In Proceedings of the 5th ACM International Conference on Digital Libraries, June 2000, San Antonio, Texas.

Asynchronous Information Space Analysis Architecture Using Content and Structure-Based Service Brokering

Ke-Thia Yao, In-Young Ko, Ragy Eleish, and Robert Neches

Information Sciences Institute
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
E-mail: {kyao, iko, ragy, rneches}@isi.edu

ABSTRACT

Our project focuses on rapid formation and utilization of custom collections of information for groups focused on high-paced tasks. Assembling such collections, as well as organizing and analyzing the documents within them, is a complex and sophisticated task. It requires understanding what information management services and tools are provided by the system, when they appropriate to use, and how those services can be composed together to perform more complex analyses. This paper describes the architecture of a prototype implementation of the information analysis management system that we have developed. The architecture uses metadata to describe collections of documents both in term of their content and structure. This metadata allows the system to dynamically and in a context-sensitive manner to determine the set of appropriate analysis services. To facilitate the invocation of those services, the architecture also provides an asynchronous and transparent service access mechanism.

KEYWORDS: Data-driven brokering, asynchronous service access, content and structure, metadata, component architecture, information management, information analysis

INTRODUCTION

The GeoWorlds project at USC/ISI focuses upon a special niche in digital libraries: rapid assembly of custom collections of information for groups focused on high-paced tasks. Assembling such collections, as well as organizing and analyzing the documents within them, is a complex and sophisticated task. It requires understanding what information management services and tools are provided by the system, when they appropriate to use, and how those services can be composed together to perform more complex analyses. It is a significant challenge to provide a set of services in a manner that helps "non-wizards" perform these tasks. It is an even greater challenge to do so in an open and extensible environment in which new services can be added and made available. This paper reports progress on techniques that serve to meet these requirements. It complements previous work on federated systems [12].

GeoWorlds [4] is a component-based information management system that integrates various information organization and analysis tools together with a geographic information system[1]. Figure 1 illustrates some of the services within the current prototype of the GeoWorlds system. Within GeoWorlds, the Collaborative Information Space Analysis Toolset supports various services for gathering, analyzing, editing and visualizing information. These services help users efficiently organize *task-oriented information spaces* by helping them making sense of the data sources: characterizing them, partitioning them, sorting and filtering them [9].

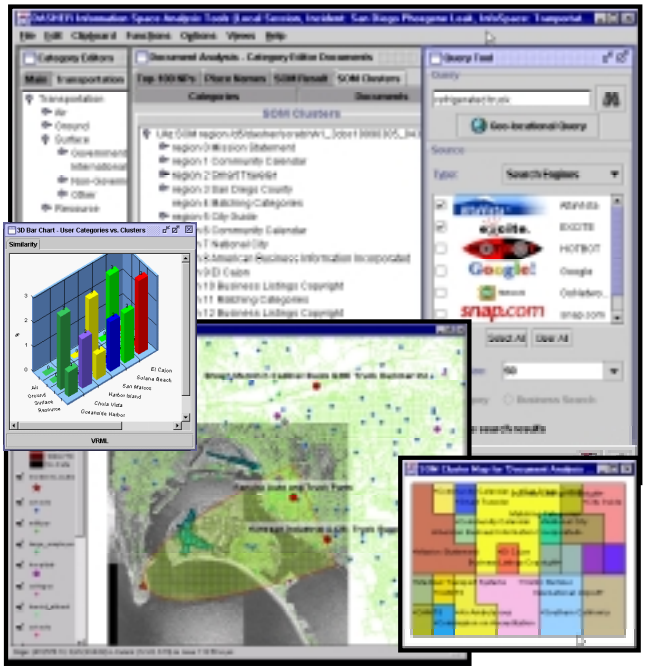

Figure 1: GeoWorlds Overview

The information management functions provided by the system can be classified into three major types: information gathering, information analysis, and information visualization. Using the *information gathering functions*,

[1] All the system components are implemented in Java (or Java wrapped legacy components); persistence is provided via XML serialization of Java objects.

users can extract relevant documents from various information sources such as Web search engines, Web directory services, on-line yellow pages, news video archive databases, etc. Then, the users can process the collected documents using the rich set of *information analysis functions* such as noun-phrase extraction, document clustering, category comparisons, language translation, etc. The source document collections and the analysis results can be displayed in various ways by the *information visualization functions* in the system.

Organizing a task-oriented information space involves cyclic application of these three types of functions. The user retrieves initial document collections from the information sources by creating topic-related queries and submitting them to the information gathering services. The initial data retrieved may be unorganized, or too big to browse, or even unrelated to the topics. A set of information analysis services must be applied to the initial document collections to reorganize, characterize and filter them. Then, the visualization components must be applied to the analysis results in various ways to help the user make sense of them. To fully organize a body of information, these steps must be repeated until the information space is sufficiently refined and populated to give users structured information relevant to the tasks that motivated their work.

During this cyclic process of organizing an information space, it is essential for the user to know which services are applicable to which document collections in the information space at any given stage of analysis. We seek to meet this need by providing a *data-driven service brokering* and an *asynchronous service access* mechanism.

The data-driven service brokering components provide context-sensitive matchmaking functions that use metadata descriptions about input data to identify appropriate services. The metadata description for a document collection is composed of two parts: a *content description* and a *structure description*. The content description represents the meaning (type) of the document collection (e.g. a yellow-page collection, a video document collection, etc.), and the structure description characterizes the document collection organization structure (e.g. a flat document list, hierarchical document clusters, etc.). By dividing the metadata description into these two different types, we can reduce the complexity of the description and simplify the classification and matchmaking processes. The metadata can be reused by multiple document collections that have same content type or organization structure.

