

# An Extensible and Scalable Content Adaptation Pipeline Architecture to Support Heterogeneous Clients

Thomas Phan, George Zorpas, and Rajive Bagrodia  
University of California, Los Angeles  
Parallel Computing Laboratory  
3809 Boelter Hall, Los Angeles, CA 90095-1596  
{phantom, zorpasg, rajive}@cs.ucla.edu

## Abstract

*The importance of middleware and content adaptation has previously been demonstrated for pervasive use of Web-based applications. In this paper we propose a modular, extensible, and scalable middleware component called the Content Adaptation Pipeline that performs content adaptation on arbitrarily complex data types not limited to text and graphic images. Furthermore, the architecture can be used as part of many client-server applications, not just Web browsers. In our work we leverage the XML language as a uniform means to describe all the elements in our architecture, including the client device and user profiles, the data characteristics, the transcoding operations performed on the data, and the resultant adapted data. We illustrate the flexibility of our architecture to support new data types and adaptation operations by first showing its use with data from a real-world medical application and then extending its capabilities to handle animated graphics and also real-time streaming RTP data. Finally, we demonstrate scalability in our architecture by executing the Content Adaptation Pipeline over a distributed set of servers running an efficient protocol.*

## 1. Introduction

The proliferation of mobile network-ready devices has escalated Internet-connected client heterogeneity to an all-time high. Client devices today can vary drastically from a high-performance wired workstation to a mobile personal digital assistant connected with a low-bandwidth cellular data modem. When one takes into account the possible variations in device characteristics, the resultant combinations clearly make the approach of uniform data distribution among these clients ineffective. In this paper we build upon a foundation of previous efforts that have investigated the necessity for adapting data to fit the needs of varying clients.

In our research we have developed an extensible and scalable architecture called the Content Adaptation Pipeline (CAP) to perform content adaptation, also known as transcoding or distillation. Extensible content adaptation has emerged as a vital topic in ubiquitous or pervasive computing due to the necessity of handling ever-changing software and hardware requirements of heterogeneous clients. To make these ideas concrete and to demonstrate the relevance of our work, we have implemented our CAP architecture in the context of the Interactive Mobile Application Support for Heterogeneous Clients<sup>1</sup> (iMASH) research project [4] [32], whose

<sup>1</sup>This work is funded by National Science Foundation award ANI-9986679, "iMASH: Adaptive Middleware and Networking Support for the Nomadic Healer."

goal is to create a software and hardware infrastructure to support wireless computing for use by mobile physicians. In the iMASH design, an Application Server (AS) acts as a database repository for users working from a number of clients. These client machines are heterogeneous, varying in CPU performance, display capability, storage capacity, network bandwidth, installed runtime libraries, and other characteristics. The AS is typically a legacy system not specifically designed to provide service to clients with heterogeneous device characteristics.

To address this issue, the iMASH architecture utilises a Middleware Server (MWS) layer to provide content adaptation. To effectively serve heterogeneous clients, a data object's presentation can be changed. For example, it has been suggested in other research work (e.g. [6], [17]) that a graphical image be adapted, or transcoded, to fit the network and display limitations of a client machine by, for instance, reducing the colour depth, reducing the resolution, or cropping out portions. Such previous work has involved adaptation of fixed, well-known data types such as images and text because these predominantly comprise current Web content.

On the other hand, this paper addresses a broader problem: we describe an extensible and scalable content adaptation architecture that can handle a variety of data objects stored in a repository Application Server and requested by heterogeneous client devices. Web pages typically contain data objects of pre-defined, well-known types such as multimedia sound and video, text, and images. Our system provides the capability to support *arbitrarily complex data types*; by incorporating clearly-defined hooks where software packages can be attached, the architecture will be able to handle future types that are as yet not known. Our implementation can additionally be added into many types of client-server systems running any protocol, not just WWW interactions using HTTP. To that end, the proposed architecture decomposes the requisite activities needed for adaptation into separate functionalities for: collecting client profiles; identifying the characteristics of data objects; using heuristics to generate commands to adapt the objects; and producing the resultant adapted data.

To demonstrate our design, we have implemented a Middleware system that performs content adaptation for the benefit of a real-world medical client application called the Teaching File. Our Middleware executes the portions of our Content Adaptation Pipeline by leveraging existing Java libraries as well as the XML language as a uniform means to describe data within the various stages of our architecture. Additionally, with its modularity our architecture is expandable to support both new data types as well as new client devices. We illustrate the Pipeline's flexibility by first showing its original intended use with the proprietary Teaching File medical data and then extending its capabilities to transcode XML/HTML, animated graphics, and real-time stream-

ing RTP data. Finally, to insure that our middleware-based approach scales well with increasing load, we designed our Content Adaptation Pipeline to operate over a distributed set of servers running an efficient protocol.

This paper is organised thusly: In §2 we discuss iMASH background and other research projects related to ours. In §3 we detail the Content Adaptation Pipeline and its components and provide information on our implementation. The extensibility to handle new data types not initially anticipated is showcased in §4. To address the issue of scalability, in §5 we describe the use of multiple middleware servers aggregated together to improve performance under high load. To quantify our design we performed a variety of experiments to gather performance results, shown in §6. We conclude our work in §7.

## 2. iMASH Background and Related Work

The iMASH project is a multi-discipline collaborative effort between the UCLA Medical School and Computer Science Department. We envision doctors using iMASH-enabled technology to be able to access medical records on legacy databases from any location, any platform, and at any time within a hospital equipped with wireless connectivity. This ubiquitous computing will hopefully improve patient care through better information accessibility.

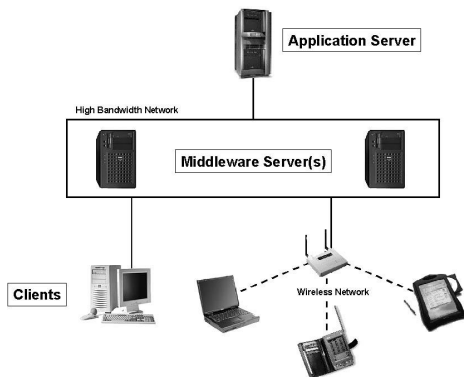


Figure 1. The iMASH architecture

To support this goal, iMASH provides a three-tier architecture, as depicted in Figure 1. An **Application Server** (AS) provides information for multiple heterogeneous **clients** connected via wired or wireless networks. A **Middleware Server** (MWS) tier is introduced to support secure, nomadic access to the AS by these clients; this layer is positioned such that all data between the AS and the clients is sent in-band through this tier. In general, this Middleware Server layer can be distributed across a number of separate machines, forming an aggregate *Middleware Service* that can improve scalability.

