PicturePiper: Using a re-configurable pipeline to find images on the Web

Adam M. Fass
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
Tel: 1-412-268-3620
E-mail: afass@cs.cmu.edu

Eric A. Bier, Eytan Adar
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304, USA
Tel:1- 650-812-{4439, 4758}
E-mail: {bier, adar}@parc.xerox.com

ABSTRACT
In this paper, we discuss a re-configurable pipeline architecture that is ideally suited for applications in which a user is interactively managing a stream of data. Currently, document service buses allow stand-alone document services (translation, printing, etc.) to be combined for batch processing. Our architecture allows services to be composed and re-configured on the fly in order to support interactive applications. To motivate the need for such an architecture we address the problem of finding and organizing images on the World Wide Web. The resulting tool, PicturePiper, provides a mechanism for allowing users access to images on the web related to a topic of interest.

KEYWORDS: dataflow, image retrieval, pipeline, WWW searching

INTRODUCTION
Users do many things with documents, including spell checking, language translation, format conversion, and printing. Typically, these operations are implemented as features in different programs, possibly on different platforms. Thus, a user wanting to spell-check a document, translate it to French and convert it to postscript may have to use as many as three different programs.

A solution to this problem is to implement stand-alone document services and provide a framework to combine them all. This framework allows applications to use services on any machine through a single uniform interface, regardless of the services' locations, the operating systems that they run on, etc. Such a framework is called a document service bus.

A document service bus may also provide a way to compose services in order to process a batch of documents. For example, the service bus could allow an application to compose the French translation and Postscript conversion services into a macro service, and then invoke this macro service on 100 documents in order to translate them all and convert them all to Postscript. The application is notified when the entire operation has been completed.

Despite their advantages, current implementations of document service buses are not well suited to some kinds of applications. For example, imagine a client application that lets a user search a large database of text in many languages. The user starts by entering a text query to find matching documents in the database. Since our user can only read English, he wants all of the matching documents in other languages translated. He then wants each one automatically summarized in a single paragraph.

Using a document service bus, a client could compose services for searching, translating, and summarizing and receive notification once all of the documents have been found and processed. However, current service buses don't allow the client to show the results to the user as they arrive. Even if the user could view the early results, the client cannot modify a batch process while it is running in order to change parameters and add services. Thus, if the user looks at the first few documents and decides that he wants longer summaries and he wants to filter out newsgroup postings, he must cancel the entire batch process and start over.

Thus, we see that service buses are not optimal for applications that process streams of documents and allow

UIST '00. San Diego, CA USA

the user to configure the processing on the fly. In this paper, we present a new kind of service bus architecture, a "pipeline service bus," which is better suited to applications with these needs. In a pipeline service bus, services can be invoked on an input source and they will process new documents as they become available. Services can be arranged in a pipeline so that the output of one service feeds into the next service. Services deposit all of their output in a document repository that an application can access in order to retrieve the results at any time. The application can add services to or remove services from a pipeline without disrupting the rest of the services.

Existing document service bus architectures allow us to easily integrate new document services, snapping them together like building blocks and making them available to users through a common interface. However, these architectures are commonly built to perform batch operations on static sets of documents. UNIX pipes and dataflow architectures allow users to combine operations to process streams of information, but they do not provide a mechanism that allows applications to discover which services are available and invoke remote services on different platforms. By combining pipelines and service buses, we are able to easily compose document services to handle streams of documents.

As a proof of concept, we have implemented PicturePiper, an interactive browser for finding images on the web. PicturePiper allows a user to interactively browse a stream of incoming images by repeatedly clustering them and selecting the clusters that look most interesting. PicturePiper is a fully functional image finding tool, and our purpose in building it was to demonstrate that a pipeline service bus architecture can support an interactive browser. Internally, PicturePiper uses a re-configurable pipeline of document services, each with its own thread of execution.

We will first give a high-level overview of our pipeline service bus architecture. We will then explain how PicturePiper works and use it as a motivating example to explain the architecture in more detail. Finally, we will point out other kinds of applications that could benefit from an interactive service bus.

PIPELINE SERVICE BUS ARCHITECTURE OVERVIEW

The pipeline service bus is a pipeline of document services, all of which run in parallel. Each service has its own thread of execution, which repeatedly reads in documents from its input buffer, processes them, and writes output documents to its output buffer. Currently, all services run on the same machine.

Each document service has an output buffer in which it places its output documents, as shown in figure 1. The output buffer for one service acts as the input buffer for the next document service in the pipeline. Each buffer also keeps a reference to every document that has ever passed through it. We refer to this collection of documents as that buffer's permanent store.

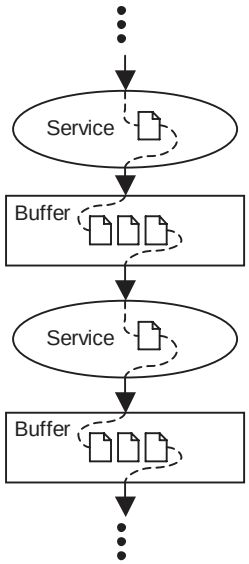

Figure 1: Each service in the pipeline service bus reads documents from an input buffer and writes modified or new documents to an output buffer. The output buffer of one service acts as the input buffer for the next service.