Our system provides a uniform, asynchronous service access mechanism. Using this, services with heterogeneous communication methods such as Socket, Java RMI (Remote Method Invocation) [6], JavaSpace [5], etc. can be easily integrated to the system, and rendered transparently accessible by the client regardless of their locations and communication methods. Our asynchronous service access architecture is a layered architecture that is composed of the client layer, the job pool layer, and the service layer. Underlying services can be dynamically reconfigured without affecting the client layer. The architecture allows the services and job pools to be fully distributed over the Internet.

The sections following detail concepts and design issues in the data-driven and asynchronous service access architecture. We first provide an overview of the architecture, followed by detailed descriptions of data organization and service brokering, service access and invocation, and data visualization.

SYSTEM OVERVIEW

Our goal in designing the system was to meet the following requirements:

- **Transparency**. Clients requesting services should not need to know the name or the location of the server processing the request. The mechanics of selecting the appropriate servers and dispatching request should be hidden from the client. Similarly, the services should not have to care how the clients invoke them.
- **Specificity**. Transparency should not mean ambiguity. The client should be able to invoke precisely the desired service based on content and structure of the data.
- **Composability**. The architecture should allow services to be chained and executed together to handle single client requests.
- **Asynchrony**. The architecture should allow service requests to be handled asynchronously to facilitate batch processing. The client should be able to request a service, log off, and then log in again to retrieve the results.
- **Extensibility**. The architecture should facilitate adding new information services without requiring major changes to either the new or the pre-existing services.
- **Fault tolerance**. The system should not be disrupted by client or system service failures.
- **Reusability**. The system should be general enough to be useful in a broad range of applications.

As we discuss the design of specific aspects of the system in the sections following, we will relate them back to these criteria. The remainder of this section provides a general context for those design discussions.

Conceptually the system is organized into three parts: data, analysis services, and visualization services (viewers). The data is organized hierarchically. It defines the organization of the document collection that the analysis services use to process and the viewers use to display.

The system defines a set of basic documents that are viewed as primitive by the information management system. The basic document types include HTML, plain text, images, and video. The system is able to store metadata about these basic documents, such as title and URL, but it does not directly manipulate these basic documents. It relies on the services and viewers to manipulate and examine them. What

the system does directly manipulate are collections of these basic documents. The data hierarchy describes how these collections are organized. The collections are characterized by two orthogonal classifications: structure and content. The structure defines how the documents are grouped and their adjacency relationship. Some possible document structures are hierarchical, flat, and acyclic directed graphs. The content defines the information expressed in the collection. For example, a collection may represent the result of noun phrase collection or classification based on geographical location. An *information space* is a set of document collections.

The analysis services and the viewers operate on the document collection types in the data hierarchy. For example, our Category Editor viewer can display and edit hierarchical document structure, and the Map viewer can display geographical location-based document collections.

Figure 2 illustrates the system components and procedures involved in identifying available services using data-driven matchmaking, selecting specific services, and invoking the selected services transparently and asynchronously.

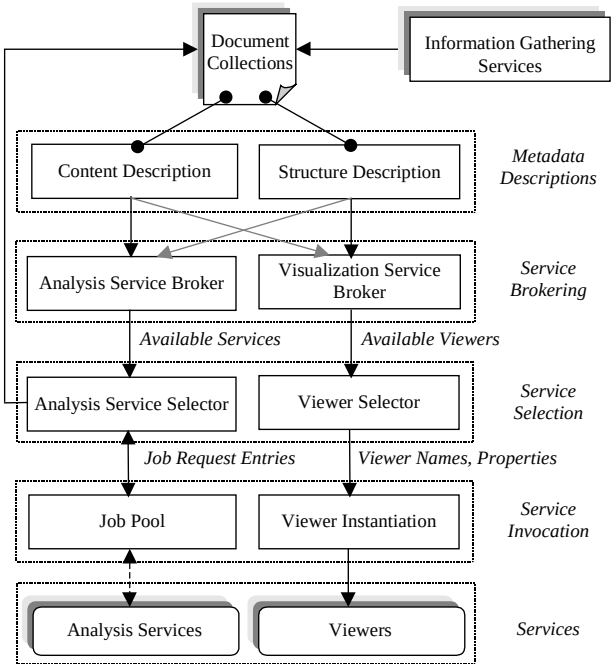

Figure 2: System components; service brokering and invocation procedure

As a result of performing an information gathering service or an information analysis service, a user gets a *document collection*. This is an organization of the resulting documents in certain structure with specific meaning in its content. The *service brokering components* use this content and structure information to determine which services are appropriate and meaningful for processing the document collection. The data generators produce *metadata descriptions* of the document collection, subdivided into content and structure information, and send those descriptions to the service brokers.

The *analysis service broker* matches the input document collection against analysis services registered with the system. It determines which can extract interesting characteristics from the documents and/or reorganize the collection by sorting and filtering. The *visualization service broker* finds alternative viewers for displaying the document collection. Matching service descriptions against the document collection, the analysis service broker mainly relies upon content descriptions; the visualization service broker emphasizes the structure descriptions.