In previous work [29] [30] we have shown how this architecture supports nomadic access via the notion of Application Session Handoff. With this technique, a user working on a client device can transfer his application session to another client with minimal interruption in service. The session, typically a set of variables and data structures, is sent from one machine to another using the network as a conduit, thereby granting the user the freedom of uninterrupted access from any network-connected device, either mobile or fixed.

The focus of this paper is to show how the Middleware will also provide an extensible and scalable capability for content adaptation, which is essential for such an environment. Adaptation may be required for one or more of the following reasons: (1) the user's client device may lack the hardware or software to properly present a particular data type (for instance, the CPU may not be sufficient to handle a computationally-intensive 3D rendered object, a PDA may not be able to display colour or play sounds, or a desktop may lack the proper run-time codecs to decode a type of streaming multimedia); (2) the device's network bandwidth may be too limited to sufficiently support large data; or (3) the user's role limits his or her rights to access given data (for example, an assigned physician may be allowed to view a patient's data records, but other staff members may need to have the records filtered in order to enforce certain privacy legalities). Because content adaptation plays such a central role in our system, having an efficient and easily augmentable mechanism to perform adaptation is vital. This paper describes such a capability.

Adaptation for data delivery to heterogeneous clients has been actively explored by many research groups; in general, these efforts did not address the need for easy extensibility to support arbitrary data structures. Early work in this field was shown in [6] [14] [15] as part of the BARWAN project. In their research, the authors implemented an adaptation engine using a network of workstations to transcode data to fit the needs of only Palm personal digital assistants. The research in Odyssey [27] allowed applications to adapt themselves to network condition information. The Mobiware Toolkit [3] also facilitated delivery and manipulation of data in response to changing mobile network quality of service conditions. [17] presented a thorough analytical model for transcoding. Conductor [40] is a middleware framework that deploys adaptation agents into the network. The project in [18] used an annotation-based transcoding system where each HTML page or separate parts of each page must be associated with annotation files that give descriptions as to what kind of adaptations are needed for the Web page. Aurora [19] focused largely with transcoding web pages for disabled users. Additionally, in the business enterprise community, Enterprise Portal products are used to offer unified, customised access to server databases and web content (maintained within a company's intranet) to fit specific users or groups.

The preceding efforts implemented transcoding via middleware proxies, a technique shown in [7] to add functionality to the WWW's client-server architecture. An alternative is to have the server provide *content negotiation* by maintaining pre-transcoded data specific for discretised ranges of client. In [26] the adaptation architecture used a structure called the Infopyramid to represent WWW-related content at multiple modalities and resolutions. Although this makes object retrieval fast, it requires that all objects are content-analysed and adapted at the server for every possible situation that may arise before objects are requested. In [5] [23] [24] the authors utilised a proxy to augment client requests to servers running the content negotiation functionality of HTTP 1.1. Although server-side pre-transcoding can provide benefits, the application server is encumbered by additional calculation and storage to support transcoding, thereby violating our fundamental tenet that the application server should remain inviolate.

This paper does not develop novel capabilities or algorithms for content adaptation of multimedia, a well-researched area (e.g. [12] [13] [28] [1] [9]). Rather, we utilise such adaptation schemes by plugging them as needed into our extensible architecture, as we shall discuss later.

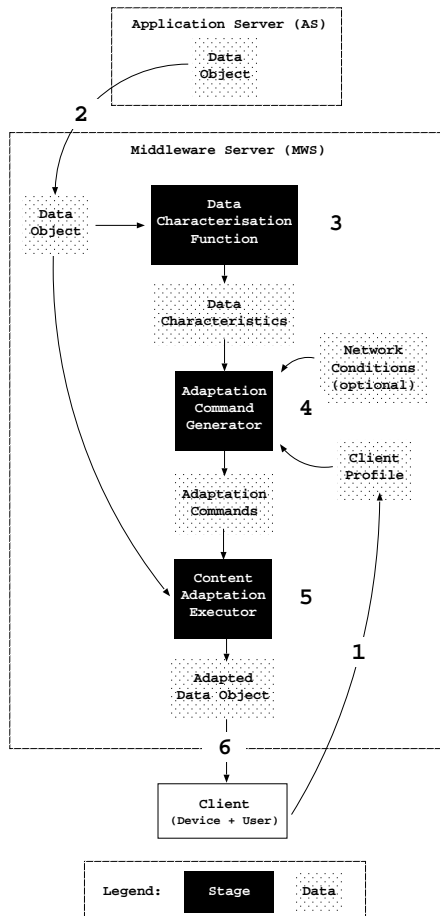


Figure 2. Operations within the Content Adaptation Pipeline.

### 3. Architecture of the Content Adaptation Pipeline

#### 3.1 Overview

Consider a situation wherein a user working on an iMASH-enabled client wishes to download a data object from the Application Server (AS). Figure 2 illustrates the sequence of steps carried out within the CAP to support this access.

A client application must register itself with a Middleware Server (MWS) in order to participate in the iMASH-enabled environment [29]. This registration is needed for user authentication and service discovery. As part of the client registration, the client's profile is sent to and stored at the MWS, as shown in step (1) of the Figure, to provide the MWS with necessary parameters to perform content adaptation. The client profile includes representative information for both the client device as well as the user. The client device profile contains information regarding the device's display, memory, installed runtime libraries, network interface connection, and CPU, among others. The user profile provides information such as the user's access privileges and role. Both types of infor-

mation can be dynamically changed but are initially provided at the time of the registration. The client profile will be explained further in subsection §3.2.

The client application proceeds to request a data object from the AS via the MWS. If the data object is not already cached at the MWS, the MWS requests it from the AS. After the object is sent from the AS, as shown in step (2), it is cached at the MWS. To identify the data object, we pass the object through (3) the Data Characterisation Function, or DCF, to determine its data type as well as its characteristics. The DCF is detailed in subsection §3.3.

The data object's characteristics and the client profile (and optionally the current network conditions) are passed as inputs into (4) the Adaptation Command Generator, or ACG, to produce a set of commands specifying how the object should be adapted to best match the client's constraints. These commands are then parsed by (5) the Content Adaptation Executor, which will call the appropriate routines within its pool of available adaptation library functions to perform the actual transcoding on the previously cached object. Finally, (6) the adapted objects are sent out of the pipeline to the client. The ACG and the adaptation functions are further discussed in subsections §3.4 and §3.5, respectively.