The input and output buffers described above are implemented by reading and writing documents to a document repository, as shown in figure 2. Our repository is organized into documents and collections. A collection is a recursive structure that may contain both documents and other collections, much like a directory in a file system. However, a document or collection can be simultaneously contained in any number of collections. Documents and collections have lists of associated name/value pairs called properties, where the values are arbitrary data. Properties can be used to store information such as the owner of a document or collection, the author, creation date, etc.

The repository provides a unique handle for each document contained therein, and it is these handles that move between services in the pipeline. The services are free to use the handles to retrieve the appropriate parts of the document and modify them if necessary. The output of a document service may be a handle for a document that it read as input, or the handle for a new document that it has created based on its input.

A client application such as PicturePiper interacts with the pipeline service bus through a well-defined interface. The client constructs an initial pipeline by adding document

services in the desired order, and the entire pipeline or individual services can be stopped and re-started. The client can remove the last service from the pipeline or add new services to the end of the pipeline.

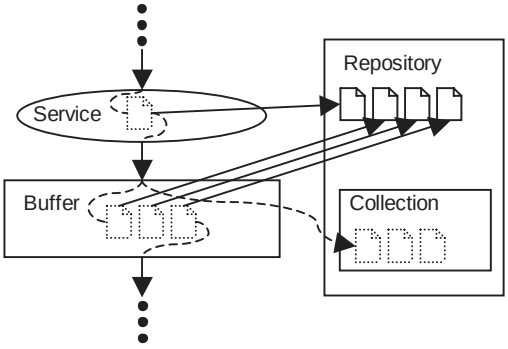

Figure 2: The pipeline service bus manipulates documents in a repository. Only document handles are passed from one service to the next. Each buffer uses a document collection as a permanent store.

As mentioned above, the output buffer for each service in the pipeline keeps a copy of all of the output documents for that service in a permanent store. These permanent stores are implemented as document collections in the repository, and the pipeline exposes these collections so that a client program can examine them. This is the only mechanism available to client applications for viewing the progress and results of the pipeline - there is no explicit notification alerting client programs that work has been done. This approach is well suited to dealing with a constant stream of documents, as the client application can get a complete update whenever it likes, and it is not bombarded with constant events or callbacks every time a service does some work.

PICTUREPIPER

In order to demonstrate that a pipeline service bus architecture can support an interactive application, we have implemented the architecture and written a client application called PicturePiper. In this section, we describe how PicturePiper works from a user's perspective. This discussion will provide concrete examples of the kinds of operations that our architecture supports, and we will later discuss exactly how these operations are implemented.

PicturePiper is a tool for finding images on the World Wide Web. It is useful for finding many images that relate to some topic of interest, and exploring and narrowing the large collection of results. The first thing a user sees when running PicturePiper is a dialog box that prompts him to enter a series of keywords, as shown in figure 3. The system sends these keywords to a web search engine such as AltaVista or Google, which finds HTML pages on the web that match the text query. The system scans the resulting pages for images, and allows the user to explore the resulting set.

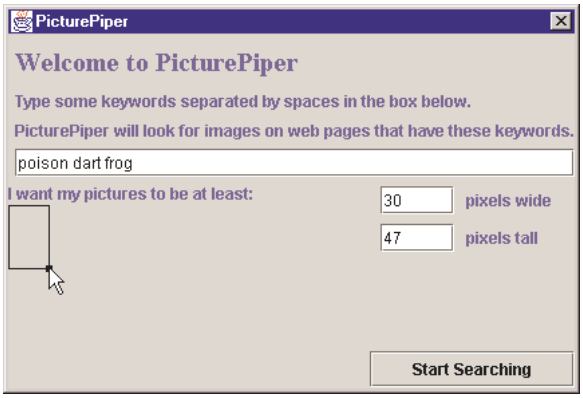

Figure 3: The user begins by entering a query and specifying a minimum width and height for the images.

PicturePiper displays the first few images that it finds in rows, as shown in figure 4. However, the set of images found on pages that match a query is often too large and eclectic for the user to be able to navigate with this kind of display. PicturePiper helps the user to explore and make sense of this collection using the Multi-Modal Scatter/Gather technique [5][6]. In this technique, the system first separates the collection of images into a fixed number of clusters, each of which contains images that are similar in some way. For example, each cluster may contain images that have similar colors. Other features on which PicturePiper clusters are image complexity (with one extreme being cartoon-like images and the other photographs) and textual features. Textual features are

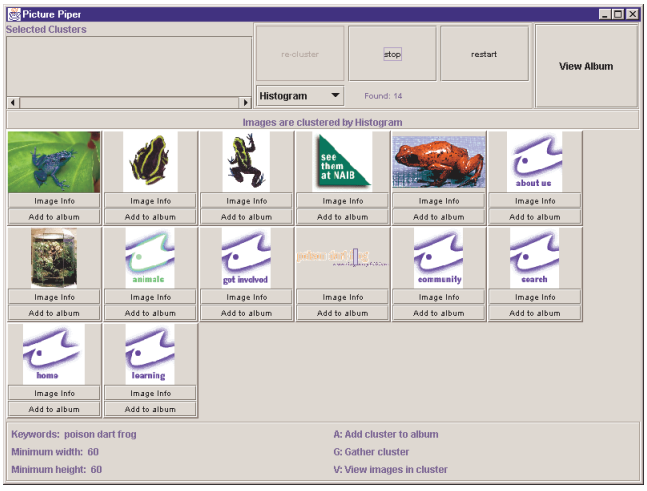

Figure 4: The first few images are displayed in rows.

generated by analyzing the web page from which the image was pulled.