There may be multiple available services that matched against the metadata description. Service selection allows users to choose among them. The *analysis service selector* or the *viewer selector* displays a service selection dialog, receives user's choices, and generates appropriate job request entries, or viewer names and properties, to invoke the corresponding services. The generated job request entries are sent to the *job pool* to invoke the analysis services asynchronously. The viewer name and properties will be used by the *viewer invocation component* to instantiate the selected viewers with the format specified in the property list. The analysis service selector registers appropriate job event listeners to the job pool to monitor the progress of the submitted jobs and receive results.

DATA ORGANIZATION AND SERVICE BROKERING

Our Collaborative Information Space Analysis Tool Set needs to handle diverse types of document collections generated by various information gathering and analysis services. Thus, the system has to provide efficient ways to manage the data, identify the types of the data, and use the data for invoking services. Since many difference document analysis and visualization services may be registered, meaningful and dynamic matches of the data against the services are also critical to help the user to select and perform appropriate services. To identify the type of data, we associate a metadata description with each document collection in order to find out appropriate services for a specific document collection. The service brokers use the metadata to perform data-driven matchmaking.

The system provides a uniform, graph-based internal representation for the document collections, by which we can represent the hierarchical document categorization structure, the title and properties of each category or document, and resource pointers (URL and the local cache pointer) of each document. However, each document collection may have a different content type and organization structure. This information is represented in metedata description about the document collection.

Metadata Descriptions

As the number of data types increases, it becomes unmanageable to classify and identify them by using simple

IETF MIME types[2] or Java class names. We developed an initial metadata description mechanism by which we can represent the *content* and *structure* information of a document collection.

The metadata description of a document collection is machine understandable [2] and helps the system perform more intelligent service selection and invocation. This makes it possible to build a more composable and extensible service brokering system that finds alternative services when a requested service is not available. Newly added data types can be identified and matched with the services without updating the algorithm.

The *content description* is metadata that represents the meaning embedded in the content of a document collection. For example, the result of a place name extraction service is a set of document clusters where each cluster node is a place name node which has a geographical coordination information as its property, and this type of content can be represented by the metadata type, *Place Name Document Clusters*. Figure 3(a) shows a sample content type hierarchy for sets of document clusters.

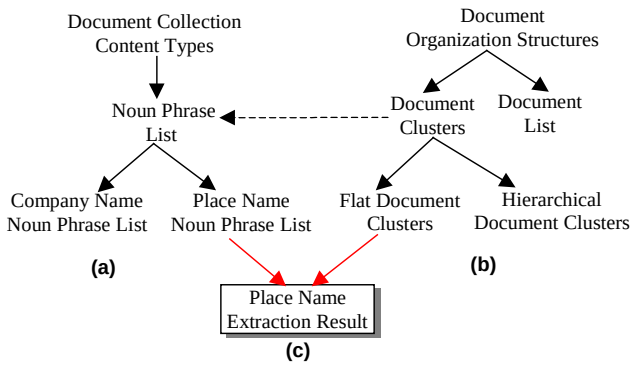

Figure 3: Examples of content and structure types - (a) a content type hierarchy for noun-phrase based document clusters, (b) a structure type hierarchy for document clusters, (c) a metadata description for the place name extraction result

The *structure description* is metadata that represents the document organization structure of a document collection. For example, a set of document clusters may have *Hierarchical Document Clusters* as its structure type, which is a structure where each cluster node contains documents or other sub-clusters as its children. Figure 3(b) shows a part of the structure type hierarchy for document collections.

Each metadata type has its own *ontology name* and *generic properties*. For example, a content description that describes a Web document collection extracted from a Web search engine has *Web Document Set* as its content type name and *Source*, *Summary* and *Rank* as its properties[3].

The metadata type (class) hierarchy shown in Figure 3 represents the ontology structures and the subsumption relations between metadata types. A sub-metadata-type inherits all the properties in its super-metadata-types. Currently, the subsumption relations and properties are not explicitly specified and stored in the system. The subsumption relations are encoded in each ontology name when it is defined. Properties are consented between the document collection generators and consumers. In order to represent more complex relationships between data and services and to support more sophisticated matchmaking system, we are currently developing a metadata description language that can explicitly represent the content and structure types, their subsumption relations, and properties using W3C's RDF (Resource Definition Framework) schema [14].

A content description may have embedded structure information. For example, the noun-phrase based document clusters have *Document Clusters* as their default structure type. The dotted arrow in the middle of Figure 3 shows this default structure relationship. Service brokers use this default structure description to match appropriate services if a document collection's metadata description doesn't include any structure information.

A document collection's metadata description is a set of content and structure descriptions. Multiple content or structure descriptions can be associated with a document collection. Figure 3(c) shows an example a metadata description for the place name extraction result that includes a content description which is an instance of *Place Name Document Clusters*, and a structure description which is an instance of *Flat Document Clusters*.

The division of the metadata description into the two independent types gives us the following benefits:

- **Simple ontology hierarchies**. Each ontology hierarchy is relatively simple than a unified one that has a deep tree hierarchy.
- **Easy classification**. We don't need to consider the structure types when we classify a data type under the content type hierarchy, and vice versa.
- **Reusability**. The content and structure types become more reusable because those are more general than the combined types.
- **Efficient matchmaking**. Depending on the service types, we can match one of the metadata type prior to another to improve the matchmaking performance (see next section for details).

[2] http://www.isi.edu/in-notes/iana/assignments/media-types/

[3] In our scheme, the value of *Source* property represents the search engine name such as AltaVista (www.altavista.com), Excite (www.excite.com), etc., and *Summary* or *Rank* properties have boolean values that indicates whether the summary text or rank information is available for each document in the collection.

