# Improving Wireless Access to the Internet By Extending the Client/Server Model

J. Pasquale, E. Hung, T. Newhouse, J. Steinberg, N. Ramabhadran

University of California, San Diego - Department of Computer Science and Engineering
La Jolla, CA 92093-0114 - USA

e-mail: {pasquale,eyhung,newhouse,jsteinbe,nramabha}@cs.ucsd.edu

**ABSTRACT**

*We propose a new approach to structuring Internet applications for wireless clients that is based on extending the client/server model. The idea is to extend the client or server by introducing an intermediate vantage point of control between them. This enables special protocol support to effectively deal with the wireless link, and supports remote processing of server data so that it may be tailored specifically for that client. This control point is embodied by middleware that is layered above the network, and therefore does not require changes to the Internet's basic protocols nor to those of its underlying physical networks. We develop the model, and then describe ongoing research that focuses on various practical aspects and shows how we are realizing the model in useful working systems.*

## 1. INTRODUCTION

The current trend in how users access the Internet is via simpler and smaller wireless client devices. In fact, the emerging view of the Internet infrastructure is that of a core of mostly wired links that connect servers, covered by a variety of wireless clouds that provide connectivity at the fringes to client devices. These new-generation wireless devices provide convenience and freedom to users as a result of their being small, light-weight, and not tethered, while providing connectivity and access to the power available from the many resources attached to the Internet. The Internet protocol structure has been remarkably flexible to support this view, despite that it was not designed with this scenario as a common situation.

In fact, this is not without problems, which stem from deeply rooted assumptions based on the Internet's "end-to-end" design philosophy. The Internet is now being pushed towards supporting far greater heterogeneity, including endpoint devices that have significantly less computing power, and wireless network links that have significantly lower performance, reliability, and security characteristics. Let us consider these issues in more detail.

User access devices such as PDAs (personal digital assistants) have significantly smaller displays, slower processors, and smaller memories, than more traditional end-user computers such as workstations and PCs. Given the trend towards wider variety and specialization (enhanced digital phones, digital audio players, digital video glasses, etc.), the differences in power and expected capabilities will continue to grow larger. Importantly, what the Internet can expect the end-user device to do (i.e., whether and how it performs expected end-to-end protocol functions) is becoming more constrained. This issue also affects content-providers and application programmers, as their model of the machine that allows the end user to access content and run applications is no longer simple.

That these devices already or will eventually (and inevitably) connect to the Internet via wireless links introduces further complexity. Wireless links have significantly different properties regarding performance, reliability, and security, than the more prevalent wired links with which the Internet evolved. Wireless links generally have much lower bandwidths and much higher error rates. Outages that result in disconnections are common, and the ability to eavesdrop is qualitatively easier when compared to wiretapping a wired link. And, wireless implies mobility, which, in addition to the unique complexities it introduces, exacerbates these other problems by increasing variability.

We believe that these differences are so significant that the standard mechanisms of typical Internet protocol implementations cannot simply be enhanced or tweaked to adequately address the resulting problems. When coupled with the limited capabilities of new end-user devices, the implications require a revisiting of basic assumptions that have driven Internet protocol and application design, including and especially regarding how they relate to the Web.

Basically, the Internet assumes powerful endpoints (in contrast to common telephone networks). The Internet expects endpoints to carry out significant control functions, and in fact, the end-to-end design principle [1] encourages that as much as possible be done on an end-to-end basis, as this simplifies the internals of the Internet (e.g., routers), leading to increased efficiency and lowered costs. This philosophy, which has served the Internet well, implies that a user "pays" (in one way or another) only for what they need; as long as the Internet delivers packets, it is up to the endpoint to enhance this most basic service with such properties as in-order delivery, reliable transmission, and flow control, to name a few.

The endpoint is also expected to realize abstractions such as streams ("continuous" flows of data without message boundaries). Even some functions that are considered more internal to the network, such as congestion control, are to be carried out by endpoints. In fact, all of the mentioned functions are carried out by TCP (Transmission Control Protocol), a protocol that only runs at the endpoints (the lower-level Internet Protocol, IP, which also runs on routers, is mainly charged with packet delivery, routing from one place to another). But, what happens if the endpoint is incapable of carrying out these functions (either at all, or above some required level of quality)?