Once the system clusters the collection of images, it displays them as shown in figure 5. Each cluster is represented by a widget that shows a representative image and has three small buttons. As shown in figure 6, the user can select interesting clusters, and the combined set of images can then be re-clustered using a different image feature. The result will be a new set of clusters, and the user can continue the process of selecting interesting clusters and re-clustering as many times as he likes.

PicturePiper chooses locations for the cluster widgets so that the distances between the widgets are representative of the similarities between clusters, i.e. clusters that contain similar images should be closer together in the window. Our system determines the coordinates for the widgets by using Multi-Dimensional Scaling [13], which takes the distances between the cluster centroids as input and outputs the suggested 2D coordinates for the cluster widgets.

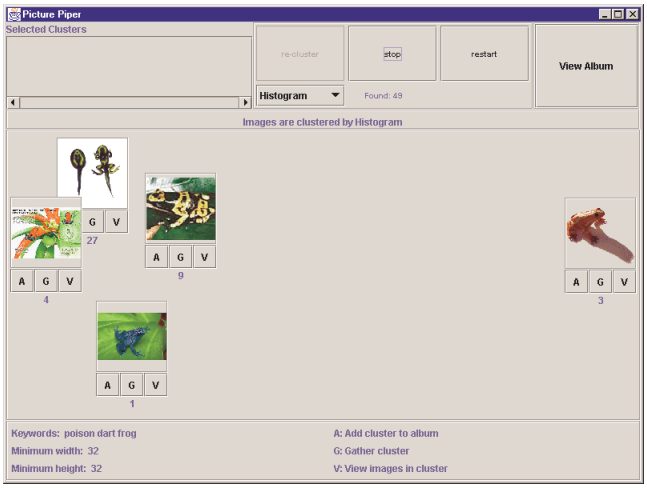

Figure 5: Once the system finds enough images, it automatically clusters them and presents a widget representing each cluster.

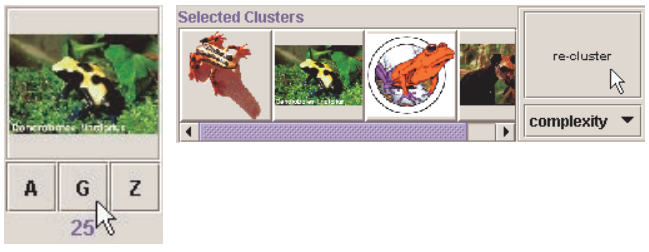

Figure 6: The user can gather clusters by pressing the G button, causing them to be displayed in the "Selected Clusters" area at the top left corner. The user can then re-cluster the gathered clusters.

PicturePiper differs from previous implementations of Scatter/Gather in that it is constantly finding new images on the web and adding them to the collection, even after the user has begun to use Scatter/Gather to narrow the results. PicturePiper computes clusters using whatever images are available at the time, and places new images in the existing cluster with the closest centroid as they become available. If a user has selected interesting clusters and re-clustered several times, the system compares the new image with the cluster centroids for each of these iterations. The end result may be that a new image appears in a cluster that the user is currently viewing.

PicturePiper can be used to find images that will later be used in written documents, presentations, etc. At any time, the user may decide to store a single image or an entire cluster in the PicturePiper "photo album", shown in figure 7, which represents the set of images that the user wants to keep. When the user is done exploring, he can write the contents of the photo album to a directory so that the images can be imported and used by other applications.

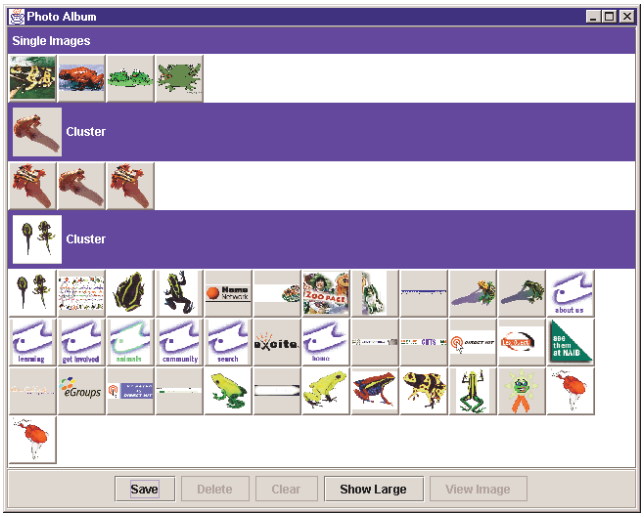

Figure 7: The user can add single images or entire clusters to the photo album.