As we will show, we use XML extensively in our architecture because it provides a consistent mechanism to represent important information and because it is a de facto industry-wide standard in data representation. The Content Adaptation Pipeline in the MWS can access XML-encoded information by using Document Object Model [11] routines that can easily extract information from an XML document by using tag names.

As a running example to demonstrate the use of the Content Adaptation Pipeline through the remainder of this section, we will utilise an iMASH-enabled application called the Teaching File. This program has been developed for use by radiological professors and clinicians at the UCLA Medical School as a pedagogical tool for medical students in their program. The Teaching File is used to compose and display a graphical image and accompanying text annotations by aggregating all requisite information into a file called a teaching file. We emphasise here that the information within a teaching file is composed of a custom data type. The data structure contains a standard graphics image, image manipulation commands, and annotative text. In order to leverage this program so that it may be used by multiple heterogeneous clients within the iMASH environment, our Content Adaptation Pipeline must be able to handle its custom data type. Indeed, as we shall show, our mechanism is flexible enough to utilise this data structure and demonstrates its capability for using other complex data types.

#### 3.2 Client Profiles

The client profile is an XML document that contains a pre-defined set of tags that each client attempts to complete with its respective information before uploading the profile to the MWS. This set of tags specifies available information that the adaptation functions will need later in the pipeline. If the client is unable to determine the needed information, a default value is provided. Providing this client profile for the MWS does not incur much repetitive network transmission overhead because this profile can be cached at the MWS and the XML document is a simple text file that can be compressed prior to sending. Our client profiles serve a similar role as the W3C's Composite Capabilities/Preferences Profile [8]; however, our implementation provides a lightweight set of information and is not comprehensive, thereby avoiding the latter's excesses as noted in [24].

The client profile has two parts: the *client device* profile and the *user* profile. The client device profile can include attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<client name="COMPAQ iPAQ 3630" id="0007">
  <screen_v_res>300</screen_v_res>
  <screen_h_res>200</screen_h_res>
  <screen_colours>256</screen_colours>
  <RAM>16</RAM>
  <OS>WinCE 3.1</OS>
  <NIC_speed>2</NIC_speed>
</client>
```

**Figure 3. Device profile of the iPAQ PDA showing its identifying name, its vertical and horizontal screen resolutions, the maximum number of screen colours, RAM in megabytes, its operating system, and its network interconnection in megabits per second.**

such as the client's RAM capacity, operating system, and screen resolution. In Figure 3 we show an example profile for one of the devices in our testbed, a Compaq iPAQ PocketPC personal digital assistant (PDA). The other half of the client profile contains user information to identify the user's access privileges depending on the application running on the client. The user profile may be used to grant limited rights to selected users to modify data objects or to access information at different levels of security. For efficiency, the client profile may be cached at the MWS.

### 3.3 Data Characterisation Function

For every object requested by the client, the Data Characterisation Function (DCF) is responsible for identifying the object's data type and producing descriptive metadata. For example, for an image data type, the DCF can determine attributes such as width, height, and number of colours. The object's metadata information is then stored into a Data Characteristics XML document, which will be used in the next stage of the CAP.

For an object to be characterised, its data type must be known. The DCF can determine the type using one of three methods:

1. As we have mentioned, in our architecture the client requests service from the MWS, which acts as a proxy on behalf of the client. Communication between the client and the MWS is performed through an agreed-upon messaging API. When the client application requests an object from the AS, it is very often the case that the client knows exactly what data type to expect. As part of its service request to the MWS, the client can convey the data type information to the MWS service process, which in turn provides this information to the DCF. This method is the preferred means for the DCF to obtain the type.
2. When the AS returns the data object to the MWS, it may be possible for the AS to return the data type as well. For instance, web servers return the MIME type of each returned datum as part of the HTTP protocol.
3. The DCF can parse the data object to determine the type. Many (but not all) data formats have a header which can be parsed to determine its type (e.g. JPEG files).

After the DCF determines the object's type, attribute information is extracted. We have a predefined set of tags for each type of object that exists in our AS, which we appropriately call a template. Upon the retrieval of the required information, the appropriate template is filled out and stored in the XML document. In this

```
<?xml version="1.0" encoding="UTF-8"?>
<TeachingFile>
  <TeachingFileViewport>
    <Size Width="512.0" Height="512.0" />
    <ScaleX>0.00388671875</ScaleX>
    <ScaleY>0.00388671875</ScaleY>
    <PanX>0.0</PanX>
    <PanY>0.0</PanY>
    <NumberOfRotations>0</NumberOfRotations>
    <IsVerticallyFlipped>>false</IsVerticallyFlipped>
    <FileName>imash_data/fetal-us.jpg</FileName>
    <Index>1</Index>
    <note> A 34-year old primigravida woman was
      referred to our center for evaluation
      of a possible cystic hygroma.
      Sonography revealed a 21-week fetus
      whose size was consistent with dates
      based on a prior ultrasound. A large
      cystic structure measuring 79 x 87 mm
      arising from the left mandibular/
      auricular area was identified.
    </note>
  </TeachingFileViewport>
</TeachingFile>
```

**Figure 4. Data characterisation of the Teaching File custom data type. (This text has been reduced and formatted for clarity.)**

regard, the DCF is similar to Sun's Javabeans Activation Framework [20], a Java-specific library extension that provides an API for typing data.

For the DCF recognition process to be possible, we make the assumption that every object can be sufficiently described so that its resultant attributes can be utilised later in the pipeline. We assume that information-extracting functions exist. For example, we have used the Java Image class to characterise image objects; for the Teaching File data type, we leveraged proprietary code (made available from the application's designers) to determine its attributes. In cases where objects cannot be described in any possible way, no transcoding is performed on the data.

XML's role as a data description language makes it a natural choice to describe the data object. In Figure 4 we show the XML-encoded data characteristics of a Teaching File data structure. Note that the structure is an aggregation of smaller items of data. Our system is capable of handling arbitrarily complex structures provided that the Data Characterisation Function has been configured to identify such data. If necessary, the system will recursively descend through sub-data types until all atomic items have been identified.

In our running example, the Teaching File data structure contains only one item that can be content-adapted, the JPEG image. In Figure 5 we show the XML description of the JPEG image within the Teaching File we showed in Figure 4. We note, though, that our recursive data characterisation would allow the DCF to identify aggregate data structures and to obtain the needed metadata to perform adaptation on arbitrary types, not just JPEG images. By recursively descending through an abstract data structure to find its constituent atomic data types (such as JPEG images), our algorithm can easily handle composite data. Such an approach is important because we have already encountered the use of proprietary data formats, such as the PACS [34] imaging format, in the medical domain we are studying. We note here that currently our implementation does not use the constraints defined in the pro-