Data-driven Service Brokering

Various *analysis services* have been integrated into the system to support: extracting certain characteristics from a document collection (e.g. noun phrase extraction, summarization, etc.), reorganizing the document collection (e.g. document clustering, category merging, sorting, etc.) and comparing different categorizations. We also provide a number of visualization components. Figure 4 summarizes the current basic document analysis and visualization services.

- Document Analysis Services
  - Keyword extractors: noun phrase extractor, company name extractor
  - Document summarization
  - Language identification and translation
  - Document clustering
  - Category manipulations: merging, sorting, filtering
  - Category comparison
  - Category fan-out (for Yahoo categories)
- Visualization Services
  - Category tree viewer
  - Document list viewer
  - Noun phrase list viewer (Frequency list view)
  - Working set explorer
  - Category editor
  - Color-coded classifier
  - 3D bar chart viewer

Figure 4: Basic document analysis and visualization services

As we apply the system to different types of tasks, we need to integrate more analysis and visualization services that are specific to the task domain. For example, we added a *Place name extractor* analysis service and *Locate documents on the map* visualization service to the system to relate a document collection to the geographical information system in GeoWorlds. We are currently integrating more services to link the information space analysis tools to a molecule browsing system and to a time-sensitive news article browser.

The service brokers in the system perform matchmaking between document collections and these various services. The matchmaking process is *data-driven* and finds *all and only* those services for a document collection that are feasible given its metadata description. The metadata description is composed of the content and structure descriptions, as we explained in the previous subsection.

As suggested by Carl Lagoze and Sandra Payette in their open-architecture digital library infrastructure proposal [8], the content-dependent manipulation of digital objects is one of the important requirements to build a reliable repository architecture for a digital library system. The *analysis service broker* in DASHER supports this feature by matching meaningful analysis services against the content description of the input document collection.

Most of our analysis services can accept some popular structure types such as *Document List* and *Hierarchical Categories*. Thus, matching content types is more effective than matching structure types to filter out irrelevant services. For example, all the document analysis services listed in Figure 4 can accept a document collection with *Hierarchical Categories* structure, but the *Category fan-out* service[4] can perform a meaningful analysis only when the category structure is composed of Yahoo[5] categories (i.e. the content type should be *Yahoo Hierarchical Categories)*. So, the analysis service broker always matches the content description prior to matching the structure description of the document collection.

Some analysis services such as *Category comparison* and *Category merging* require multiple document collections for their inputs. In this case, the services cannot be matched until the required collections are available.

The *visualization service broker* finds appropriate viewer components that can visualize the input document collection. In contrast to the analysis service broker, this broker considers the structure description first because in most cases, any document collection that has a certain organization structure can be visualized by a particular viewer regardless of its content type.

Currently, each service component maintains its own list of acceptable document collection types. The brokers query each service to check whether it can handle certain set of document collections. We are working on separating the capability description out of each service and collecting them into a metadata space. This will let us perform the matchmaking process on the metadata level without querying to the actual service instances.

Service Selection and Invocation

To process their data users, users are helped by service selectors to choose among services found by the brokers. A service selector displays a user interface that shows available service names and elicits user selections. It then maps the selected service names to the appropriate job request entries or viewer information. This is then sent to the service invocation components to access the services. Multiple selections can be made using the service selectors to perform services in parallel. We provide two service selectors - one for selecting the analysis services and another for selecting the viewers.

The *analysis service selector* performs mappings between service names and job request entries after getting user selections. Then, it instantiates the job request entries by

[4] The fan-out service provides the user a quick overview of a set of related categories in large category structures, such as Yahoo's category structure. Given a set of documents, the service first determines the categories the documents are classified under. Then it fans out by using the documents under those categories to determine other related categories.

[5] http://www.yahoo.com

including job IDs and other required arguments. The job request entries are sent to the job pool, which searches for and invokes service instances that can handle the requests.

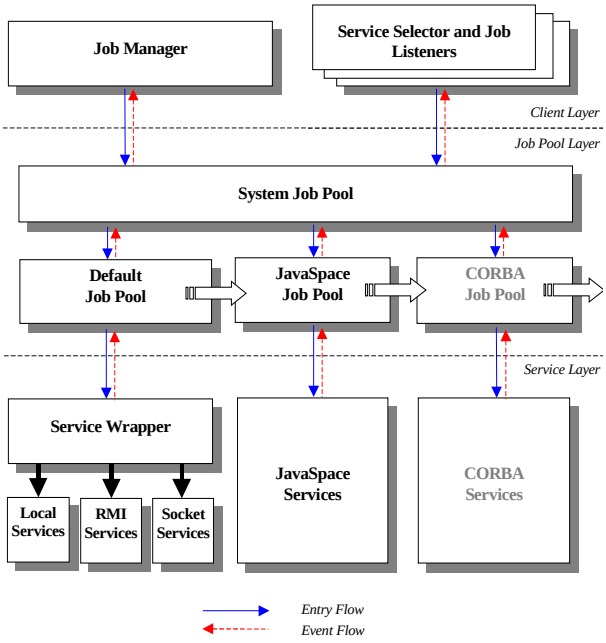

Figure 5: Asynchronous services access architecture

The input document collections can be sent to a service through arguments in the job request entry or by a separate control entry that conveys the input data. The service selector registers *job listeners* with the *job pool* to monitor status and receive the results (see next section for more details).

Some of the selected services may not be available at a certain time. Service availability can be checked by submitting its job request entry to the job pool. There are two ways to provide availability information to the user. The first method is to check availability of the matched services before invocation and show only available services. This is more reliable, but it may take a long time when there are many services matched. The second method is to check the availability after the user selects among the candidates. In this case, the service selector should ask the user about alternative services when the selected services are not available.

