its subsequent deflation when such expectations do not materialize.

4.2.1 A hypothetical scenario

In this hypothetical scenario, we consider a distributed simulation environment that is network-enabled (i.e., it can operate over a network) but not network-aware (i.e., it is not adaptive to network conditions). The software is deployed over a local-area network during initial usage and then ported to a wide-area grid context. The project is successful in the local context (the "bubble"), only to fail later on when the same software is used in the wide-area network (the deflation of the bubble). The resulting perception among most developers would probably be that simulations cannot be supported with current network technology. On the other hand, local-area and wide-area networks differ by order of magnitude in their characteristics, such as delays, bandwidth, and loss rates. The crux of the issue was really that the simulation was network-blind and consequently unable to cope with the heterogeneity of the local- and wide-area contexts.

We maintain that network awareness and adaptability are critical to the continuous development of effective applications. On the contrary, network unawareness leads to an intrinsically weaker software design that no amount of hardware resources can cure. We feel that network awareness and application adaptability should be part of the software development process as described in the next section.

5 Development of distributed applications

Network awareness and application adaptability can be realized using two different and complementary approaches. The first approach, discussed in this section, is to pursue additional steps during the software development cycle. The second approach, discussed in next section, is to build an adaptive middleware that monitors the network performance and then configures the application accordingly to meet application requirements. In this section, we generalize the lessons learned from the creation of our three examples, and elaborate on the principles that are crucial to develop applications with network awareness and adaptability.

5.1 Network environment investigation

Network awareness and application adaptability should receive significant attention in the software design phase. Network environment investigation should be an additional step within the existing software development methodologies (Pfleeger 1998; Schach 1996). This is because network awareness and adaptability are crucial factors in determining the software architecture and selection of technologies used to develop the application. In our examples, because the Internet lacks sufficient available bandwidth, HCAL employs multicast technology to meet the requirement of disseminating data to a large number of receivers. In turn, the use of multicast communication leads to the incorporation of a middleware into the HCAL architecture to deal with the problems introduced by multicast, such as caching, scheduling, interaction with various multicast schemes. Internet robotic systems require frequent communications and cooperations between software components, while the desired QoS cannot be guaranteed by the network. This scenario drives the developers to use the agents technology since agents can travel among the hosts in response to changes in the robotic tasks and in network conditions.

Generalized from our experience, the investigation procedure should have the following steps.

Information collection Combined with the analysis of the technical requirements (e.g., requirements on performance, scalability, reliability, security), information should be selectively collected from related materials, including manuals, technical reports and publications that are relative to the deployment network. Meanwhile, developers should look into the similar and typical distributed applications that have been deployed in the network. Developers need to list out the positive and negative features with their causes of these applications.

Analysis Based on the collected information, developers should classify the network situations that applications can experience during runtime. Developers need to identify the missed functionalities in the network that can cause applications to fail to meet the technical requirements. The analysis result contributes to the determination of what technologies and schemes should be

taken as solutions in the software development, which in turn affects the software architecture.

Prototyping It is desired that prototypes are created to affirm the analysis and evaluate the solution methods. The running environment can be the network where applications are deployed. Meanwhile, simulation (NS-2 2004) and emulation (Emulab 2004) environments are effective testing platforms because specific network conditions can be intentionally created. Prototyping enables developers to quantitatively evaluate the effectiveness of the proposed designs. Moreover, sophisticated testing tools, such as execution profiling (Podgurski et al. 2003; Varghese 2003; Xu et al. 2004), can be applied to explore unexpected problems in the network and trace their causes. The result of this step is the complementary information of the network environment. Modification and improvements are made on previous designs.

The above steps are repeatedly conducted to improve network awareness and application adaptability. Although most work of network investigation is done during the design phase, the investigation should proceed through the whole life of software development because new problems can arise due to the unawareness of some aspects of the network.

5.2 Configurability

Applications have been traditionally designed under the assumption that a desired network service level can be achieved through a signaling protocol. For example, an application would assume that bandwidth is reserved by running the RSVP (Barden et al. 1997) protocol. In reality, an application cannot manipulate a best effort network with distributed control. Moreover, networks are dynamic environments that can differ at different locations and times. We recognize these constraints and insist that applications should be configurable to enhance their adaptability to various network conditions.

