SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 7, Number 4, pp. 101–117. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

A DISTRIBUTED CONTENT-BASED SEARCH ENGINE BASED ON MOBILE CODE AND WEB SERVICE TECHNOLOGY

VOLKER ROTH[†] , JAN PETERS[‡] , AND ULRICH PINSDORF[§]

Abstract. Current search engines crawl the Web, download content, and digest this content locally. For multimedia content, this involves considerable volumes of data. Furthermore, this process covers only publicly available content because content providers are concerned that they otherwise loose control over the distribution of their intellectual property. We present the prototype of our secure and distributed search engine, which dynamically pushes content based feature extraction to image providers. Thereby, the volume of data that is transported over the network is significantly reduced, and the concerns mentioned above are alleviated. The distribution of feature extraction and matching algorithms is done by mobile software agents. Subsequent search requests performed upon the resulting feature indices by means of remote feature comparison can either be realized through mobile software agents, or by the use of implicitly created Web services which wrap the remote comparison functionality, and thereby improve the interoperability of the search engine. We give a description of the search engine's architecture and implementation, depict our concepts to integrate agent and Web service technology, and present quantitative evaluation results. Furthermore, we discuss related security mechanisms for content protection and server security.

Key words. distributed systems, content based retrieval, mobile agents, Web services, content security

1. Introduction.

The availability of vast amounts of multimedia contents in the Internet requires sophisticated means for searching and retrieval. Current search engines are generally based on a centralized gatherer which traverses the hyperlinks of the World Wide Web starting from known entry points, and which retrieves and digests all relevant data found. This approach has two disadvantages: (a) it is data intensive, and (b) search engines cover only contents which are freely available for download. One might argue that transfer volume is not an issue because ample bandwidth is available on the Internet backbones. However, edge networks generally pay considerable penalties if they exceed their transfer volume quotas. They have an interest not to exceed their quota and to keep it as low as possible. Search engines should be designed to honor this desire. The second disadvantage results from the fact that commercial content providers loose control over the distribution of their intellectual property, once the search engine downloads it. Consequently, providers offer local searches and the number of algorithms they provide is likely limited.

In this paper we report on our distributed search engine prototype, which pushes content-based feature extraction (and optionally feature comparison) to the edge networks, and which can alleviate the aforementioned intellectual property concerns. The search engine is based on *mobile software agents* (see *e.g.* [52, 53] for an introduction to mobile agents). The benefits of mobile agent based feature extraction are:

- Multiple image sources can be processed in parallel. Each image source contributes to the processing power required to extract salient image features of its images.
- Multiple (*e.g.*, composable) feature extraction and comparison algorithms can be deployed concurrently and easily.
- Feature vectors are generally more compact than images, therefore less data must be transported from image sources to the gatherer.
- Feature extraction takes place at the image source. The images must not be exported from it, and original images generally cannot be reproduced from the feature vectors.
- The search algorithm can be provided by the searching client instead of the content providers. This leads to a decoupling of content and algorithm providers.

The achievable amount of parallelization and the reduced data transfer volumes have a significant positive impact on completion time of the feature extraction process. Disadvantages of the mobile agent based search engine are:

- Image sources have to set aside computing resources for feature extracting agents.
- Content providers have to host a middleware for mobile agents and give them access to their image content.
- The search engine's client has to provide a mobile agent middleware to distribute feature extraction resp. comparison algorithms, and to send further query agents for search requests.

[†]FX Palo Alto Laboratory, USA, volker.roth@acm.org
[‡]Fraunhofer IGD, Germany, jan.peters@igd.fraunhofer.de
[§]Fraunhofer IGD, Germany, ulrich.pinsdorf@igd.fraunhofer.de

- Running mobile code on a server poses a considerable security risk. Therefore, security of the mobile agent middleware is an essential requirement for the practicality of the approach.

In case, the feature extraction process and the distribution of corresponding feature comparison algorithms to image sources is realized by means of mobile agents in an initial index phase, following search queries need not necessarily rely on the same interaction paradigm. As described in §4, recent research results allow us to implicitly and transparently describe resp. provide access to distributed algorithms by means of the Web service protocol suite. According to the application scenario (confer §2), this fact relativizes some of the above mentioned disadvantages:

- Presuming that distribution of feature extraction and comparison algorithms is done by a small group of experts using mobile code, subsequent provision of these specialized algorithms can be done for a broad group of users using Web service compliant client applications.
- Further assuming that the level of trust between image providers and algorithm providers is higher than the level of trust between image providers and end users of the search engine, image providers can clearly distinguish the security policies and mechanisms bound to the respective interaction paradigm (mobile code vs. Web service technology).