```
<?xml version="1.0" encoding="UTF-8"?>
<data_characteristic>
  <fetal-us.jpg>
    <type>jpeg image</type>
    <num_colours>65536</num_colours>
    <dimension_v>480</dimension_v>
    <dimension_h>480</dimension_h>
  </fetal-us.jpg>
</data_characteristic>
```

**Figure 5. Data characterisation of JPEG image contained within the Teaching File custom data type. This particular image, fetal-us.jpg, corresponds to the image listed in the middle of the Teaching File data structure shown in Figure 4.**

files to apply an “overall” transcoding to an object of an aggregate data type. Instead, the recursive subdivision method applies the constraints to each atomic type. Although this is the easiest manner to handle data types with multiple hierarchical levels, in the future we will look into developing a methodology to apply the constraints to all the atomic elements at once using a weighted measurement. Such an approach was utilised in [26] to transcode an HTML page’s constituent elements.

As an optimisation in the DCF stage, object descriptions can be created off-line and stored with the objects at the AS before they are requested. This approach incurs extra secondary storage at the AS and communication overhead between the AS and the MWS, but we assume that the reduction in characterisation time at the MWS will be worth the tradeoff. In the event that the AS is a legacy system, we can store the characteristic files at the MWS. Note that this approach implies that whenever a data object is updated at the AS by some client application, the object’s characteristics must be immediately updated to avoid concurrency problems.

**DCF Extensibility:** This stage of the pipeline is easily extensible to support the identification of new data types. When a new object type is added to the system, the DCF can be provided with procedures that upon execution will properly identify a given object’s type and characteristics. An XML template for that specific type must also be provided so that the DCF can represent its characteristics. The DCF is further extensible by allowing changes to be easily made to identify a new characteristic of importance of a data type. For example, an image’s alpha channel may need to be considered in addition to its resolution and number of colours.

### 3.4 Adaptation Command Generator

The Adaptation Command Generator (ACG), the most complex stage of the pipeline, generates a set of commands to appropriately adapt a data object. Each command will invoke an adaptation function on the object later in the pipeline. The ACG takes as inputs the Client Profile XML file (representing the device and the user) and the Data Characteristics XML file. The ACG may optionally receive information regarding available bandwidth. The ACG function then starts comparing the restrictions in the client profile with the object characteristics described in the Data Characteristics file. If it is decided that any adaptations are needed to be made to the objects, the appropriate command, together with required parameters, is generated for each adaptation and stored into the XML file that will contain the adaptation commands.

We have developed a protocol by which the ACG makes decisions based on a set of rules to adapt the object to match all hard-

ware limitations set by the profile. For example, if the client profile specifies that the device can display at most 64 colours, a 128-colour image would have its colour bit-depth reduced to match the profile. Similar adaptation would be necessary to match attributes such as screen resolution, installed runtime codecs, and number of audio channels. Additionally, we are currently working on using available network conditions, such as bandwidth and jitter, as input factors into the ACG.