HOW THE PIPELINE SERVICE BUS SUPPORTS INTERACTIVE PICTURE FINDING AND ORGANIZING

PicturePiper needs to allow the user to explore a collection of images while it continues the costly operations of searching the web, scanning the resulting HTML pages, downloading images, and processing them. The pipeline service bus is well suited to carrying out all of these operations in parallel, as it allows new documents to constantly flow through the pipeline even as the client program is examining the output of any service.

The Scatter/Gather technique requires an architecture that is capable of creating clusters from a set of documents or from a subset of the current clusters. The necessary operations are implemented as services that can be added to the end of the pipeline in order to cluster the appropriate set of documents at the user's request. We have created document services to find good cluster centroids for a set of documents, assign documents to the closest centroid, and filter out documents that do not belong to a selected set of clusters. Thus, PicturePiper can carry out any of these operations by adding the appropriate service to the end of the pipeline while the existing services continue to find and process new images.

In utilizing the Scatter/Gather technique a user applies data processing operations (filtering, clustering, etc.) to the current set of images. As new images are constantly flowing in from the web, it is required that the same data operations that were applied to the old images be applied to the new ones as well. Our system must therefore retain a memory of the operations previously applied to the images.

As part of the Scatter/Gather operation, PicturePiper leaves a pair of services in the pipeline to keep track of the selected clusters and the newly created centroids. Together, these services determine if a new document belongs to one of the selected clusters and which of the new centroids it should be assigned to. In this way, the pipeline represents a history of the user's activity and based on this history PicturePiper decides how to process new documents.

ARCHITECTURE DETAILS

Implementation Details

PicturePiper was written in Java 1.2 using Borland's JBuilder [3] environment. The Java Advanced Imaging API (JAI) [20] is used to process images in order to compute features for clustering. The ADC classes [8] are used to tokenize HTML pages. The Pipes API developed at Xerox PARC provides support for clustering and manipulating feature vectors.

Operations allowed by the service bus client

Clients interact with the pipeline service bus through a well-defined set of interface routines. The pipeline service bus provides several methods to add services to the pipeline, one for each different type of service. These methods instantiate the new service, create a new output buffer, and link the new service and buffer into the existing pipeline. Each of these methods takes a different set of parameters that are required for the service that is created. For example, the clustering service takes the number of clusters to be created and the feature to use for clustering as parameters.

In the current implementation, services can only be removed from the end of the pipeline. The service removal method stops the service if it is running and removes both the service and its buffer from the pipeline. Resources associated with the service and buffer are not necessarily freed, and the application can keep its handle to the service as long as it likes. The collection that served as the buffer's permanent store will remain intact unless it is removed from the repository.

Each service has a separate thread that executes a loop which reads input documents from the input buffer, does some processing, and writes output documents to the output buffer. The client can start or stop individual services using their handles, and this operation starts or stops that service's thread. The thread cannot be stopped immediately, as this can have unpredictable results in the Java thread model. Instead, a flag is set and the thread checks this flag every time it returns to the beginning of the loop.

The buffers implement a re-flow operation, which allows the client to take all of the data that has ever passed through a particular buffer and send it through again starting from that point. The re-flow operation simply takes all of the documents in a buffer's permanent store and places them in that buffer's FIFO queue so that the next service in the pipeline will read them as input. PicturePiper, for example, uses this operation to find cluster centroids for a number of images and then assign each of those images to a centroid, as discussed below.

The client can get a handle for the document collection that is being used as the permanent store of any buffer. The client can use the repository interface to get the size of a document collection or a list of handles for everything that is contained in the collection.

Example pipeline: the PicturePiper client

This section explains in detail how PicturePiper uses the pipeline service bus interface to download images, process them, and perform Scatter/Gather operations.

Initially, PicturePiper constructs a pipeline with the following services, as shown in figure 8a:

- Web Search
- Image Finder
- Image Downloader
- Feature Extractor
- Centroid Creator
- Multi Dimensional Scaling (MDS)

The first four stages are never removed and are always left running unless the user presses the "stop" button. The *Web Search* service is initialized with a query, which it

sends to a web search engine. It parses the HTML output generated by the search engine to find the URLs of pages that match the query, and it creates an output document for each URL.

The *Image Finder* service reads in these URL documents, retrieves the HTML pages that they refer to, and scans these pages for image tags. For each tag that it finds, it writes an output document containing the absolute URL of the image. The text near the image is added to the output document as a property.

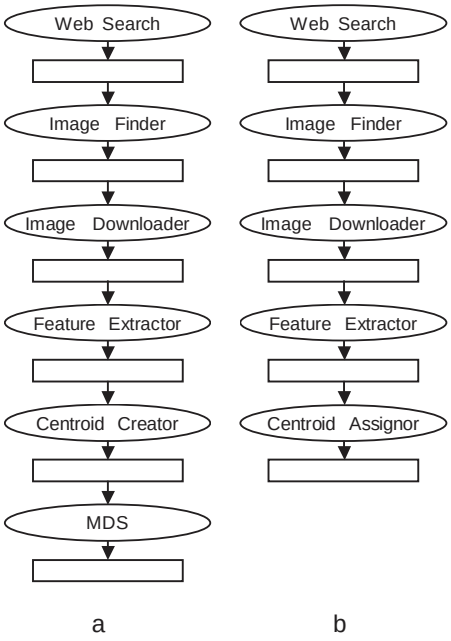

Figure 8a: The initial pipeline constructed by the PicturePiper client application.
Figure 8b: Once the Centroid Creator and MDS services are finished, they are removed and a Centroid Assignor service is added.