We propose a new approach to structuring Internet applications that is based on an extension of the client/server model. The idea is to introduce an intermediate control point between client and server (or more generally, sender and receiver), placed "beyond the wireless link" relative to the client, i.e., such that the wireless link is between it and the client. This provides a vantage point of control that will enable special protocol support to deal with the wireless link, and to support processing of server data to tailor it for that client. Finally, this control point is embodied by middleware that is layered above the network, so that we do not seek changes to the Internet's basic protocols nor to those of its underlying physical networks.

## 2. MODEL DEVELOPMENT

We first consider how to introduce the intermediate control point. We begin with the simplest model of communication, that of a single sender S and receiver R communicating over a network. Imagine taking the sender S, and splitting it into two parts, a base $S_b$ and an extension $S_e$, as shown in Figure 1. The base and extension will typically be distributed on different nodes of the network. Where in the former case S simply sends a message to R, now $S_b$ sends the message, it is received by $S_e$, which then forwards it to R. This is not limited to the sender side. The receiver R can similarly be split into a base $R_b$ and extension $R_e$, distributed on different nodes, and when a sender S sends a message, it first is received by $R_e$, and then passed to $R_b$.

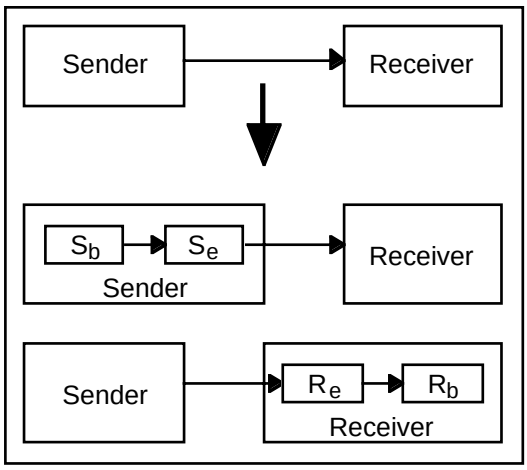


Figure 1. Extending simple communication.

Most importantly, the decomposition of base and extension, and its distribution over nodes in the network, is transparent to the "other side," i.e., the receiver (sender) if it is the sender (receiver) that is being decomposed. In the simplest case, the extension simply acts as a relay for the base without the other side being aware. In more interesting cases, the extension offers the opportunity to modify the message, and also the ability to carry out a specialized protocol between base and extension that is different from that between the extension and the other side.

To achieve this goal of transparency, it is helpful that this proposed model be viewed as a hierarchical decomposition of the simple one: S->R becomes $[S_b\text{->}S_e]\text{->}R$, or $S\text{->}[R_e\text{->}R_b]$, and the one side views the bracketed portions of the other side as monolithic. This is important for the following reasons:

- The original sender/receiver protocol stays the same. This protocol is based on already established and agreed-upon conventions and standards, and generally cannot be changed or determined dynamically. Rather, such protocols are generally assumed prior to communication, not just for the actual communication of data, but also to establish communication between a sender and receiver that are independent.
- The “bracketed” protocol can be different from that of the original sender/receiver. This protocol is completely determined by the side being decomposed and is unaffected by (and does not rely on) the other side. It can be specialized to meet the needs of unique situations (e.g., limited capabilities of sender or receiver, problematic properties of the network between base and extension) and even determined dynamically (or parameterized) to deal with current conditions (e.g., system load and network traffic).

These properties provide a nice balance between simplicity and flexibility. Conceptually, we still have a single sender communicating with a single receiver, with no change in the protocol between them. Yet, we now have the ability to introduce an intermediate control point between sender and receiver, and to establish a specialized protocol between one end and the intermediate point.