```
<?xml version="1.0" encoding="UTF-8"?>
<adaptation_command>
  <fetal-us.jpg>
    <type>jpeg image</type>
    <reduce_colours_to>256</reduce_colours_to>
    <reduce_v_res_to>300</reduce_v_res_to>
    <reduce_h_res_to>200</reduce_h_res_to>
  </fetal-us.jpg>
</adaptation_command>
```

**Figure 6. Adaptation commands produced from the Adaptation Command Generator.**

A sample set of adaptation commands is shown in Figure 6. These commands adapt the image characterised in Figure 5 to match the constraints of the PDA profiled in Figure 3.

**ACG Extensibility:** The ACG stage can be augmented to handle a variety of new situations. To handle new data types, the ACG can incorporate new heuristics in order to properly produce the commands needed to effectively adapt the content of the new data type. Changes in ACG heuristics can also reflect modifications in the client profile (e.g. the client hardware has changed or the user’s permissions have been reduced), in the Data Characterisation Function (e.g. an audio stream’s sampling rate needs to be considered in addition to its compression factor), or in the Content Adaptation Executor suite of functions (e.g. a new algorithm for transcoding JPEG images is now available).

### 3.5 Content Adaptation Executor

The Content Adaptation Executor contains a suite of procedures that perform the actual adaptation modifications on the objects. This stage performs three successive steps. It parses XML-encoded commands delivered from the Adaptation Command Generator, uses a lookup table to map the adaptation commands to adaptation library calls, and finally executes the adaptation function(s) with the proper parameters on the requested object. The final output is the newly content-adapted object that will be sent to the client.

We assume that the functions that perform the actual data transcoding already exist and can be used within our architecture. Algorithms to perform adaptation have been extensively researched and are beyond the scope of this paper. For the Teaching File application, we have transcoded the images in its data structure by using the graphical image manipulation functions provided by the Java imaging classes. As a final showing of our running example, we illustrate the final result in Figure 7 by showing the adapted image for the Teaching File implementation on the Compaq PDA. This application was written in Waba [35], a programming environment very similar to Java but intended for PDAs. Note that the data was adapted according to the profiling shown in subsection §3.2.

**Content Adaptation Executor Extensibility:** This stage of the pipeline can directly incorporate off-the-shelf previously de-



**Figure 7. The Teaching File prototype running on a Compaq iPAQ PDA. The device's profile was shown in Figure 3. From this profile, the MWS reduced the number of colours and the resolution of the image. Scrolling is needed to view the annotations.**

signed software components to perform the actual transcoding. We have found much success utilising the manipulation methods from the Java Image class to handle discrete images and the Java Media Framework to handle streaming data. We have further been able to incorporate open-source code to handle animated graphics and XML manipulation packages to adapt data already in XML form. Indeed, we emphasise here that any such functionality to handle data can be easily added into this pipeline stage.

## 4 Extensibility

The principal strength of our architecture is its extensibility to accommodate diverse data object types and content adaptation procedures, including those not yet known. Here we discuss our architecture design to support this goal in subsection §4.1 and then follow that up with three specific case studies of extensibility beyond the original goal of Teaching File data; we specifically augment the CAP to handle HTML (§4.2.1), animated images (§4.2.2), and real-time data streams (§4.2.3).

### 4.1 Extensibility by Design

Three fundamental design tenets of our architecture directly aid its extensibility. First, the CAP decomposes the necessary functionality into well-defined stages with clear interfaces, such that software components needed to process new data types and client requirements can be identified, written, and added to the appropriate modules. Second, our design utilises XML as a standard means of conveying information between CAP stages, making it easy to enhance functionality. For instance, new parameters can be added to a client profile simply by defining the appropriate tags and values. Third, the CAP is implemented in Java, allowing us to leverage its wealth of available binary-compatible software libraries. For legacy code written in other programming languages,

we have been successful in merging such code with Java by using the Java Native Interface library.

Extensibility can be viewed in two contexts: handling new data types and handling new client constraints. Extending the CAP to handle new data types is similar to adding a new MIME type and handler to a Web browser: the system must be able to identify the new type and be able to act upon it correctly. The addition of a new type requires the CAP to perform exactly three steps: (i) identify its characteristics, (ii) produce commands to adapt the content based on the object's characteristics, the client profile, and (optionally) network conditions, and (iii) perform the actual content adaptation. Not coincidentally, these three steps correspond directly to the stages of our architecture, namely the Data Characterisation Function, the Adaptation Command Generator, and the Content Adaptation Executor, respectively. Referring back to Figure 2, these are steps 3, 4, and 5, which are the exact, discrete, and well-defined stages where extensions are introduced.

Just as a web browser requires third-party vendors to provide dynamically loaded plug-ins to handle new MIME types, so too does the CAP require that the application engineer or iMASH administrator provide modules to allow: (i) the Data Characterisation Function to characterise the new type, (ii) the Adaptation Command Generator to logically form the proper commands to adapt the data to fit the constraints, and (iii) the Content Adaptation Executor to perform the transcoding of the new data type.

The system is further extensible by allowing changes to be easily made to support new client constraints. Suppose a new client device is introduced that requires even more restrictive content adaptation on an existing data type. The changes needed to be made would follow a similar course of action as before. The data characterisation function for this data type may need to be changed in order to identify a new characteristic of importance. Additionally, the adaptation command generator will need to take into account the new client device's profile as well as any new data characteristics to produce the correct commands. Also, a new adaptation function may need to be introduced in order to perform the adaptation effectively for the client.

Finally, we can add new procedures into our pipeline in order to perform additional functionalities. Adding a new procedure simply requires positioning the routine at the proper location within the CAP. For example, it may be possible to augment the system with code to perform encryption, compression, and application session handoff.

### 4.2 Extensibility case studies

As mentioned earlier, the CAP architecture was initially designed to support our first application, the Teaching File medical program, and its data types. In preceding sections we have shown how the CAP has been able to successfully perform its transcoding duty for the Teaching File. We now look at three specific case studies that demonstrate how the CAP was extended to handle data types we did not originally intend to support.

#### 4.2.1 Case study #1: XML and HTML

The first case study involved the adaptation of objects already formatted in XML. For simplicity, we took advantage of the fact that HTML is a subset of XML, and hence HTML files were used as the data objects being delivered. For sample data we decided to use the bookmarks file created by the Communicator 4.7x web browser from Netscape/AOL because this program maintains its bookmarks as an HTML file (and thus as an XML file). The objective of this case study was to extend the CAP to reduce the size of

the bookmark file by filtering out all bookmarks that had not been visited within the last two weeks. Such a transcoding would not only produce a new smaller file but would also presumably retain only useful links if we believe the most-recently-used heuristic is meaningful in this context.