We built our search engine prototype on the mobile agent server *SeMoA* [41]. Although SeMoA supports a rich set of security features, we do not claim that its security is perfect—the fact that it is programmed in Java alone renders it vulnerable to a variety of *Denial of Service* (DoS) attacks [7]. However, SeMoA provides a rich set of cryptographic features to protect the data of agents (*e.g.*, collected images) against disclosure on untrusted hosts, and an architecture that emphasizes separation of agents.

The basic concepts of mobile agent based image search engines have been mentioned by several authors before [6, 2, 39, 54] but have not yet been addressed in sufficient detail and in the context of a practical system. In this paper, we contribute a nuanced discussion and comparison of operation modes, a description of our implementation, and a quantitative analysis of the benefits of our search engine.

2. Concepts and Models. There does not seem to be a universal understanding when a program crosses the border to agent-hood. Software agents are often defined as being *reactive*, *autonomous*, *goal-oriented*, and *continuous* [18] though further attributes exist. *Mobile* agents have the ability to relocate—at some point of their execution they can halt and initiate a *migration* to some distant host where they resume execution. During migration, an agent's program as well as its current execution state and accompanying data is transported to its new host. When and where an agent migrates is part of the agent's program. In general, mobile agents rely on an infrastructure of mobile agent servers which handle agent transport, setup and deinstallation. What mobile agent technology brings to bear on the problem of image indexing and retrieval is easy means of *software distribution*. Briefly, mobile agents provide a flexible and easy mechanism to transport content-based feature extraction and matching algorithms to the source of the images rather than vice versa. The impact on network utilization and scalability if profound—instead of putting the burden of gathering contents completely onto the shoulders of a centralized gatherer and its connected network interface the load is shared among the gatherer and the image servers and all image servers can be indexed in parallel.

The W3C defines a Web service architecture [51] as a software system designed to support interoperable machine-to-machine interaction over a network. Thereby, a Web service is an abstract notation which is implemented as a computational resource. These Web service compliant modular software components are described, published and invoked by means of a few core specifications, namely *Web Service Description Language (WSDL)* [11], *Universal Description, Discovery and Integration (UDDI)* [32], and *Simple Object Access Protocol (SOAP)* [50]. Based on these specifications a variety of new protocols and standards have been evolved, addressing workflow definition and service federation, publish-subcribed messaging and transactions, as well as reliable messaging and security. In turn, this Web service protocol stack provides the foundation for current distributed service-oriented architectures, already impacting a broad range of commerce and industry. Integrating mobile agent and Web service technology, as described in details in §4, results in a middleware of the image provider which is able to accept client-specific feature extraction and matching algorithms as mobile code components, and in turn, provides this functionality to a broad range of Web service compliant applications.

Below, we illustrate two slightly varying models of image search engines based on mobile agent and content-based retrieval technology. The first model resembles an optimized gatherer which still has a central repository of feature vectors (though feature extraction is done remotely). We refer to this model as the *gatherer model*. The second model keeps a distributed index and no data needs to be shipped over the network during indexing. We refer to this model as the *incubator* *model*[1]. Furthermore, we present an *extended incubator model* comprising Web service technology to provide the actual search functionality to a broad group of end users. The general search is always threefold. First, a number of index agents sent by the broker are responsible for feature extraction at the content servers (*index phase*). Second, a search agent is used to find images which match a user defined query based on the pre-calculated feature vectors (*search phase*). Finally, a fetch agent collects the desired images from the according image servers (*fetch phase*). Alternatively, the last two steps can be replaced by Web service interactions.

Below, we describe and discuss the three models. In §3 and §4, we explain in greater detail to what extent and how we have implemented the models in our prototype.