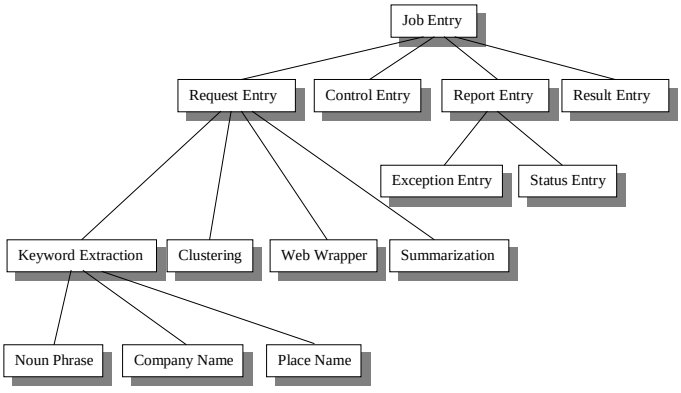

Figure 6: Job entry hierarchy for service/client information exchange. The Job request entry subtree defines a hierarchical organization of services.

The *viewer selector* collects the selected viewer names and forwards them with the input document collections and their metadata descriptions to the *viewer instantiation component*, which actually invokes the viewers. The instantiated viewers present the input document collections with the formats specified in their metadata descriptions.

The default viewer name for each structure type is maintained in the viewer selector. A default viewer is invoked if explicitly requested by the user or if the viewer properties are not specified in the input document collection metadata. The result document collections returned by the analysis services are usually visualized using the default viewers.

ARCHITECTURE FOR ASYNCHRONOUS SERVICE INVOCATION

The service invocation framework (Figure 5) provides the service selector with the set of available services, represented as job entries. The service selector uses these job entries to invoke the actual service. This paradigm facilitates *transparency*, because the service selector need not know how to invoke the service or where the service is being performed. The service invocation is performed *asynchronously*, which allows for batch mode processing. Also, the framework provides facilities for service providers to wrap their components as services and to modularly *extend* the system.

Job Entries and Events

A job entry is the basic unit of communication among client-layer components and underlining service-layer components. Job entry implements Sun's Jini [7] entry, so the Job Entries can be posted to Jini and JavaSpaces. Job entries are divided into four subclasses: request, control, report and result. Client components use job request entries to initiate service computation processes, and control entries to manage them. Control entry directives include cancel, return, pause and resume. The services report and result entries, wrapped as asynchronous events, back to the client components. Report entries contain intermediate results and status messages. Result entries contain the final result. They signal that the computation process is terminating. The client gives each computation process a unique job ID number. This number is used to channel control entries to the computation process, and to channel the report and result entries to the client. The arrows in Figure 5 depict the flow of entries and events.

The Job request class sub-hierarchy in Figure 6 defines the types of available services. *Keyword extraction* extracts keywords from document collection, *Clustering* groups similar documents together, *Web Wrapper* returns a collection from search engines, and summarization returns summaries for each document in the collection. Keyword extraction services currently subdivide further into noun

phrase, company name and place name extraction.

Job Pool Layer

The job pool layer is a two-level tree of job pools (see Figure 5) that manages and controls job processes for the client. Through the System job pool, the root of the job pool tree, the client can send job requests and job control entries to the services. Also, by registering as a listener, a client can receive report and result entry events. A client can register as one of three types of listeners. It can register to listen to entry events related to a specific job ID, related events in a job group, or to all job events. From the client-side perspective the job pool layer effectively uncouples the client-side service requestors from the server-side service providers. The clients do not need to know how the job requests are handled, nor who handles them. From the server-side perspective this *transparency* facilitated by the use of service interfaces (discussed more below).

The System job pool maintains a chain of lower-level job pools from which it delegates job assignments. For each job request entry, the System job pool successively queries the job pools in the chain to determine if it can handle the job. If a job pool responds in the affirmative, the System job pool delegates the job request to it.

All job requests made through the job pool are logged based on the job ID. The log contains information about which job request is still active, and which lower level job pool is processing it. This log is used to direct the flow of entries and events across the job pool layer. Also, this log is saved with the client's information space. This allows the client to retrieve job results from previous sessions.

Communication Mechanism

What differentiates the lower-level job pools from each other is the communication mechanism they use to exchange information with the actual service. We have implemented two lower-level job pools: default and JavaSpace. Depending on necessity, other job pool implementations are possible, such a job pool based on CORBA [11]. The default job pool is able to invoke services local to the client, services accessible across Java's RMI mechanism, and services accessible across sockets. The JavaSpace job pool uses Sun's JavaSpaces, a distributed message exchange mechanism based on Yale's Linda system [3]. The following paragraphs describe JavaSpaces and its job pool in more detail.

JavaSpaces supports just four operations: write an entry into the space, read an entry, take an entry, and notify an object when an entry is written. All four operations are implemented *asynchronously*. They do not block. Since the job entry hierarchy extends JavaSpace entry, any job entry can be directly submitted to the space. An entry, which is similar to a tuple in the Linda system, is a Java object that contains the data to be exchanged among any program communication through JavaSpaces. In contrast to the classless Linda system, entries are strongly typed and the object oriented inheritance relation is maintained. Entries are matched by their field values as well as their class types. This allows the space to match a super-class entry with a sub-class entry. For example, a client could use a *Keyword Extraction* job request entry to request *any* available keyword extraction service, or it could use a Company Name extraction entry to *specifically* request that service.