The steps to extend the CAP correspond to the three steps we discussed in 4.1. When the user requests the bookmarks file from the Application Server, (i) the file should be characterised as an XML file. We extended the Data Characterisation Function by introducing code segments that would properly parse the data and identify it as XML (in this case, HTML).

Once the data was characterised, (ii) adaption commands were then generated by the Adaptation Command Generator by augmenting the ACG with new heuristics. The resulting generated adaptation commands specified that all the links that had not been visited within two weeks were to be filtered out. The age of the links can be determined because Communicator annotates each bookmark link with the date of its most recent visit. This simple transcoding directive could easily be replaced by an arbitrarily more advanced set of adaptation rules.

Note that this directive to filter two-week-old links cannot be derived directly from a standard user profile. This situation is an example of an application-specific transcoding that requires application-specific constraints in the profile; such an extension can be easily accommodated by simply extending the profile itself with new XML tags and values.

To perform the actual transcoding, (iii) we augmented the Content Adaptation Executor suite of functions with the use of XSLT, a language for transforming XML documents [39]. To run this stage of our pipeline, we created an XSLT document specifying the filtering rules previously discussed and executed its directives on the bookmarks file with the Xalan-C++ [37] open-source XSLT processor. The processor was able to successfully manipulate the HTML during the adaptation stage of the pipeline, thereby producing a reduced HTML document without complications.

We note that the Communicator bookmark file is but one example of a data object already encoded in XML. In the database community and in industry, XML is already used extensively to encode myriad data. Also, as noted, all properly written HTML files are XML files and can thus be transcoded successfully with the CAP. Individual atomic units within an HTML file, such as an image file, can also be transcoded after being identified during recursive descent.

#### 4.2.2 Case study #2: Animated Images

We next looked to extend the CAP by incorporating the capability to handle the graphic format of the Unisys/Compuserve GIF89a image standard. This format allows multiple graphic frames to be aggregated together to form one GIF file. Most modern web browsers have built-in support to present this image type in an animated fashion; some browsers even stream the individual GIF frames on-demand from the server. The objective in this case was to augment the CAP to transcode the individual frames of this new image type while still delivering it to the client as an animated GIF.

To adapt this new data type, (i) we first augmented our DCF to recognise it by virtue of its “.gif” extension. To produce adaptation commands, (ii) we utilised a fixed user profile that required the graphics file to be reduced in colour bit depth and in resolution; in this case study, we did not need to extend the ACG because such heuristics were already present to handle JPEG images. To transcode the GIF images, (iii) we added modular code into the Content Adaptation Executor suite. Individual frames (whether one or several) were identified via header information and ex-

tracted with open-source software published on Usenet. After each frame’s characteristics were identified (based on colour, dimensions, transparency, interlacing, and so forth), adaption commands were devised to transcode each frame to meet the specifics of the adaptation commands. The adaptation functions utilised the Java Image class methods to transcode individual frames, and open-source software aggregated the frames into a single GIF file again, ready for delivery to the client.

#### 4.2.3 Case study #3: Real-Time Streaming Data

The third significant extension added the capability to handle real-time streaming data sent using the Real-Time Transport Protocol (RTP), [33]. RTP is a transport-layer-independent end-to-end protocol that allows for the transmission of real-time audio and video sent via unicast or multicast. The protocol is a standard utilised in commercial products such as RealNetwork’s RealPlayer and RealServer. This case study’s objective was to transcode a stream to fit the multimedia constraints of the client and deliver it as RTP data. For simplicity, in our implementation we focused on adapting audio files encoded as MPEG layer 3 (MP3) and sent over RTP.

In our implementation (i) we identified and characterised the audio stream based on its encoding rate and number of channels. With that information, (ii) we created adaptation commands based on a fixed user profile to match the constraints of sampling rate and bitrate. We then (iii) executed the adaptation functions to re-encode the audio at lower rates.

Our implementation leveraged Sun’s Java Media Framework (JMF) 2.1.1 library [22]. The JMF is a robust package that supports capture, transmission, and display of several RTP formats for both audio (U-law, GSM, DVI, MPEG 1/2/3, etc.) and video (H.261, H.263, MPEG 1). Such a toolkit would be useful to us only if our architecture provides well-defined hooks where it can be incorporated, which the CAP does. The transcoding process is done with the JMF API by receiving an RTP stream as a DataSource, directing it into a JMF Processor, configuring and realising the Processor, and directing output to a DataSink. The transcoding is done on-the-fly to maintain real-time streaming delivery of the audio content.

Finally, for completeness, we note that the transcoding implementation was very effective in reducing the data payload. In our trials, we transcoded a 6.4 MByte MP3 file encoded at {44100 Hz, 128 kbps, stereo} to {22050 Hz, 32 kbps, mono}. The resulting file was 1.6 MBytes, a 75% reduction in size.

## 5 Scalability

Throughout the course of this paper, we have presented the use of the CAP running within the Middleware layer of the iMASH architecture (as was illustrated in Figure 1). The functionality provided by the CAP can easily be executed within a single Middleware Server (MWS) box. However, such an approach is not scalable and may impede performance under several conditions, such as increased number of clients requesting simultaneous service from the Middleware, increased frequency of requests from a given client, increased size of payload delivered per request, increased Middleware CPU service per request, and increased Middleware CPU or network load produced by external sources.

This problem may be potentially solved by intelligent use of pre-processing and caching. As the number of client profiles increases to represent available {user, device} pairs, the CAP must be able to handle this growth. The profiles are uploaded from the client devices only when the user logs on. Profiles can additionally

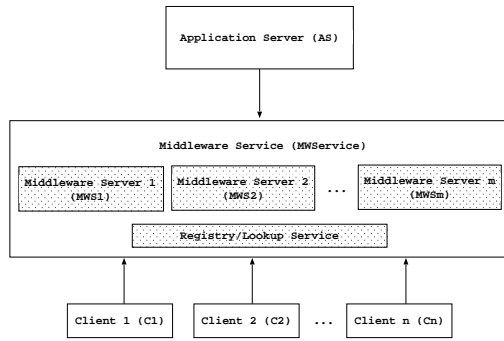


Figure 8. A broad view of the Middleware Service.

be cached to disk and kept in memory as the CAP executes. The Data Characterisation Function must be able to encode the characteristics of data into XML as each datum is requested by the user. These objects can be pre-characterised and cached at the Middleware Server. Finally, the content adaptation functions themselves are CPU-bound tasks that cannot take advantage of multithreading on a uniprocessor. Under some circumstances, an object can be pre-adapted or cached to ease load.

However, a more general solution is necessary to solve the performance degradation problems referred to above. Such a solution is provided by several servers acting as an aggregate Middleware Service (MWSservice) to provide the scalable functionality intended by the iMASH environment, as shown in Figure 8. Such an architecture involving the aggregation of separate computing stations is not novel and is well-known as a Network of Workstations [2]. Our MWSservice is formed from such a network of individual Middleware Servers that can be distributed over either a local or a wide area [31]. We will show that our distributed MWSservice scalably handles under increased offered load conditions by limiting the maximum service time experienced by clients to not grow without bound. This approach is similar to previous work on load-balancing [10] [25] in distributed computing.