2.1. The Gatherer Model. The gatherer model consists of a central image broker, several image servers, *index agents*, *search agents*, and *fetch agents* (all agents are mobile and relocate during their life cycle). The image broker dispatches index agents which transport feature extraction algorithms to one or more image servers. On these servers, the index agents extract relevant feature vectors from local images and send or take image entries back to the image broker. At the image broker, all image entries are merged into the central index (see Fig. 2.1 for illustration). Each image entry consists of a feature vector, the URL pointing to the host where the image was retrieved, an image ID that uniquely identifies the image at the image server, an optional thumbnail, and optional further information on the image such as its size or licensing information. The globally unique ID of an image consists of the globally unique URL plus the locally unique image ID.

Based on the index data, image brokers can either serve requests in a client/server fashion, or they support mobile agent queries as follows. A client sends a search agent to the broker which queries for similar images by means of an example image, a sketch, a prototypical image, or a feature vector which is extracted from either of these query images. The query result consists of extended image entries which contain the normalized distance between the query and the entry's feature vector in addition to the entry itself. The search agent transports the result set back to the client who selects images for retrieval based on the included thumbnails, or refines the query (*e.g.*, by means of relevance feedback). Once an image is selected for retrieval, the client sends a fetch agent that migrates to the server on which the image is stored (directed by the URL which is stored in the image entry) and retrieves the image based on the local image ID also contained in the entry (see Fig. 2.2). This can be preceded by a negotiation phase in which agent and image provider agree *e.g.*, on licensing terms and the payment of license fees.

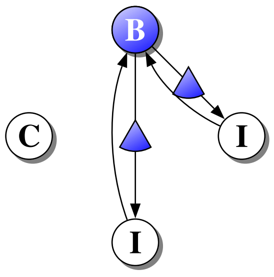


FIG. 2.1. *Gatherer model (index phase): The broker (B) dispatches feature extraction agents (denoted as triangles) to the image sources (I). Once the agents completed extracting the features of all images, they carry the feature vectors back to the broker.*

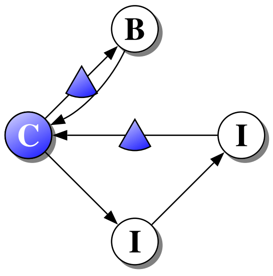


FIG. 2.2. *Gatherer model (search/fetch phase): The client (C) sends a search agent to the broker (B), which retrieves image entries matching the client's query, and transports the entries back to the client. The client selects images for retrieval, which are subsequently collected from the image sources (I) by the fetch agent.*

The transfer volume savings of this model are proportional to the compression factor of the feature extraction algorithm. On the one hand, a constant overhead incurs because the feature extraction algorithm must be transported to each image server. On the other hand, only one network connection request is required for transporting the agent—compared to one connection per image in the case of ordinary gatherers (unless the gatherer and the image server support sessions). Content providers retain control over their intellectual property because only feature vectors are exported, from which the original image generally cannot be reproduced in high quality.

2.2. The Incubator Model. The incubator model can do even better than the optimized gatherer model. In the incubator model, one index agent per image server is dispatched and takes residence at the image server. There, it extracts features as previously described but sets up an index directly at the image server (see Fig. 2.3). The index agent may

[1]Incubator: "A place or situation that permits or encourages the formation and development, as of new ideas." The American Heritage Dictionary of the English Language, Fourth Edition

also monitor the local image repository for changes, and it can update its index accordingly and incrementally. The only communication between the broker and the index agent is a short notice that the computation of the index is completed and the index agent is ready to provide service to search agents. The image broker is still a central point of access but it resembles more a yellow pages server. It refers search agents to the image servers where index agents reside (see Fig. 2.4). Once the client selects an image for retrieval, the process continues as in the gatherer model.

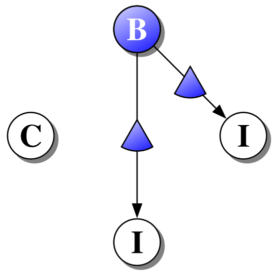


FIG. 2.3. *Incubator model (index phase): The broker (B) dispatches feature extraction agents (denoted as triangles) to the image sources (I). Once the agents completed extracting the features of all images, they register a feature comparison service at the image sources.*

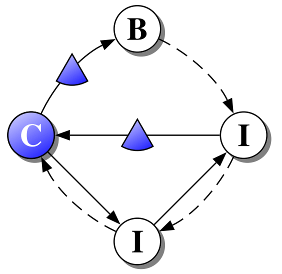


FIG. 2.4. *Incubator model (search/fetch phase): The client (C) sends a search agent to the broker (B), which retrieves a list of image sources, and searches them in turn (dashed lines). The client selects images for retrieval, which are subsequently collected from the image sources (I) by the fetch agent.*