Given this simple structure, what is gained? We will illustrate this via a set of examples. In the first example, consider a sender connected to a receiver via a path composed of a wireless link followed by wired links. To be able to deal specifically with the problems of the wireless link, the sender is decomposed into a base and extension and these components are distributed so that they straddle the wireless link, as shown in Figure 2. Communication between the base and extension can now be accomplished via a specialized protocol that takes the wireless link into account. This protocol (or set of layered protocols) may:

- Compress/decompress messages to make better use of the wireless link's limited bandwidth
- Retransmit messages with timeout periods tuned specifically for the wireless link
- Encrypt/decrypt messages to deal with the ease of eavesdropping over the wireless link.

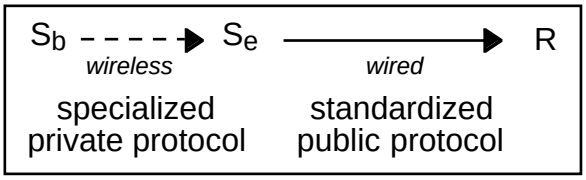


Figure 2. The sender’s base and extension can communicate using a specialized protocol optimized for the wireless link.

Similarly, if it is the receiver that is connected via a wireless link, it can be decomposed into a base and

extension. In addition to carrying out a specialized protocol that performs the functions already stated above, the extension may:

- Tailor the message to specific requirements of the receiver, as in filtering unwanted data and reducing images (their size, resolution, color depth, etc.) that are to be displayed on the receiver's limited display
- Maintain a copy of critical messages in case the receiver loses power in the future, allowing it to retrieve them at a later time
- Maintain a copy of the last n messages in case the receiver is lost or destroyed, allowing a recovery process to retrieve them and make determinations about the receiver's state (similar to the function of a "flight recorder").

In all these examples, we are essentially customizing the communication to deal with the characteristic peculiarities of the endpoint and the wireless link, at a range of levels, from hardware to system software to application-level software. This brings up the all-important question of what is the proper level for the extension to operate.

We are assuming that the extension can be placed at a location between sender and receiver to achieve the potential gains. But is it reasonable to expect that, say, a router, accept foreign code on behalf of a sender or receiver and arbitrarily execute it? This is essentially the approach taken by the proponents of Active Networks [2]. This would significantly increase the complexity of the network, and goes against the current Internet structure whereby most of the complexity is located at the endpoints, with routers carrying out the limited "network layer" functions as embodied by IP.

A compromise to this solution is that only certain routers be endowed with this capability. These might be strategically located ones, such as those on or near base-stations[1], as a primary use of the extension is to deal with the problems presented by wireless links. However, this does not remove the complexity, it only limits its locations. All users of those routers must pay the costs of that complexity, whether they use the extension-execution capability or not. Note that these costs may be indirect ones, such as those due to failures in reliability or reductions in performance that arise from the new bugs that will inevitably be introduced by complicating that node's network layer software, or because of the difficulty in upgrading and improving the existing implementation that is now significantly more complex.

At the opposite extreme of placing this capability "inside the network" is to make extensions completely part of the application. The advantage is that the network is unaffected, and all costs fall on the user of that application and no one else. But to allow the extension to be physically located away from the base, the endpoint to be extended must have the ability to execute user-level code not only on its original machine (which will now run the base), but another machine whose location provides the benefits we are seeking.

[1] The use of the term “base” in this paper, which refers to a software component, should not be confused with “base-station,” the familiar term used in wireless systems.

Actually, in practical settings, this is not an unreasonable supposition. For example, a user working at home might have a PC that is both connected to the Internet with a high-bandwidth link (cable TV) and also serves as a base-station for the home wireless LAN. Now the user can access the Internet with their wireless PDA anywhere in the home, with extensions running on their PC (and the base code running on the PDA). Similarly, at work, the user's office PC might act in this role.

The approach we propose is a middleware solution, that goes between the network layer and application layer. This middleware would run above the network layer, and indeed, in a user-level process (as opposed to part of the operating system). Any machine that is designated as able to run extensions would support this middleware. Prime candidates would be, as mentioned above, a user's home or office PC. More generally, they might be third-party machines that rent processing time to users who purchase such time on demand, or where they have pre-existing contracts (as with a user's wireless network provider, where the machine may be the base-station itself, or more likely, a machine directly connected to it for this purpose). In these latter scenarios where the extension code is not trusted because it originates from a foreign machine under a different administrative authority, care must be taken that it can be determined to be safe to run, or that its potential damage is limited.