The *Image Downloader* service reads in the image URL documents, downloads the images, and stores them in a local image cache so that they can later be retrieved. This service does not create new documents or modify the input documents - it simply has the side effect of putting the image's bitmap in memory. Its output buffer receives the same URL documents that arrived in its input buffer.

The *Feature Extractor* service reads in the image URL documents and retrieves the image data from the cache. It then computes a vector for each of the available clustering features: color histogram, complexity, and associated text. The feature vectors are added to the image documents as properties.

To cluster the images coming through the pipeline, the client first adds a *Centroid Creator* service. The purpose of this service is to compute the cluster centroids, and not to actually separate the documents into clusters. The Centroid Creator service can compute centroids based on any of the features that have been computed by the Feature Extractor service, and the resulting centroids are vectors of high dimension.

This service waits until it reads in some minimum number of image documents, and it then runs the clustering algorithm. The Centroid Creator service creates *n* empty document collections, where *n* is the number of cluster centroids that it is asked to find. Each of these document collections represents one of the computed centroids, and the service adds properties to the collections indicating what feature was used in clustering and the coordinates of the cluster centroid. These centroid collections are written to the output queue as shown in figure 9, and Centroid Creator service thread stops executing.

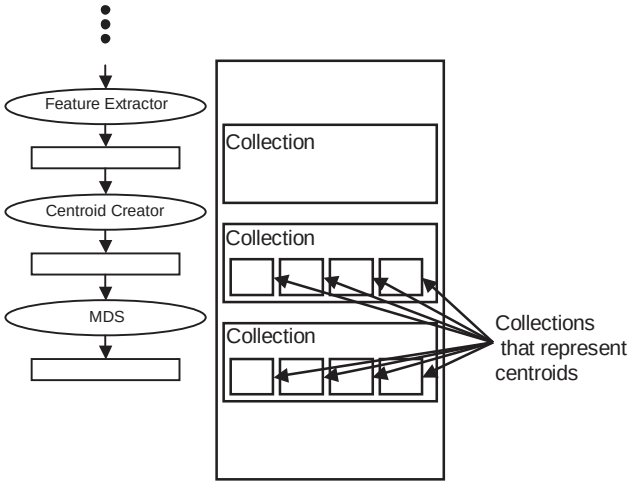

Figure 9: The permanent stores of the output buffers of the Centroid Creator and MDS services will contain collections that represent cluster centroids.

The Multi-Dimensional Scaling (MDS) service exists to provide 2D coordinates for each cluster centroid, which can be used to display the centroids in a window. It reads in the centroid collections and the high-dimensional centroid coordinates are used as input to the Multi-Dimensional Scaling algorithm to find the 2D coordinates. These coordinates are added to each of the centroids as properties, and they are then written to the output buffer. When the client program detects that this has been done, both the Cluster Creator and MDS services are removed from the pipeline.

Once the centroids have been found, all of the documents that were used by the Centroid Creator service and all

future documents must be assigned to one of the centroids. The client invokes the reset operation on the buffer that preceded the Centroid Creator service in order to insure that all of the documents will be assigned to a cluster, and it then adds a Centroid Assignor service, as shown in figure 8b.

The *Centroid Assignor* service is given the handle of the collection that served as the permanent store in the output buffer of the *Cluster Creator* service. This serves two purposes. First, this collection contains a set of centroid collections with information about where the centroids are. Second, the collection is used as the permanent store for the output buffer of the Centroid Assignor service.

The Centroid Assignor service compares each input document with the coordinates of the cluster centroids to find the closest one. A reference to the collection representing the closest centroid is added to the document as a property, and the document is then handed to the output buffer.

The output buffer of the Centroid Assignor service works differently from other buffers in the pipeline. As mentioned earlier, it uses a collection that contains a set of centroid collections as a permanent store. When adding documents to the permanent store, it checks the property added by the Centroid Assignor service to see which centroid collection the document was assigned to, and it is placed in the collection which represents that centroid. Thus, the collections that were created to represent the centroids will now contain the documents that are assigned to them. This is illustrated in figure 10.

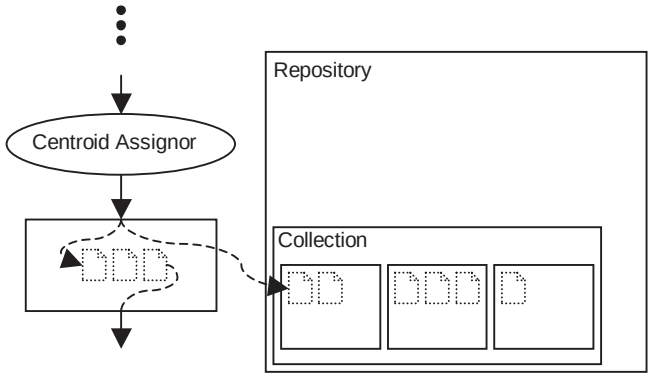

Figure 10: The output buffer for the Centroid Assignor service places each document in the collection which corresponds to the centroid to which it has been assigned.