The JavaSpace job pool's job request processing protocol is as follows. The job pool writes the job request entry to the space, registers with the space to listen to job report and job result entries. On the server side, JavaSpace notifies the registered service of the availability of the job request entry. Then, the service takes the job request, and registers to listen to job control entries. After processing the job, the service writes the job result entry to the space. The space notifies the job pool that the job result entry is available, and the job pool takes the result entry.

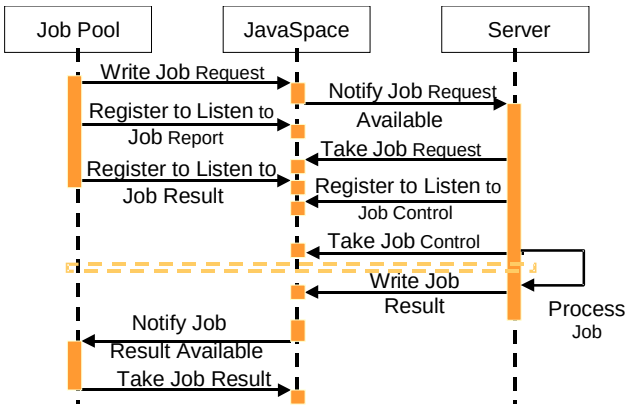

Figure 7: Asynchronous service invocation protocol

Multiple services can register to be notified when a job request entry is written. This provides a degree of *fault tolerance* to the system. If a particular service fails, other services may be able to take the job request. JavaSpaces also provides a transaction mechanism. If a service fails after it has taken the job request entry, JavaSpaces will roll back to a state before the entry is taken.

The space sends entry availability notifications in the form of a Java Remote Event. We have created a dispatcher class that listens to these Remote events, and converts them to regular Java Bean type events. The dispatcher delivers JavaSpace events in threads to enhance the response.

Service Interfaces

When job request and job control entries are sent by a user they eventually reach a particular service. A service is any object that implements the Service Interface. The interface allows services to be modularly implemented without regard to how they will be invoked or which communication mechanism they will use. Adapter classes are provided to allow services to connect with specific communication mechanisms.

To support this, services must *asynchronously* implement five methods: `getJobEntries`, `process`, `control`, `addServiceListener`, and `remove-ServiceListener`. When prompted using the `getJobEntries` method, the service must respond with an array of Job Request Entries it can handle. The service accepts a job request through the `process` method. The job execution prcess can be managed through the `control` method. Finally, the service must send the process output as events to all the registered service listener.

DYNAMIC VISUALIZATIONS AND VIEW TRANSFORMATIONS

While organizing an information space, the user typically iterates over many document collections of varying content and structure. Each of these collections can be visualized in various ways, each of which reveals different characteristics of the collection. To help the user better explore and understand the information space, the visualization service broker dynamically provides the user with a list of possible viewers for any given document collection. That list is based on the collection's content and structure. We call this *dynamic visualization.* Dynamic visualization satisfies both the transparency and specificity requirements. The user does not need to know the name of the visualization service to invoke the viewer, and the visualization services offered to the user are those specifically oriented to the document collection at hand.

Figure 8 shows alternative visualizations of a particular document collection, which is a clustering, generated based on place name extraction. The *tree view* of the document collection helps us to figure out how the classification structure was formulated. The *frequency list view* shows the distribution of the original documents over the place name nodes. The *map view*, which plots the document clusters on the map based on the place names, shows how the documents are distributed over the geographical regions.

Recognizing that these viewers are applicable to this data set involves using the document collection's structure and content metadata. In Figure 8, viewers (a), (b) and (c) can be matched by the broker using only the input data structure description. However, to match viewer (d), the broker needs to consider the content description because the map viewer can only visualize document collections with location information.

View Transformations

Each viewer keeps the original metadata description about the document collection being displayed. When prompted by the user, the viewer can use those descriptions to ask the visualization service broker to suggest a list of alternative viewers. The user can select among these to have the system create an alternative view. We call this *view transformation,* which is especially useful when the document collection has multiple characteristics that cannot be visualized in a single viewer.

Once alternative visualizations of a document collection are created, they appear in separate tabs of the same window

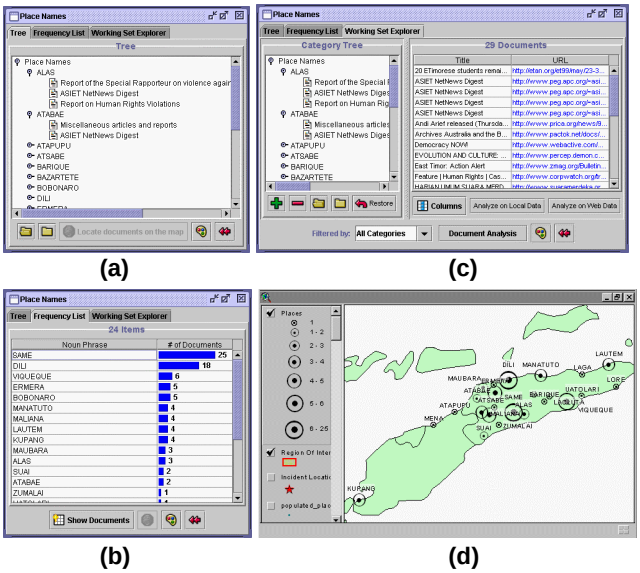

Figure 8: Various visualizations for place name based document clusters - (a) tree view, (b) frequency list view, (c) working set explorer, (d) geographical map view

frame as seen in Figure 8 (a), (b) and (c). Figure 8(d) is an exception because the viewer is an external application. Results of analyses performed on that document collection are also placed in separate tabs of the window frame. This is an attempt to localize and group in a consistent format various manifestations of the same document collection. When the user requests further processing in the next cycle of information space analysis, the analysis results are sent along with the original document collections to the analysis service broker. They can then be used to match for additional services.