Initially each MWS in the distributed MWSservice determines autonomously if it is capable of providing service by comparing a representative value, such as the well-known metrics of proximity to mobile clients, CPU load, service queue length, and available bandwidth, against a threshold quantity. We have implemented a simple policy based on CPU load in our experiments, as we shall show in §6. If a particular MWS is available to service clients, it registers itself with a registry service that will list all available MWSes for clients to utilise. This registry allows clients to discover a MWS within the MWSservice that can perform work on the client's behalf. If the MWS determines that it can no longer adequately provide service, it deregisters itself from the registry. When the MWS is again capable of operating sufficiently, it registers itself again. If more than one MWS is capable of fulfilling the client's request, a MWS is chosen randomly. If no MWSes are registered, several courses of actions are possible, such as having the lookup registry attempt to poll the MWSes for availability, conscripting a random MWS, or dropping the client's service request and returning an appropriate error.

Note that this approach immediately offers a distributed and scalable means of load distribution and admission control. Each MWS autonomously determines its availability by evaluating its own metric. By deregistering itself from the MWSservice registry, the MWS prevents itself from entering a state with an unstably in-

creasing workload. Additionally, the registry itself can be a scalable service (such as those using the Service Location Protocol [16] and Jini [36]) that is not subject to a bottleneck.

The MWSservice can also dynamically balance the load among the individual servers. Suppose a client  $C_i$  has been given service by a MWS, resulting in a portion of the client's session state being cached at the MWS. At some point the MWS may become unavailable and thus deregister itself to avoid any new service requests. However, the MWS should still provide service for  $C_i$  and any other clients that were already admitted. For these clients the MWS may need to enforce a load-balancing policy. To respond to such a situation, the MWS will perform a *service transfer* of the client's session to another available MWS. This target MWS can be found by the first MWS through the registry. At this point the client  $C_i$  should be informed that it must now interact with the target MWS. The client can attain this information through two means. The registry can give the address of the target MWS, but this would require that the client be able to asynchronously receive such updates from the registry as well as maintain its normal communication channel with the MWS. There is a better alternative: at the next instance when the client requests service from the initial MWS, this MWS will give the address of the target MWS. This on-demand updating can be done through the regular API interfacing the client and the MWS.

## 6 Performance

In this section we provide additional implementation details and present experimental results we have measured to analyse our architecture's performance.

All the functions in the CAP are written in Java with Sun's JDK 1.3. For the simple JPEG image adaptations used in our experiments, we have found that the standard Java library provides more than adequate image-manipulation tools found within the AWT classes. For the more complex data structures of the Teaching File program, however, we utilised custom classes provided by the programmer to identify and characterise this data type. All XML activity was handled with JDOM [21], a Java implementation of the Document Object Model. The JDOM classes allowed us to both encode and decode XML information throughout the CAP.

The interaction between the AS, the MWS, and the Teaching File clients use a variety of different communication protocols that stem from the Teaching File's legacy implementation. Originally the clients communicated directly with the AS through HTTP with Java Servlets running at the AS returning teaching files. With the introduction of the MWS, we have allowed the MWS to interact with the AS on behalf of the clients; this communication uses HTTP and Servlets as before. Now however, the clients communicate with the MWS through Java Remote Method Invocation (RMI), a choice made due to our familiarity with RMI's remote procedure call semantics. The fact that we could enable the Teaching File to utilise the CAP further shows that our architecture's design is capable of supporting more than just Web clients.

In experiments we quantified the overhead incurred by the CAP to perform adaptations on a JPEG image. Although the CAP is capable of handling more complex data types, we use a simple JPEG for our experiments as its size is easily controlled. We measured the time to complete the adaptation as a function of increasing size of the original object. Our target client was the Compaq iPAQ PDA, whose profile (from Figure 3) shows it as having a 300x200 screen and 256 colours. The original JPEG image was 1280x960 with thousands of colours and was scaled down to correspond to the image byte sizes on the x-axis. The times are a measurement

only of the CAP at the middleware; they exclude network transmission time between the client and the MWS but do include the time for communication between the MWS and the AS. Specifically, the measurements involve steps 2 through 5 of Figure 2. In these experiments our MWS was a 500 Mhz Pentium II desktop running Windows NT connected with high-speed ethernet.

For brevity we omit the resulting graph, but as is well known with transcoding, it is clear that the processing time for the CAP increases as the size of the original image increases, which is intuitively simple. Our results showed that even for the relatively large size of 300 kbytes, the completion time of the CAP is acceptable, under 1600 milliseconds. We note that our Middleware Server is a shared machine and that the executions incur network HTTP traffic to retrieve the image.

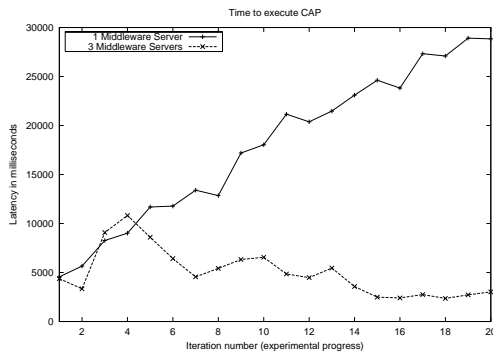


Figure 9. Latency for multiple Middleware Servers.

To demonstrate how we addressed the issue of scalability, we present results shown in Figure 9 that reflect the execution of the CAP distributed over three Middleware Servers (forming an aggregate Middleware Service, as discussed in §5). In this experiment, we structured the Middleware Service to perform the transcoding capability for a JPEG image as before. However, in this case, we additionally implemented a simple load-balancing scheme to demonstrate our Service Transfer algorithm. It is important to note that load-balancing is but one policy that will trigger the MWS-to-MWS response; as mentioned previously, other reasons can include client mobility and bandwidth considerations. In this implementation we used a heterogeneous collection of Sun UltraSPARC machines connected by high-speed ethernet as our Middleware Service. These older machines were chosen due to accessibility and convenience.

The Figure graphs the latency to perform the transcoding function as experienced by a given client. The x-axis represents the iterations of a given client during the experiment, and thus, the progression of time. In the experiment we launched a new client (requesting a JPEG) at random intervals based on a Poisson distribution with a mean of 15 seconds. Upon launching, each client requests the data object from the Application Server via the MWS and continues to make such requests at random intervals using a Poisson distribution with a mean of 30 seconds. We localised our measurements on one client, and therefore, each tick on the x-axis represents one request iteration for this client (after a random interval).

The data plot for the 1-MWS case shows that the latency seen by the client grows without bound as more clients are introduced into the system. This growth is due to the fact that the MWS must perform the computationally-bound task of JPEG transcoding. For the 3-MWS case, we implemented a load distribution policy of

having each MWS monitor its current CPU load by checking the average process queue length as returned by the Unix C library call `getloadavg(3C)`. When its load passes an arbitrarily chosen threshold, the MWS refuses further service. In the case of the client under observation in this experiment, it continues to request service from the MWS, but when the server becomes full, it is redirected for service at another MWS. With this easily implemented scheme, we see that the client's completion latency grows and then falls off, showing that the self-regulating load-balancing scheme provides scalability.