At this point, the client can examine the permanent store for the output buffer of the Centroid Assignor stage to see all of the image documents that have been assigned to each centroid collection. As more images come through the pipeline, they are placed in one of these centroid collections and the client updates the main display by constantly checking the size and contents of these collections.

So far we have discussed how cluster centroids are created, how image documents are assigned to clusters, and how the client gets feedback about these events to update its display. After these events transpire, the pipeline consists of the first four services enumerated above plus a Centroid Assignor service. The client program displays a set of cluster widgets with the number of image documents contained in each cluster. At this point, the user can view the clusters and their contents, and he may choose to select a few interesting clusters and re-cluster only the images in those clusters.

The user presses buttons on the cluster widgets to mark clusters as interesting, but no action is taken until the user actually requests that the interesting clusters be re-clustered. When this happens, a Gather service is added to the pipeline and it is initialized with the set of interesting centroids. The Gather service reads image documents from the output buffer of the Cluster Assignor service and examines the property added by that service to see if each document has been assigned to one of the interesting clusters. Image documents that are in an interesting cluster are written to the output buffer, and all other images are discarded. Thus, the permanent store for the output buffer of the Gather service will contain only the images that have been assigned to one of the interesting clusters.

Once this has been done, a new Create Clusters service and a new MDS service can be added to the pipeline, and the process starts again. Every time new clusters are to be created, the Create Clusters and MDS services are temporarily added to the pipeline until they have finished their work. Each time the user selects interesting clusters and re-clusters them, a new Gather service and Centroid Assignor service will be left on the pipeline for the rest of the session.

Case studies of selected services

The services used by PicturePiper differ in the ways they use input and output buffers, in how long they run, and in the number of simultaneous threads that they use. This section exposes these differences by taking a closer look at some of the services used by the PicturePiper client.

A typical document service executes a loop that reads a document from the input buffer, makes a modification, and then writes the modified document to the output buffer. The Feature Extractor service is one example of this kind of service. The purpose of the Feature Extractor service is to calculate feature vectors to represent the image's color histogram and complexity, as well as a vector representing

the word counts for the text that was found near the image. This service reads in an image document and retrieves the content and the text property in order to calculate the feature vectors. The vectors are attached to the image document as new properties, and the image document's handle is then written to the output buffer.

The Web Search service is unique in that it does not read its input from an input buffer. This would be impossible, since the Web Search service is always the first service in the pipeline used by PicturePiper. The Web Search service contacts a search engine such as AltaVista or Google in order to get search results for a query, and it creates a new document for each of the pages that the search engine finds. The handles for these documents are written to the output buffer.

Other services are unique because they are not designed to process a continuous stream of documents, but rather to perform a single operation before they are removed from the pipeline. The Create Centroids service is an example of this. This service is initialized with a threshold number *t*, and instead of reading one document at a time it keeps reading all of the documents that are available until it has at least t documents. It then runs a clustering algorithm on the documents that it has, writes a set of collections representing cluster centers to the output buffer, and stops executing. The client program is responsible for detecting when the service is done and removing it from the pipeline.

Some services utilize more than one thread to process their input. This is important for services that need to download content from the Internet, as it allows them to process new input while waiting for the content to arrive. The Image Extractor service uses a fixed-size thread pool to find the image tags in several HTML pages simultaneously. The main thread for this service reads input documents and hands each one off to a worker thread. The worker threads are then responsible for writing the resulting image documents to the output buffer and then putting themselves back in the thread pool when they are done.

POTENTIAL APPLICATIONS OF A RECONFIGURABLE DOCUMENT SERVICE PIPELINE

The document service pipeline described in this paper can be modified by adding or removing services from the pipeline, even as the pipeline continues to process documents. Thus, a person or system relying on this pipeline can try different sequences of services without losing the work that has been done on documents that have already been processed by the pipeline. This capability combines well with applications where documents arrive in a stream, where documents need to be sent in a stream, or where compute-intensive services need to be performed on a large number of documents. In this section, we list potential applications in the areas of document scanning, document printing, electronic mail processing, and browsing document repositories.

Document scanning

Many companies are scanning paper documents and converting them to electronic form, by using optical character recognition (OCR). Given a stream of scanned documents, the pipeline we describe could be used to perform the various steps of conversion to electronic form. As the first completed documents arrive at the output of the pipeline, a human operator can look for errors. Often OCR algorithms will repeatedly make the same mistake, such as confusing "ri" for "n" in some fonts. The operator can describe the correct interpretation of characters to the OCR stage of the pipeline. Subsequent documents would then be processed with the improved OCR. In addition, the pipeline can be asked to re-process the earlier documents, so they also will benefit from the improvements.

Document printing

Many high-volume printing applications require printing a stream of documents, such as telephone bills or advertisements, where each document is customized in some way for its recipient. Such applications require a set of processing steps that merge in the variable data, convert the document to printable form, and send the document to an available printer. A reconfigurable pipeline would provide print shop operators with additional flexibility. For example, if another laser printer becomes available during the print run, a stage can be added to the pipeline that balances the print load across the updated set of printers.