Users cyclically perform information space analyses using these dynamic visualization and view transformation features. The process helps them to focus in on specific interesting parts of the information space, to populate the information space, and to extract useful characteristics from the document collections.

RELATED WORK

Like ours, the repository architecture for the open-architecture digital library infrastructure proposed by Carl Lagoze and Sandra Payette [8] also considers the content type and the structure type of a digital object separately, in order to provide manipulation functions sensitive to content or structure. However, they use the different metadata types to provide appropriate manipulation functions within a digital object. We use them instead to find higher level, external services.

The semantic and syntactic brokering concepts in MCC's InfoSlueth [10] are similar to our content and structure based brokering concepts. However, the brokers in InfoSleuth are not data-driven, and their semantic and syntactic descriptions are metadata about the target resources instead of the input data. The user agents in InfoSleuth have to explicitly specify the capabilities of the

required services in their requests. In contrast our users don't have to know which services are available and meaningful for their document collections at a certain stage of an information space analysis cycle. They need not specify any particular service types to analyze or visualize their data.

Using a distributed object framework, the Stanford InfoBus [1] provides a suite of information management protocols for uniform access to data and services. The protocols include data item and collection, metadata, search, payment, and rights and obligations management. We also provide uniform access to a variety of services, although the selection services that we choose to provide are different. The services we provide include keyword extraction, clustering, web wrappers, translation and summarization. The key difference between systems is that we provide a uniform way of accessing the services, instead of a suite of protocols. We are exploring whether this ability scales, or depends on offering a small, homogenous set of services.

EVALUATION

The first phase implementation of the GeoWorlds system is complete. We have delivered it to US Pacific Command Headquarters in Hawaii where it is receiving experimental use as an aid to intelligence information analysis. We are in the process of receiving feedback from them to improve the system. Recently, we have been invited back to install additional copies, and to train additional users.

To evaluate the effectiveness of the system at quickly creating useful repositories we have generated information spaces on a variety of topics, such as disaster relief (toxic chemical spreading in a harbor), economic impact analysis (ginseng distribution), and research trends (in mobile computing). Figure 9 depicts the process of creating an information space of humanitarian assistance in Indonesia.

FUTURE WORK

We are currently developing a new metadata architecture that can manage metadata descriptions more efficiently and perform matchmaking more dynamically. In addition to the current content and structure descriptions of document collections, metadata descriptions about ontology and capabilities of the analysis and visualization services will be provided.

We are developing a metadata description language using W3C's RDF, which provides common conventions for representing semantics, syntax and structure of a resource [13]. Using this metadata description language, we can represent subsumption relations between data or service types, type properties, and relationships between types.

The metadata architecture will provide a task-oriented metadata space that collects and combines descriptions for the data and services that are relevant to a task. The service brokers will perform reasoning on this metadata space to identify available services for certain types of document