A typical implementation of this middleware is that it be comprised of a server process that accepts requests to execute either pre-installed or dynamically loaded extensions, and that supports the dynamic binding of the extension to the base and other endpoint. The side that runs the base would also run a corresponding portion of the middleware to carry out such functions as finding and contacting the intermediate machine, identifying or supplying an extension to execute, and taking part in the dynamic binding of the base with the extension.

This distributed middleware may support a set of common functions that are useful to many applications. Many functions that might be embodied in extensions, such as various types of image filters, compression and decompression modules, encryption and decryption modules, and caches, are prime examples. More generally, it would support the ability to dynamically deploy corresponding portions of a specialized protocol that operate at base and extension.

Finally, there must be provisions made for mobility: Given a base and extension that bracket a wireless link, if and when the base moves, it may be necessary to move the extension (e.g., if such a move causes the base to move to a new cell in a cellular network).

Throughout this discussion, there is an implicit assumption that all of the benefits derived from this extended model of communication outweigh the complexities it introduces. While we have avoided adding complexities to the network, we have certainly made the application layer more complicated. Indeed, this complexity is now visible to the programmer, which stands to reason since we wish to empower the programmer to make the application dynamically adapt to deal with the problems of limited-capability clients and wireless links as presented above.

How do we make this complexity palatable? Since most Internet applications are structured according to the

client/server model, our approach is to extend this model according to the same main principle we used to describe our extended model of communication: hierarchically decompose one side, and make this transparent to the other. In addition, we use the concept of mobile code for dynamic deployment of extensions and to deal with mobility (of the base).

## 3. THE EXTENDED CLIENT/SERVER MODEL

The client/server model is well established and is the most common way of structuring distributed applications on the Internet, including email, ftp, and Web browsing. For example, Web browsers are clients that use HTTP (HyperText Transfer Protocol) to obtain content from Web servers in a simple request/response fashion.

We extend the client/server model by allowing either the client or server to be decomposed into a base and extension (see Figure 3), just as we extended the simple model of sender-to-receiver communication in the previous section. In most cases, it will be the client that is extended, as it is typically the client that will access the Internet via a wireless link. And because of the wide variety of user devices that will act as clients, the ability to tailor responses from existing servers, without forcing these servers to be aware of this variety and therefore have multiple versions of the same content, is critical to the efficient evolution of the Web and its integration of these new devices.

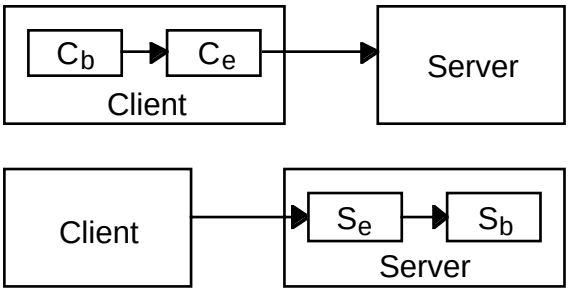


Figure 3. The Extended Client/Server Model.

In this model, prior to contacting a server, the client would deploy an extension to a node that supports the execution of extensions. The most flexible approach is for the extension to be in the form of mobile code, which originates either with the client, or from a remote repository. In the latter case, the client would supply a reference (rather than the code itself) that would indicate where to retrieve the extension.

The middleware that supports this model is also structured according to the client/server model. An extension runs on an extension server, which provides its execution as a service. Obtaining this service might require authentication, transfer of funds, a check on whether the extension is safe to run, and other controls of admission. Once accepted, the extension server would provide processing, memory, and communication resources (e.g., buffers, bandwidth) to effectively run the extension.

Once the client's base and extension are in place, whenever the client is to make a request to a server, this request may originate at the client base or its extension. If from the base, it first gets passed to the extension, and this transfer can occur using a specialized protocol, e.g., designed to specifically address wireless link issues. The extension also has the opportunity to modify the request, or keep a copy, before passing it to the server. As far as the server is concerned, it receives the request from the client, unaware that it came from the extension or that it originated at the base.