## 7 Conclusion and Future Work

Previous work has shown the benefit of having a middleware tier perform content adaptation, or transcoding, of data sent from an Application Server to heterogeneous mobile devices. Such functionality allows data content to be adapted, thus providing a suitable presented image for a given client device. However, such research efforts have focused on performing this adaptation in the context of the World Wide Web and its typically associated data types, such as text and graphic images.

In this paper we have suggested a new modularised and scalable architecture that provides content adaptation for not just Web-related data but for arbitrarily complex data, including aggregations of other data types. Furthermore, our system is not limited to only WWW applications and servers; it can be utilised as part of any type of client-server interaction using any communication protocol. We demonstrated its utility by implementing it to support its initial goal, a real-world medical application.

Our CAP architecture is extensible to support both new data types as well as new clients. To demonstrate this powerful extensibility, we augmented the system to support new data types not originally anticipated, including HTML/XML, animated GIF89a graphics, and real-time streaming audio. Finally, in order to have our system scale well with increasing offered load, we utilised a multiple Middleware Server architecture to distribute the service.

In the future we look to augment the CAP architecture in a variety of ways. Because our work has not focused on the content adaptation functions themselves, we will integrate more adaptation function implementations into the appropriate stage of our pipeline. Additionally, we will look into more complex applications to study, including a much more advanced version of the Teaching File that will be available in the intermediate future. We will also carefully study optimisation issues. Because our architecture is a pipelined process, it may be possible to parallelise the execution by distributing the individual component stages across different facilities. We also recognise that while the ACG can generate a set of adaptation commands to match all the constraints of the client, it may not be the optimal list of commands. In the future we will try to optimise this step by treating it within the context of a Constraint Satisfaction Problem. Finally, the user profile, with information regarding access rights and roles, must be kept secure; we will further explore this area as well.

## References

- [1] E. Amir, S. McCanne, R. Katz. "An Active Service Framework and its Application to Real-Time Multimedia Transcoding," In *Proceedings of SIGCOMM 1998*.
- [2] T. Anderson, D. Culler, D. Patterson. "A Case for NOW (Networks of Workstations)," *IEEE Micro*, February 1995.

- [3] O. Angin, A. Campbell, M. Kounavis, and R. Liao. "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking," *IEEE Personal Communications*, August 1998.
- [4] R. Bagrodia, M. Gerla, S. Lu, R. Meyer, D. Valentino, and L. Zhang. "Supporting Nomadic Healers," *UCLA Technical Report 200021*, 2000.
- [5] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. "An Active Transcoding Proxy to Support Mobile Web Access," In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.
- [6] E. Brewer, R. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hodes, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. "A Network Architecture for Heterogeneous Mobile Computing," *IEEE Personal Communications*, October 1998.
- [7] C. Brooks, M. Mazer, S. Meeks, and J. Miller. "Application-Specific Proxy Servers as HTTP Stream Transducers," In *Proceedings of the 1995 World Wide Web Conference*, December 1995.
- [8] "Composite Capabilities/Preferences Profile Working Group," [www.w3.org/Mobile/CCPP/](http://www.w3.org/Mobile/CCPP/)
- [9] S. Chandra and C. Ellis. "JPEG Compression Metric as a Quality-Aware Image Transcoding," In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, 1999.
- [10] T. Chou and J. Abraham. "Load Balancing in Distributed Systems," *IEEE Transactions on Software Engineering*, July 1982.
- [11] The Document Object Model Homepage. [www.w3.org/DOM/](http://www.w3.org/DOM/)
- [12] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*, Wiley, New York, NY, 1973.
- [13] R. Floyd and L. Steinberg. "An Adaptive Algorithm for Spatial Gray Scale," In *Proceedings of Society for Information Display*, vol. 7, no. 2, 1976.
- [14] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. "Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Prospectives," *IEEE Personal Communications magazine*, August 1998.
- [15] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, and E. Brewer. "Experience With Top Gun Wingman, A Proxy-Based Graphical Web Browser for the USR PalmPilot," In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK, September 1998.
- [16] E. Guttman. "Service Location Protocol: Automatic Discovery of IP Network Services," *IEEE Internet Computing*, July 1999.
- [17] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. "Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing," *IEEE Personal Communications magazine*, December 1998.
- [18] M. Hori, G. Kondoh, K. Ono, S. Hirose, and S. Singhal. "Annotation-Based Web Content Transcoding," In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Netherlands, May 15-19 2000.
- [19] Anita Huang and N. Sundaresan. "A Semantic Transcoding System to Adapt Web Services For Users With Disabilities," In *Proceedings of the ACM SIGCAPH Conference on Assistive Technologies*, Nov 13-15, 2000, Arlington, VA.
- [20] "The JavaBeans Activation Framework Homepage," [java.sun.com/products/javabeans/glasgow/jaf.html](http://java.sun.com/products/javabeans/glasgow/jaf.html)
- [21] "The JDOM Webpage," [www.jdom.org](http://www.jdom.org)
- [22] "The Sun Java Media Framework Home Page," [java.sun.com/products/java-media/jmf/](http://java.sun.com/products/java-media/jmf/)
- [23] A. Joshi. "On Proxy Agents, Mobility and Web Access," *Baltzer Mobile Networks and Applications* 5, 2000.
- [24] V. Korolev and A. Joshi. "An End-End Approach to Wireless Web Access," In *Proceedings of the International Workshop on Wireless Networks and Mobile Computing*, April 2001.
- [25] M. Litzkow, M. Livny, and M. Mutka. "Condor - A Hunter of Idle Workstations," In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
- [26] R. Mohan, J. Smith, and C. Li. "Adapting Multimedia Internet Content For Universal Access," *IEEE Transactions on Multimedia*, 1999.
- [27] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. "Agile Application-Aware Adaptation for Mobility," In *Proceedings of the 16th ACM Symposium on Operating System Principles*, 1997.
- [28] M. Orchard and C. Bouman. "Color Quantization of Images," *IEEE Transactions on Signal Processing*, vol. 29, no. 12, 1991.
- [29] T. Phan, K. Xu, R. Guy, and R. Bagrodia. "Handoff of Application Sessions Across Time and Space," In *Proceedings of the IEEE International Conference on Communications (ICC 2001)*, Helsinki, Finland, June 2001. [pcl.cs.ucla.edu/papers/](http://pcl.cs.ucla.edu/papers/)
- [30] T. Phan, R. Guy, J. Gu, and R. Bagrodia. "A New TWIST on Mobile Computing: Two-Way Interactive Session Transfer," In *Proceedings of the 2nd IEEE Workshop on Internet Applications (WIAPP 2001)*, San Jose, CA, July 23-24, 2001.
- [31] T. Phan, R. Guy, and R. Bagrodia. "A Scalable, Distributed Middleware Service Architecture to Support Mobile Internet Applications," In *Proceedings of the 1st ACM Wireless Mobile Internet Workshop (WMI 2001)*, July 21 2001, in Rome, Italy.
- [32] T. Phan, G. Zorpas, and R. Bagrodia. "The Convergence of Heterogeneous Internet-Connected Clients Within iMASH," *IEEE Personal Communications magazine*, April 2002.
- [33] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobsen. "RTP: A Transport Protocol for Real-Time Applications," *RFC 1889*.
- [34] D. Valentino. "Considerations in Implementing Large-Scale PACS," *Conference on RIS, PACS and Teleradiology: Future-Proof Solutions*, Lubbock, TX, USA, 1998.
- [35] The Wabasoft Homepage. [www.wabasoft.com](http://www.wabasoft.com)
- [36] J. Waldo. "The Jini Architecture for Network-Centric Computing," *Communications of the ACM*, vol. 42, no. 7, 1999.
- [37] "Xalan-C++ Overview," [xml.apache.org/xalan-c/overview.html](http://xml.apache.org/xalan-c/overview.html)
- [38] "The Extensible Markup Language webpage," [www.w3.org/XML/](http://www.w3.org/XML/)
- [39] "The XSLT Transformations webpage," [www.w3.org/TR/xslt/](http://www.w3.org/TR/xslt/)
- [40] M. Yarvis. "Conductor: Distributed Adaptation for Heterogeneous Networks," Ph.D. dissertation, UCLA, 2001.