Electronic mail

Electronic mail messages arrive as a stream to e-mail servers, which make them available to people using mail readers. Such servers may process the mail through a series of services before delivery. For example, e-mail messages may need to be checked for viruses, or have duplicates removed. Future e-mail servers may have richer services, such as converting attached documents into formats that are easier to read or even translating the message into another language. A flexible pipeline like the one described here would allow administrators or even individual users the ability to modify the processing stages at any time, so that future e-mail messages will receive handling that better serves each user.

Browsing document repositories

PicturePiper itself allows the user to browse images on the World Wide Web applying various filtering and clustering steps in order to find appropriate images. This approach can be generalized to browsing other types of media on the Web or in any document repository. For example, it could

be used to find news stories, corporate documents, digital audio recordings, or digital videotapes on the World Wide Web, in a corporate document repository, or from a commercial document repository such as a Dialog database. It could have particular benefits in using commercial databases, since it remembers the results of previous queries on the database and can be stopped and restarted at any time, potentially saving access charges compared to approaches that would download all of the information first and then filter and display it to the user.

RELATED WORK

The PicturePiper system described in this paper combines three technologies into a single system. It includes an image retrieval system based on key word queries, an image browsing technique based on multi-modal scatter-gather, and a reconfigurable pipeline for composing document services. In this section, we compare PicturePiper to related systems of all three kinds and to closely-related work.

Image retrieval

Like Alta Vista image search [1], and ditto.com [7], PicturePiper finds images on the World Wide Web. It differs from these systems in two ways. First, it not only presents the images to the user, but clusters them to help the user find the very best ones. Second, instead of fetching a page worth of images and then another as the user clicks, it keeps working on behalf of the user, downloading more and more images as time goes by, increasing the chances that the user will find a good image, and making it easy for the user to save a large number of images to disk at once if desired.

Image clustering

Like Query By Image Content [12] [2] [10], Virage's VIR image engine [21] and FotoFile [14], PicturePiper provides facilities for grouping images based on features. However, the image clustering in PicturePiper can be done repeatedly using the technique of scatter-gather [5] [6] so the set of candidate images can be narrowed down rapidly to a few likely candidates. In addition, PicturePiper uses multi-dimensional scaling to show the user, by spatial layout, which images are more or less similar to each other along the selected attributes. Hearst et al. [11] have found that relevant documents tend to be more similar to each other than non-relevant documents. The PicturePiper approach is motivated by our hypothesis that the same is true for images. While the feature queries of these other systems are more powerful for finding a specific image that the user has in mind, PicturePiper's clustering may prove superior for situations in which the user wants to learn what is available and then pick the images that best fit his/her needs. Rodden et al. recently reported on some experiments using multidimensional scaling on images [19]; their preliminary results showed an improvement in finding a general class of images over presenting the images in a random order. PicturePiper's display differs from the one in Rodden et. al, in that it uses multi-dimensional scaling on clusters of images instead of individual images to allow the user to get a feeling for a larger set of images than could be displayed on the screen at once as individual thumbnails.

Pipelines and document service buses

Pipelines and dataflow architectures have long been used in tasks such as data visualization. In the apE system [9], users interactively build a pipeline to create visualizations from scientific data. Later versions of apE allow cyclic pipelines, giving the user the ability to affect previous stages in the pipeline by interacting with the later visualization stages. Our pipeline service bus differs from apE in that it allows only linear pipelines, and stages can be added to and removed from the pipeline while it is processing.

The ATTICS system [15] is a framework for filtering and processing text documents. Like our pipeline service bus, ATTICS allows the addition of new classifiers, filters, and learning algorithms by constructing modules that follow an application program interface (API). In ATTICS, the user can combine pipeline stages by specifying them in a control file. However, applications built with ATTICS can not give the user any kind of interface to modify, add, or remove pipeline stages once processing has begun.

The Stanford University InfoBus [18] facilitates the interoperability of a variety of document services by providing a fixed object-oriented protocol. The services themselves may follow the protocol, or they may rely on proxy services. The InfoBus primarily addresses two issues: 1) allowing programs to interact with a wide variety of services, and 2) allowing clients to efficiently retrieve results from servers. Our current version of the pipeline service bus assumes that these problems have been solved and focuses instead on organizing many services to interactively process documents. Ultimately, a future version of the pipeline service bus should incorporate features of both architectures.

The pipeline service bus also differs from InfoBus in that it uses a document repository to store HTML documents, images, and any other information that is generated by document services. Thus, a client application like PicturePiper can be written without worrying about how data should be passed between services.

DLITE [4] is an application based on the InfoBus that allows the user to visually drag and drop document services, connect them to one another, and drop documents into them for processing. DLITE allows the user to directly manipulate and compose document services, where

PicturePiper presents the user with more abstract operations (gather, re-cluster, etc.) that may be implemented with one or more services.

Closely-related work

One system that is similar to PicturePiper in several ways is AMORE [16] [17]. Like PicturePiper, AMORE is a system that finds images on the World Wide Web and uses several kinds of clustering to display the images to the user. PicturePiper differs from AMORE in at least three ways. First, it uses scatter-gather so that users can pick a particular cluster to learn more about and use further clustering to do so. Second, it uses multi-dimensional scaling to show which clusters are more or less closely related. Third, it uses a pipeline so more images can be retrieved from the Web while the user is examining the images that arrived first.