As implied above, the request may also "originate" from the extension. This might occur if the request that was saved as a copy at the extension is to be repeated; or, it might occur if the extension did not immediately forward the request after it was received, as in a real-time situation where the extension is to forward it at a specific time, avoiding the potential variability that might be introduced by a wireless link had the request been sent directly from the base.

While we have focused on the common case of extending the client, extending the server is useful also. The obvious scenario is where the server faces problems analogous to those we considered for clients: limited processing capabilities and connectivity via a wireless link. A good example is a small wireless monitoring device that senses its environment and acts as a simple server by accepting requests for the current state.

But there are other situations that motivate extending the server that have nothing to do with processing limitations or wireless connectivity. For example, a server extension can reduce the distance, and therefore latency, to a client by providing a presence and limited selection of services near the client. Such a service might be a cache: if the cache contains the content the client requests, it provides it without accessing the server base.

While this example shows how to improve performance as well as reliability (if the base is temporarily down, the extension can still provide limited services), extending the server can also be applied to improve security. One of the major problems facing e-commerce Web servers today is denial of service attacks, where the server is bombarded with requests and effectively shutting it down. Now consider a server that launched numerous extensions, thus allowing the server to now have multiple presences. A client would send its requests to one of them (which it sees as the actual server) that would then simply be passed along to the server base. A denial of service attack upon one of the extensions would be isolated to the machine serving it (which would need to be able to recognize such an attack and then not pass along the requests).

## 4. CURRENT RESEARCH

We now present overviews of current research we are undertaking. These projects focus on various practical aspects of the issues presented "in theory" so far, and show how we are realizing them in useful working systems. We first describe a Java-based implementation of middleware that supports extensions. Next, we describe a system that specifically deals with extending Web applications to customize content for non-standard user devices. We then describe a system that supports protocol agents and their pattern-based development. Finally, we describe an extension to NFS (Network File System) that supports intermediate caching.

### 4.1 JAVA ACTIVE EXTENSIONS

The Java Active Extensions middleware system [3] addresses the practical issues involved with distributing an endpoint's functionality by providing mechanisms to load, execute, and interact with code at a remote location. We use Java as the implementation language for these mechanisms because of Java's widespread availability, platform independence, and support for mobile code (e.g. dynamic code loading from remote sources, fine-grained security policies, and object serialization). The results of our work include an API that specifies the syntax and semantics of components in the system, a middleware implementation that fulfills the system components of the API, and test results that indicate the overhead introduced by an implementation of our model is acceptable compared to common network operations.

A Java Active Extension (JAE) is a Java object containing code intended for execution at a remote location. An application dynamically loads a JAE to a remote host that is willing to execute the JAE on the application's behalf. The JAE enables the application to use the resources of the remote machine in a manner specific to its present needs (and future needs, if the JAE adapts to changing conditions). The ability to execute code remotely enables an endpoint to deploy an extension that implements customized communication with the endpoint's base.

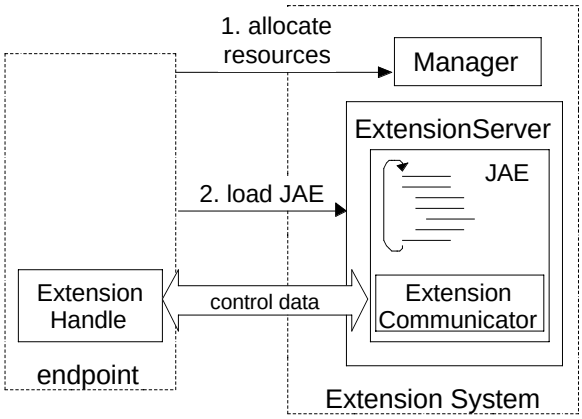


Figure 4. The Java Active Extension system architecture.