5.2.1 Configurability of network parameters

First of all, applications should have the ability to set network parameters at the end-hosts in accordance to network conditions. Network parameter configuration is often conducted when an application is started. Settings should be either manually specified by the users based on their knowledge of the network environment or automatically set by the application based on its analysis of network conditions. Generally, the configuration aims to adapt to notable variations between networks, such as the different bandwidth, delays, scalability characters on different types of networks. The effectiveness of the configuration is closely related to the extent of network awareness. For instance, generally an HCAL server limits the length of a multicast message in UDP to 576 bytes (the minimum reassembly buffer size defined by IPv4)

to guarantee the datagram can be accepted by any client. However, if the server knows that the network is an IPv6 environment, it can set the maximum UDP datagram length to 1,280 bytes (Deering and Hinden 1998). As the transmission overhead is significantly reduced, the multicast performance can be remarkably improved.

Network conditions can vary during runtime. In many cases, applications should reconfigure some setting parameters during runtime in order to maintain satisfactory performance. For example, to maximize the throughput without triggering unacceptable network congestions, an HCAL server monitors the online available bandwidth of the network and modulates the server's multicast rate accordingly. We will elaborate on this point in Sect. 6.

5.2.2 Structural configurability

Pervasive applications should be structurally configurable. The software architecture should be flexible so that its individual components can be selected or replaced depending on the network conditions and application needs. In the HCAL example, the caching component in the middleware can be switched on or off and the type of multicast scheme can be selected by the application. In the Internet robotic systems, the logical interconnection topology of the system components (VSs, VRx's, and RPs) are configured according to network conditions.

While network parameter configurability is relatively easy to implement, structural configurability is more flexible and effective to improve application adaptability. Network parameter configuration modulates setting parameters in accordance to the network condition, but it does not change the algorithms in the software and the relationship between components. The adaptability achieved by parameter configuration is limited. For example, an HCAL using the IP multicast scheme, which works within local networks, cannot adapt to the Internet environment just by reconfiguring the settings in the application. To accomplish the task, the IP multicast scheme must be replaced by an end-to-end multicast scheme, which is a structural configuration.

The procedure of designing configurable components can be described as follows:

- Identify the functions that are used to achieve awareness and adaptability.
- Analyze the relationship between these functions. The relationship can be exclusive such as the multicast schemes in HCAL, or be cooperative such as the network communication functions of the VR and VS in the Internet robotics system.
- Organize these functions into a number of components and define the interface for those components that need to cooperate with each other.
- Design interface layers for these components to plug into the software architecture. A good example is the TAL in HCAL. Through TAL, the multicast schemes are selectively plugged into HCAL systems so that the data management modules can access the multicast channels with a uniform interface.

Distributed applications are extensible when they are structurally configurable. Two reasons make application extensibility important. Firstly, it is difficult to anticipate all the network conditions. Unpredictable network conditions can cause problems which require addition of new functions and modification of the applications. Secondly, with the evolution of technology, new algorithms and schemes need to be incorporated into the applications in order to enhance performance. When the applications are extensible, new functions and schemes can be added to the software architecture with few changes on the existing code. For an application with structural configurability, extensibility is achieved by adding new components, replacing old components, or restricting modification to a few components. In HCAL, for example, new multicast scheduling algorithms, multicast schemes, caching schemes can be easily incorporated into the application because the configurability and extensibility were taken into consideration strategically. Another example, in the Internet robotics system, the end-user (the supervisor) has the ability to replace an existing VR with a new one having specialized constructs to deal with changes in network conditions. This can be accomplished on the fly, i.e., while the system is running.

6 Self-awareness and self-adaptability

Throughout this paper, we repeatedly stated that for distributed applications to be effective, they must be network aware and adaptable. In the previous section, we discussed how to develop a software application that is network aware and adaptable at design time, at deployment time, and at runtime with intervention from the developer (see the two examples at the end of Subsect. 5.2.2). In this section, we go one step further and sketch a common approach to achieve self-awareness and self-adaptability at runtime. We emphasize that the two approaches of Sects. 5 and 6 are complementary and are not exclusive of each other.

6.1 Application requirements and network performance

Different applications tend to have different communication requirements. For physical real-time distributed systems, the necessary requirement is to maintain system stability and the secondary objectives include speed of convergence and overshoot. In downloading web documents, performance is characterized by the latency to download a complete web page. In wireless applications, on the other hand, energy and power consumption are more significant factors. From such and other examples, we conclude that different applications have radically different performance requirements. Despite this fact, all application performance metrics are influenced by network-oriented performance metrics, which include delay, jitter, packet loss rate, and bandwidth availability. Consequently, application requirements can be supported by a unified middleware framework that ties network-oriented metrics and application-oriented metrics.