Based on its index the index agent serves queries of search agents which visit the image server. On each image server, the search agent merges previously collected results with the results of its local search, and prunes the overall number of results *e.g.*, to a user-defined maximum number of n images with the lowest overall distance metric (unless security considerations take precedence, see also §3.2). The distance metric must be normalized so that the pruning is accurate. Multiple search agents can be dispatched in parallel to speed up the search. In such a case, the individual search results must be merged and pruned at a suitable host *e.g.* a server provided by the broker or the client itself. Moreover a broker may incubate an image server with different index agents, each one resembling a different matching algorithm.

Additionally, brokers may launch search agents on behalf of a client *e.g.*, if clients do not use mobile agents directly but rather access the broker through a regular Web interface. In this model the optimization is independent from the overall size of the content and the compression factor of the feature extraction algorithm. Only the matching images are transferred over the network.

2.3. The Extended Incubator Model. In the extended incubator model, the index phase is clearly separated from the search resp. fetch phase by means of the interaction paradigm used between the involved entities. As in the basic incubator model described above, the image broker dispatches one index agent per image server which takes residence at the image server. Again, the index agent extracts features and sets up a local index directly at the image server which is incementally updated, if the local image repository changes. In contrast to the basic incubator model, the search service provided by the index agent is automatically wrapped by a corresponding Web service stub. Furthermore, the corresponding Web service description (WSDL) is provided and the Web service is registered at a Web service compliant directory service hosted by the image broker (see Fig. 2.5).

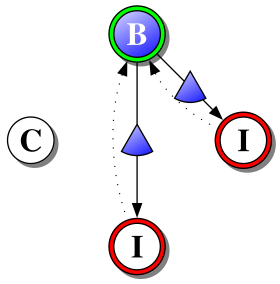


FIG. 2.5. *Extended Incubator model (index phase): The broker (B) dispatches feature extraction agents (denoted as triangles) to the image sources (I). Once the agents completed extracting the features of all images, they register a feature comparison service at the image sources which is automatically provided as Web service (visualized by red rings) and registered (dotted line) at the broker's Web service compliant directory service (visualized by green ring).*

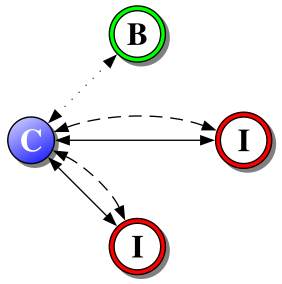


FIG. 2.6. *Extended Incubator model (search/fetch phase): First, the client (C) queries (dotted line) the Web service compliant directory service of the broker (B) which returns the locations of available comparison and fetch Web service instances. Subsequently, the client searches for images requesting (solid line) the search Web services at the image providers (I), and finally downloads the desired ones using the corresponding fetch Web services (dashed line).*

As in the basic model, the image broker is a central point of access which resembles a kind of yellow pages server. But this yellow pages server is realized as a Web service directory instead of a local service triggered by incoming search agents. Thus, the client interacts with the broker by means of the Web service discovery protocol UDDI within the search phase. Subsequently, it directly invokes the search Web service by means of SOAP requests, transmitting the image as search pattern and receiving the image IDs of comparable images—according to the selected comparison algorithm provided by the image broker—together with their thumbnails (see Fig. 2.6). Finally, the client requests the selected images from the corresponding image servers, again using the appropriate (fetch) Web services provided by these servers.

Instead of transmitting a reference image as search pattern to each image provider during the search phase, the broker could provide an appropriate feature extraction Web service which transforms received images into smaller feature vectors. Subsequently, the client would directly use this feature vector as search pattern for search Web service requests to lower the needed network bandwidth.

2.4. Comparison of Concepts and Benefits. The advantage of the gatherer and the incubator model over traditional centralized image search engines is that processor and memory consumption is shared between the image providers whose contents are processed, and the broker. Network utilization is considerably lower the mobile agent based models than in the traditional approach, and the feature extraction process completes considerably faster as a consequence of parallelization. None of the models export image contents to third parties.