The machines that execute JAEs are part of an Extension System. An Extension System manages the resources of a set of machines willing to host JAEs. A client of an Extension System must first reserve a set of resources before any of its JAEs can be loaded to the system. An Extension System involves five core components (shown in Figure 4):

- JAE – A Java object written by the application developer to perform the necessary remote tasks. At runtime, the endpoint loads the JAE to a suitable remote machine for execution.
- Manager – Resource broker for an Extension System. This component is responsible for servicing resource reservation requests from clients in accordance to an Extension System's policy.
- ExtensionServer – An execution environment for JAEs. The ExtensionServer represents a set of resources allocated to a client of an Extension System by a Manager. The client may load multiple JAEs to the ExtensionServer for simultaneous execution.
- ExtensionHandle – An application's link to a loaded JAE. The application may use this object to communicate control data to the running JAE.
- ExtensionCommunicator – Fulfills the JAE's side of the communication channel by enabling control data communication with the corresponding ExtensionHandle.

Application developers are only responsible for implementing the JAE they wish to execute remotely. Extension System middleware provides implementations of the remaining components.

In addition to executing JAEs, an Extension System provides a rudimentary data channel to address the need to bootstrap the base and extension with information required for more sophisticated communication. The communication mechanism is a queue of messages accessed by the ExtensionHandle in the endpoint's base and the ExtensionCommunicator in the endpoint's extension. Messages are any Java objects agreed upon by the base and extension.

### 4.2 WEB CUSTOMIZERS

The Web Stream Customizer (WSC) Architecture [4] tailors the extension paradigm specifically to Web browsing. Customizers are distributed web customization software modules that are dynamically deployed and used by clients during a Web session (although servers and even third parties can deploy and use them). Customizers are seamlessly integrated with the basic Web transaction model, simplifying their programming and operation. This is because the WSC system exploits the Web's proxy capabilities, and makes use of standard code mobility mechanisms (with Java as the language of choice given its portability). Thus, importantly, Customizers will work with standard browsers and Web servers, without requiring any modifications to them.

A key feature of the WSC architecture is that it supports cooperative customization at two points along the path between client and server. Many types of customizations require such cooperation and distribution of functionality. For example, data compression (e.g., to reduce bandwidth requirements, and perhaps latency) requires that compressing be done before the data crosses any relatively low-bandwidth links, and that decompressing be done afterwards. To accomplish this, a Customizer is comprised of two components: a *local component* (LC) and a *remote component* (RC). The LC runs on a *Local Customizer Server* (LC-Server), and the RC runs on a *Remote Customizer Server* (RC-Server).

In order to use Customizers, the Web browser is configured to use the LC-Server as its proxy, so that its requests are automatically forwarded to the LC-Server. Thus, when a Customizer is being used, the request passes from the client to the LC, then to the RC, and then to the server (and vice-versa for responses in the opposite direction, from server to RC to LC to client).

The LC and RC have the opportunity to modify the request and the response when they are received, before they are forwarded along the communication path.

There will generally be many Customizers, each one being a separate (LC, RC) pair, simultaneously active on behalf of a single client. All of the LCs (for that client) will run on a single LC-Server, and the RCs will be generally running on different RC-Servers, as shown in Figure 5. When multiple Customizers are active, the LC-Server must have a means of determining which Customizers should receive which browser requests, and hence, which RC-Servers will be involved in which requests. Consequently, each Customizer has associated with it a *Domain of Applicability* (DA), which specifies a set of URLs or host domains (i.e., Web servers), and this is stored at the LC-Server.

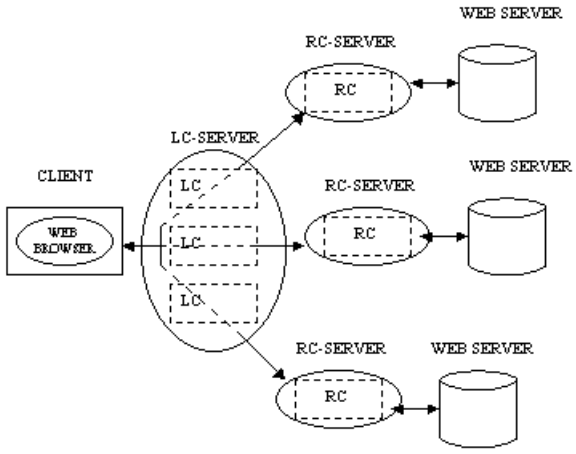


Fig. 5. The WSC architecture with multiple active customizers.