SUMMARY

We have discussed a new kind of service bus architecture, the pipeline service bus, in which document services are arranged in a pipeline. The pipeline service bus can process continuous streams of documents and allow a client to view the partial results of any service. The client can re-structure the pipeline on the fly by adding or removing services. The architecture is ideally suited for applications in which a user is interactively managing a stream of data.

We built PicturePiper, a prototype application that can be used to find images on the web relating to a topic of interest. PicturePiper uses the pipeline service bus to allow a user to interactively explore a large collection of images while simultaneously finding new images on the web and processing them. PicturePiper constructs a pipeline of document services to find and process images, and it adds and removes additional services on the fly to carry out Scatter/Gather operations at the user's request.

We believe that the pipeline service bus architecture is applicable to a wide variety of tasks that involve processing streams of documents. Possible uses include refining a query based on early results, adjusting the parameters of an algorithm based on early results, and seeing data processed several different ways in order to compare the results.

ACKNOWLEDGEMENTS

Thanks to Francine Chen for writing the image feature extraction code, and to Michelle Baldonado for writing code to interface with the AltaVista search engine. We also thank Raj Iyer for suggesting that we use PicturePiper to search for poison dart frogs.

REFERENCES

1. AltaVista Company. AltaVista Extends Multimedia Search Capabilities With Rich New Content And Web's Largest Multimedia Index. Press release for February 7, 2000. http://doc.altavista.com/company_info/press/pr020700.shtml.

2. Ashley, J., Flickner, M., Hafner, J., Lee, D., Niblack, W., and Petkovic, D. The query by image content (QBIC) system. Proceedings of the 1995 ACM SIGMOD international conference on Management of data May 22 - 25, 1995, San Jose, CA USA, page 475.

3. Borland. JBuilder. http://www.inprise.com/jbuilder/

4. Cousins, S., Paepcke, A., Winograd, T., Bier, E., and Pier, K. The digital library integrated task environment (DLITE). Proceedings of the 2nd ACM international conference on Digital libraries, 1997, pages 142 - 151.

5. Chen, F., Gargi, U., Niles L., and Schutze, H. Multi-Modal Browsing of Images in Web Documents; SPIE Proceedings, 1999, pages 122-133.

6. Cutting, D., Karger, D., Pedersen, J., and Tukey, J. Scatter/Gather: a cluster-based approach to browsing large document collections; Proceedings of the Fifteenth Annual International ACM SIGIR conference on Research and development in information retrieval, 1992, pages 318 - 329.

7. ditto.com. ditto.com - the place for pictures. http://www.ditto.com/.

8. Do, A. The ADC package. http://www.do.org/products/parser/

9. Dyer, D. A Dataflow Toolkit for Visualization. IEEE Computer Graphics and Applications. Vol. 10, No. 4, 1990, pages 60-69.

10. Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., and Yanker, P. Query by Image and Video Content: The QBIC System. IEEE Computer, Volume 28, Number 9, pages 23-48, September 1995.

11. Hearst, M. and Pedersen, J. "Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results", Proceedings SIGIR ,1996.

12. IBM. QBIC™ -- IBM's Query By Image Content. http://wwwqbic.almaden.ibm.com/

13. Kruskal, J. B., and Wish, M. (1978). Multidimensional scaling. Beverly Hills, CA: Sage University Series.

14. Kuchinsky, A., Pering, C., Creech, M., Freeze, D., Serra, B., and Gwizdka, J. FotoFile: A Consumer Multimedia Organization and Retrieval System. Human Factors in Computing Systems, CHI 99 Conference Proceedings., May 1999, pages 496-503.

15. Lewis, D., Stern, D., and Singhal, A. ATTICS (poster abstract): A Software Platform for Online Text Classification. Proceedings on the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, 1999, pages 267 - 268.

16. Mukherjea, S., Hirata, K., and Hara, Y. Using clustering and visualization for refining the results of a WWW image search engine. Proceedings of the 1998 workshop on New paradigms in information visualization and manipulation, 2000, pages 29 - 35.

17. Mukherjea, S., Hirata, K., and Hara, Y. AMORE: A World-Wide Web Image Retrieval Engine. Human Factors in Computing Systems, CHI 99 Extended Abstracts, pages 17-18.

18. Paepcke, A., Cousins, S., Garcia-Molina, H., Hassan, S., Ketchpel, S., Roscheisen M., and Winograd, T. Towards Interoperability in Digital Libraries: Overview and Selected Highlights of the Stanford Digital Library Project. IEEE Computer Magazine, May 1996.

19. Rodden, K., Basalaj, W., Sinclair, D., and Wood, K. Evaluating a visualisation of image similarity (poster abstract). Proceedings on the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, 1999, pages 275 - 276.

20. Sun Microsystems. Sun Launches Java Advanced Imaging API, supports platform-independent imaging software.
http://java.sun.com/products/java-media/jai/pr/pr990615-27.html

21. Virage Incorporated. Virage continues to set the standard in multimedia asset management. May 1996. http://www.virage.com/news/may_1996_1.html