6.2 Middleware framework

The purpose of a middleware framework is to provide a common platform for a range of applications by acting as a broker between the applications and the network, as mentioned in Sect. 2. To provide such a service, the middleware usually incorporates specific modules for monitoring, analyzing, and customizing. One powerful technique to build applications that are able to observe and change their own code at runtime is through the use of reflective programming languages. This type of programming languages gives the program the ability to inspect and adapt itself (Mahmoud 2004; Schantz and Schmidt 2001). Based on this, we distinguish three types of modules: monitoring modules, decision-making modules, and customizing modules.

The monitoring modules collect online measurements for network-oriented metrics and report their statistics, see (NLANR 2004) for a list of network performance and measurement tools. Some applications require measurement of instantaneous and real-time events (e.g., commanding robotic systems are very sensitive to individual packet losses) whereas others require average quantities over a period of time (e.g., distributed simulations depend on the average RTT, see Sect. 3).

The decision-making modules retrieve statistics on network-oriented metrics from the monitoring modules, map these statistics to application-oriented metrics, and issue appropriate signals to the customizing modules. Mapping network-oriented metrics to application-oriented metrics can be achieved using several strategies and is application dependent. In some situations, a mathematical equation can be established to predict application-oriented metrics from network oriented-metrics. Another common strategy is to have an operational-threshold profile for the application-oriented metrics and the corresponding network-oriented metrics. The customizing modules alter other system modules and configurations in response to the control signals from the decision-making modules to adapt the system to network vagaries.

These three modules work in a feedback manner that resembles a conventional closed-loop feedback control system (Fig. 8). Specifically, the monitoring modules serve as sensors; the decision-making modules as controllers; and the customizing modules as actuators.

In our distributed simulation system (Sect. 3.3), we employed a similar feedback approach. The system dedicated a communication stream to measure the round-trip-time delay between the client and the server (sensing). The estimate of the delay is then used to extrapolate the game state according to the dead-reckoning strategy (computing control information). Finally, the extrapolated value is used to forward the game state by one RTT (actuating). Although we did not implement this as a distinct reusable framework—rather it was embedded within the application itself—it represents a groundwork for a network-delay compensation middleware.

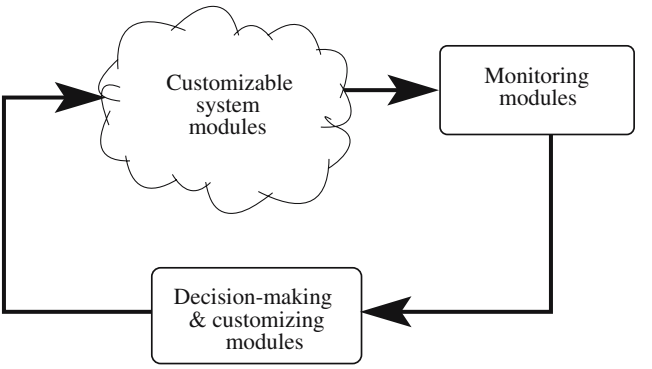

Fig. 8 Feedback control system approach

7 Conclusions

The paper has reviewed scalable healthcare alert systems and our scalable data dissemination component, our multi-agent Internet robotics system and our research in distributed simulation environments. These applications are vastly different but share one commonality: the software explicitly incorporate modules for communication features that are not supported in most of the current Internet. Although a software developer could reasonably expect that such communication functionality should be provided by the network infrastructure, in fact it is not so. The functionality needs to be explicitly implemented in the application. Further, we argued that such state of affairs derives from a fundamental design principle of the Internet. As a result, network awareness and application adaptability should be a critical component of networked applications. Conversely, network-blindness can lead to a "bubble" of over-inflated expectations.

By generalizing the experience from our examples, we argued that network environment investigation should be an additional step within the existing software development methodologies and pervasive applications should be configurable to enhance adaptability. Moreover, to achieve self-awareness and self-adaptability, system components should be able to monitor the network, and should be customizable to adapt to network vagaries. These principles are important to existing and future pervasive computing. One important yet challenging area for future work is ad-hoc systems. In an ad-hoc system, the heterogeneity of networks is more serious compared with traditional distributed systems. The network state can be more various and complex. Ad-hoc scenarios require much effort to investigate network environments in order to develop network aware and adaptable pervasive applications. Moreover, as these distributed systems have no fixed infrastructure, devices can travel in different networks that use different communication protocols. Thus, there is a need for an adaptive middleware that hides such different network protocols and provides to the applications a unified services interface.

Acknowledgements We acknowledge the help of two anonymous reviewers in greatly enhancing the clarity of this paper.