The LC acts primarily as an extension of the browser (given that the browser code itself cannot be modified). The LC runs on an LC-Server, which tends to be located on or near the client Web device. Given its close coupling with the client, the LC is generally responsible for tasks that require knowledge of resource availability and system conditions at or near the client, which may then be communicated to the RC (e.g., to improve performance, such as relaying local system or network performance status). In addition, the LC will also reverse data transformations done by the RC, such as compression/decompression or encryption/decryption.

The RC generally performs location-dependent tasks that benefit from being near the server (or simply away from the client), such as compressing response data from a server before it is transmitted over a low-bandwidth link on the communication path to the client. The RC runs on an RC-Server, which tends to be located near, or even on, a Web server of particular interest.

### 4.3 PATTERN-BASED PROTOCOL AGENTS

A protocol agent is mobile code generated or launched by a sender or receiver (designated as the *originator*) that will run at the intermediate control point and implement a specialized protocol between the originator and intermediary. The protocol agent defines a static, modular protocol data path to facilitate the construction of the specialized protocol while providing security to the intermediate point.

Protocol agents break down the components of a protocol into modular sections of code that may be reused in other protocols. For purposes of construction, modules are categorized into types:

- Processor modules, which perform operations upon the input data for processing by another module.
- Communicator modules, which are linked to an endpoint and contain code that is able to parse data from its endpoint into a suitable format for processing, or to pass on processing results to its endpoint.

The simplest form of protocol agent is structured as follows: a communicator module that communicates with the originator; a series, or stack, of processor modules that operate upon the data in a fixed order; a communicator module that communicates with the target; and corresponding module parameters. As shown in Figure 6, when application data arrives at the intermediary, it is transformed by the appropriate communicator module into a data type suitable for processing and then sent to the processor module of the series closest to that endpoint. Each processor module, after processing the data, outputs the result in the same data type to the next module in line, before arriving at the communicator module for the receiver and being sent along its way.

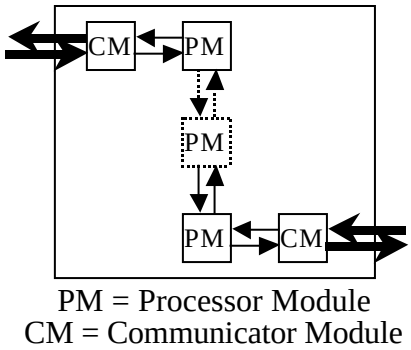


Figure 6. Structure of a simple protocol agent.

We have identified and implemented certain types of protocol agents that form useful (and reusable) patterns:

- FILTERS that process data and transform it according to a specific algorithm. Examples: an image filter that shrinks the image to fit onto a PDA's screen, or a compressor that compresses data before sending over a link with limited bandwidth.
- MONITORS that regularly monitor a remote object for changes in state. Examples: monitoring a remote system's availability, or an important Web page with sporadic updates.
- CACHES that store data for later retrieval. Examples: caching transaction results if the link between the agent and the receiver goes down, or caching to eliminate the network traffic on a static, previously referenced object.

- REGULATORS that perform flow control on the data received in order to limit bandwidth to a certain amount. Examples: a jitter control algorithm for a streaming video application, or to limit the outgoing bandwidth on a limited connection.

Most protocols use a combination of these functions, so a rich library of processor modules implementing these patterns will provide a flexible and general basis for constructing applications with specialized protocols. Our pattern-based implementation of protocol agents also improves the security of the intermediate control point through the ability to describe its behavior as a series of patterns with client-specific limitations. During the generation of the agent, the originator can specify limitations to the modules as a parameter (i.e. peak output bandwidth in a Regulator-type module, cache size in a Cache-type module, etc.), and this information is stored in a manifest that is presented to the intermediate point upon arrival. The intermediate point therefore knows the expected behavior of the agent. For modules from standard libraries downloaded from a public server, an intermediate point can even make decisions based on the past trustworthiness of the module source, introducing a measure of accountability.