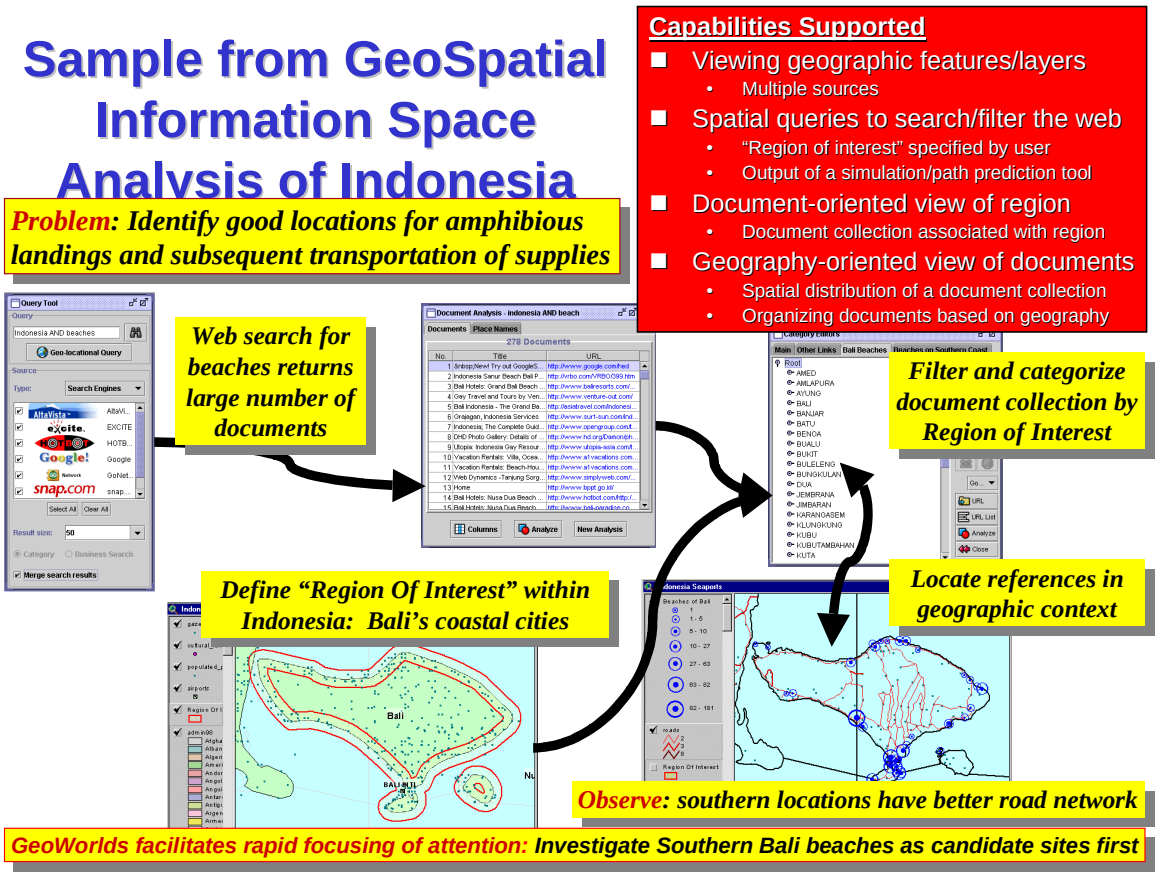

Figure 9. Sample information space analysis of Indonesia

collections without querying to the actual service instances. By combing this with a data fusion description language, we will enable users to compose virtual services that pipeline multiple services in order to achieve complex information analysis goals.

To facilitate more dynamic and integrated visualizations, we plan to re-implement all the viewers in a model-view-controller architecture. This will let changes or events generated in one view propagate to other views. For example, if the user highlights a category in the Category Editor, then the corresponding category in the Map Viewer will be highlighted as well.

We plan to extend the asynchronous service access mechanism in several directions, including a priority scheduling mechanism to sort jobs according to specific criteria, and an inter-linked web of JavaSpaces to allow users to access spaces that are not local to the network.

SUMMARY

The Collaborative Information Space Analysis Tools of GeoWorlds provides a semi-automatic information space analysis mechanism to help users organize task-oriented information spaces quickly and efficiently. The approach helps users overcome barriers of ignorance and complexity in information analysis tasks: users don't know *which* services are available, nor *how* to invoke the services. The data-driven brokering and the asynchronous service access architecture provide intelligent, dynamic and transparent service selection and invocation mechanisms. The service brokers utilize content and structure descriptions of document collections to provide efficient matchmaking. Representing content and structure separately simplifies the metadata description and improves its reusability. The architecture provides an asynchronous service invocation framework to facilitate transparent access to services. The client need not know the location of the services or the actual communication mechanism. A set of dynamic and consistent visualization features help users understand the meaning of the document collections. This assistance in using alternative visualizations of a document collection helps users find out important or hidden points that cannot be found using a single viewer.

INFORMATION AND QUESTIONS

For more information:

http://www.isi.edu/geoworlds/
http://www.isi.edu/dasher/

REFERENCES

1. Baldonado, M., Chang, C., Gravano, L., Paepcke, A., Metadata for Digital Libraries: Architecture and Design Rationale, Proceedings of the Second ACM International Conference on Digital Libraries (DL'97), Philadelphia, Pennsylvania, July 1997, pp. 47-56.

2. Berners-Lee, T. Metadata Architecture, W3C, January 1997. http://www.w3.org/DesignIssues/Metadata.html

3. Carriero, N., and Gelernter, D., “Linda in context”, Communications of the ACM, Volume 32, No. 4 (April 1989), Pages 444-458.

4. Coutinho, M., Neches, R., Bugacov, A., Yao, K., Kumar, V., Ko, I., Eleish, R. GeoWorlds: A Geographically-based Information System for Situation Understanding and Management, TeleGeo ’99, Lyon, France, May 6-7, 1999

5. JavaSpaces™ Specification, Sun Microsystems, November 1999. http://www.sun.com/jini/specs/js101.pdf

6. Java Remote Method Invocation Specification, October 1998. http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html

7. Jini™ Technology Architectural Overview, technical white paper, Sun Microsystems, January 1999. http://www.sun.com/jini/whitepapers/architecture.html

8. Lagoze, C., Payette S. An Infrastructure for Open-Architecture Digital Libraries, Cornell Computer Science Technical Report TR98-1690, June 1998.

9. Neches, R., Abhinkar, S., Hu, F., Eleish, R., Ko, I, Yao, K., Zhu, Q., Will, P. Collaborative Information Space Analysis Tools, D-Lib Magazine, October 1998. http://www.dlib.org/dlib/october98/dasher/10dasher.html

10. Nodine, M., Bohrer, W., Ngu, A.H.H. Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth™, 15th International Conference on Data Engineering, 23 - 26 March, 1999, Sydney, Australia.

11. Orfali, R. and Harkey, D. and Edwards, J. The Essential Distributed Objects Survival Guide. John Wiley & Sons, New York, 1996.

12. Powley, C., Benjamin, D., Grossman, D., Neches, R., Postel, P., Brodersohn, E., Fadia, R., Zhu, Q., and Will, P. DASHER: A Prototype for Federated E-Commerce Services. IEEE Internet Computing, Vol 1, No 6, November/December 1997.

13. Resource Description Framework (RDF) Model and Syntax, World Wide Web Consortium (W3C) Recommendation, February 22, 1999. http://www.w3.org/TR/REC-rdf-syntax/

14. Resource Description Framework (RDF) Schemas, World Wide Web Consortium (W3C) Proposed Recommendation, March 3, 1999. http://www.w3.org/TR/PR-rdf-schema/