References

Al-Hammouri A, Covitch A, Rosas D, Kose M, Newman WS, Liberatore V (2003) Compliant control and software agents for internet robotics. In: Eighth IEEE international workshop on object-oriented real-time dependable systems (WORDS)

Arora R (1999) Voice over IP: protocols and standards. URL http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/voip_%protocols/index.html

Braden R, Zhang L, Berson S, Herzog S, Jamin S (ed) (1997) RFC 2205: resource reservation protocol (RSVP)—version 1 functional specification. Status: PROPOSED STANDARD. URL http://www.ietf.org/rfc/rfc2205.txt

Branicky MS, Liberatore V, Phillips S (2003) Networked control system co-simulation for co-design. 2003 American Control Conference

Buckeridge DL, Musen MA, Switzer P, Crubezy M (2003) An analytic framework for space-time aberrancy detection in public health surveillance data. In: AMIA 2003 fall symposium

Chrysanthis PK, Liberatore V, Pruhs K (2003) Middleware support for multicast-based data dissemination: a working reality. WORDS 2003, pp 265–272

Deering S, Hinden R (1998) RFC 2460: Internet Protocol, Version 6 (IPv6) specification. Obsoletes RFC1883. Status: DRAFT STANDARD. URL http://www.faqs.org/rfcs/rfc2460.html

Emulab (2004) Utah emulation facility. URL http://www.emulab.net/

Jini (2004) Jini network technology. URL http://wwws.sun.com/software/jini/

Li W, Penkrot V, Roychowdhury S, Zhang W, Chrysanthis P, Liberatore V, Pruhs K (2003) An optimized multicast-based data dissemination middleware: a demonstration. In: Proceedings of the 19th international conference on data engineering (ICDE 2003)

Luby M, Mitzenmacher M, Shokrollahi A, Spielman D, Stemann V, (1997) Practical loss-resilient codes. In: Proceedings of the 29th ACM symposium on theory of computing (STOC'97), pp 150–159

Mahmoud QH (ed) (2004) Middleware for communications. Wiley

MSMDD (2004) Middleware support for multicast-based data dissemination. URL http://dora.eeap.cwru.edu/mware

Ngai ML, Liberatore V, Newman WS (2002) An experiment in remote robotics. In: IEEE international conference on robotics and automation (ICRA), pp 2190–2195

NLANR (2004) Nlanr distributed applications support team. URL http://dast.nlanr.net/

NS-2 (2004) The network simulator—ns-2. URL http://www.isi.edu/nsnam/ns/

Pfleeger SL (1998) Software engineering: theory and practice. Prentice-Hall, Englewood Cliffs

Podgurski A, Leon D, Francis P, Masri W, Minch M, Sun J, Wang B (2003) Automated support for classifying software failure reports. ICSE

Saltzer J, Reed D, Clark D (1984) End-to-end arguments in system design. ACM Trans Computer Syst 2(4):195–206

Schach SR (1996) Classical and object-oriented software engineering, 3rd edn. IRWIN

Schantz R, Schmidt D (2001) Middleware for distributed systems: evolving the common structure for network-centric applications. Encyclopedia of Software Engineering

Schulzrinne H, Casner S, Frederick R, Jacobson V (1996) RFC 1889: RTP: a transport protocol for real-time applications. Status: PROPOSED STANDARD. URL http://www.faqs.org/rfcs/rfc1889.html

Smed J, Kaukoranta T, Hakonen H (2002a) Aspects of networking in multiplayer computer games. The Electronic Library 20(2):87–97

Smed J, Kaukoranta T, Hakonen H (2002b) A review on networking and multiplayer computer games. Turku Center for Computer Science (2)

Tsui FC, Espino JU, Dato VM, Gesteland PH, Hutman J, Wagner MM (2003) Technical description of RODS: a real-time public health surveillance system. J Am Med Informa 10(5):399–408

Varghese J (2003) Testing and profiling middleware-supported multicast software. Master's thesis, Case Western Reserve Universiy, Cleveland, Ohio
Wagner MM, Robinson JM, Tsui FC, Espino JU, Hogan WR (2003) Design of a national retail data monitor for public health surveillance. J Am Med Informat Assoc 10(5):409–418
Xu Z, Leon D, Podgurski A, Liberatore V (2004) Detecting aaa vulnerabilities by mining execution profiles. IEEE Symposium on Security and Privacy
Zhang W, Li W, Liberatore V (2004) Application-perceived multicast push performance. In: IPDPS 2004