### 4.4 FILE SERVICES FOR WIRELESS DEVICES

Network file systems are naturally structured into a client process and a server process. However the limitations of a wireless environment make this model inadequate for a number of reasons. For example, many network file systems make extensive use of client side caching to mask the latency between the client and the server. However, PDAs lack the memory and the storage capacity to support large-scale caching and prefetching. In addition, it is undesirable to cache file updates at the client due to limited battery power and the unreliability of the wireless link. Lack of caching and synchronous file updates to the server would impose a severe performance penalty on applications using remote files, especially in the wide-area. Most cache consistency protocols are insensitive to network quality and battery power limitations at the client.

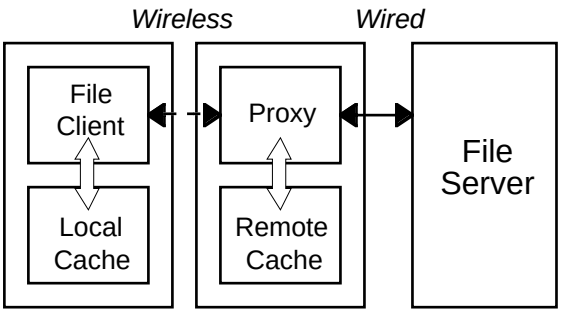


Figure 7. Network file system based on Extended Client/Server.

We are applying the extended client-server model to network file systems to mitigate these problems. We use an intermediate proxy, which is a processing point typically located at the boundary between the wired and wireless networks as shown in Figure 7. All file system requests by the mobile file client on the remote file data are routed through the proxy server, which interacts with the remote file server to satisfy these requests. The proxy performs caching and state management functions on behalf of the client. The client may maintain a local cache whose size depends on the available memory. The proxy maintains a large file cache to improve the performance of the file system. The proxy may also implement a prefetching policy in which frequently accessed data is prefetched from the server. The interaction between the proxy and the server is identical to the interaction between the client and the server in the traditional model.

The advantages of this approach are that the client is relatively stateless, therefore limiting required resources at the client. Secondly, it permits the deployment of custom protocols between the client and the proxy. Cache consistency protocols are made adaptive to the resources available at the client and the quality of the wireless link. File updates are synchronously written to the proxy to prevent loss of data due to failure. Due to the proximity of the proxy, this does not impose too much of a penalty in terms of overhead. This can be further reduced by differentiating between critical updates, which are synchronously written to the proxy, and non-critical updates, which are cached at the client if possible. The local cache at the client can be used to deal with disconnected operation to the extent permitted by the availability of memory on the client.

## 5. CONCLUSIONS

We have presented an extended client/server model of distributed computing, and have shown how it can be used to improve wireless access to the Internet. The primary benefit of the model is that it allows a client (or server) to be decomposed into base and extension code portions, where the extension can be located at a remote vantage point. This enables the dynamic deployment of specialized protocols to deal with the unique problems presented by limited-capacity endpoint devices and wireless links. To gain experience, we are pursuing four research projects that we have described, and that explore different design and implementation issues of middleware architectures that realize this model.

## ACKNOWLEDGEMENTS

This work is supported by research grants from AFOSR, DARPA, and an NSF Research Infrastructure grant as part of the UCSD Active Web Project.

## REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-end arguments in system design,” ACM Trans. Computer Systems 2 (4): 277-288, Nov. 1984.

[2] D. Tennenhouse and D. Wetherall, “Towards an active network architecture,” Computer Communications Review, 26(2): 5-18, Apr. 1996.

[3] T. Newhouse, “Java Active Extensions: A mobile-code mechanism for extending client resources,” M.S. Dissertation, UC San Diego CSE, Dec. 2001.

[4] J. Steinberg and J. Pasquale, “A Web middleware architecture for dynamic customization of Web content for non-traditional clients,” UC San Diego Technical Report CS2001-0692, Dec 2001.