The gatherer model is still somewhat centralized. Queries are answered by the broker. If the number of images is huge then the feature collection is likewise huge. Hence, while the gatherer model improves the process of index compilation it does not significantly improve the query process. Finally, if the broker fails then the entire service becomes unavailable.

Searching is less efficient in both incubator models than it is in the gatherer model because all image servers must be visited resp. queried by the search agent resp. the Web service compliant search client in turn before the query results are shown to the user. Certainly this can be alleviated by sending multiple search agents resp. triggering multiple search queries in parallel. However, the yellow pages maintained by the broker are much smaller than a full index of feature vectors, and they change less often then an index.[2] Therefore, replication (*e.g.*, by caching or by fail-over servers) can be implemented easier and more efficiently than this would be possible in the gatherer model.

The incubator models are particularly useful if image providers (who offer a broad range of images) team up with image brokers (who distribute specialized retrieval algorithms). Hence, the image broker may act as a well-known *portal site* with a focused marketing that addresses a specific target audience. The relationship between providers and brokers can be many-to-many, and their business relationships can be fluent and flexible. The advantage is that both parties can concentrate on their core competencies. Image provider specializes on content provisioning, and the image broker specializes on retrieval technology and image matching algorithms. In this regard our approach differs from *e.g.* meta search engines, which combine the results of regular search engines that in turn download contents. In our approach, however, search functionality is pushed to the content.

All the models (the gatherer model and the incubator models) can support multiple content-based retrieval mechanisms in parallel as well as retrieval mechanisms based on annotated information such as the name of photographer or painter, the year of production, or the price of licensing the image for specific uses. For instance, image servers can accept more than one index agent at the same time, and search agents resp. the Web service client can compute the intersection of multiple distinct result sets based on the global or local ID of each image entry.

In the extended incubator model, the used interaction paradigms are clearly separated according to the respective processing phases of the search engine. Thus, this model is specifically avantageous to support thin or legacy search clients which are not able to run the mobile agent middleware resp. to connect such a middleware with the actual client application: During the index phase, the image brokers still make use of the mobile agent paradigm and the resulting benefits—since image brokers and image providers are tightly bound to each other (*e.g.*, according to a business case), it can be assumed that they share the same mobile agent middleware. Providing search and fetch services as Web services by the image providers and a corresponding Web service directory by the image broker, the client application can easily be connected to the distributed search engine using one of the numerous Web service compliant frameworks available for arbitrary devices, operating systems, and corresponding programming languages.

Nevertheless, this model has some disadvantages during the search and fetch phase, which are specific to the client/server architecture as implemented by the described Web service infrastructure resp. specific to the Web service protocol suite. Depending on the underlying transport protocol (*e.g.*, HTTP), Web service requests are processed synchronously where required. Hence, the connection between client and image providers has to remain established while the search algorithm compares given images resp. feature vectors with the local index on the image server. In contrast, the mobile

[2]Image providers regularly add content but if a server is added or removed from the system then content is added or removed as well.

agent paradigm is much more flexible, *e.g.*, in scenarios with mobile devices. Furthermore, using the XML-based Web service protocols blows up the transported data and decreases the overall performance again, because of the necessary XML-based data type encoding/decoding.

As consequence, the user should carefully select or combine the appropriate search engine models according to the specific application scenario, considering the respective advantages and disadvantages.

3. Basic Architecture. The gatherer model and both incubator models are built on top of the SeMoA mobile code middleware [41]. Agents are received by network transport daemons, and they are injected into a pipeline of filters (see Fig. 3.1) which perform various security services and security checks on incoming agents (see §3.2 for more details). If an agent is admitted to the server then the agent may publish or retrieve service interface objects subject to access control restrictions. Upon termination, the agent is again processed by a filter pipeline and is subsequently migrated to its next hop.

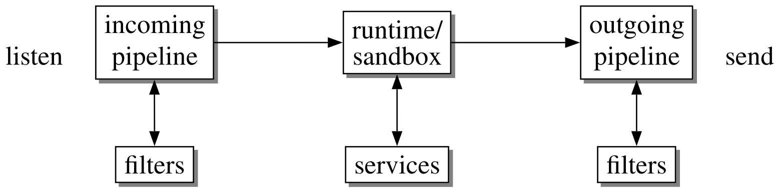


FIG. 3.1. *The middleware runs a daemon which listens for incoming agents. Each agent is piped through several filters before it is admitted to the runtime system where it can access services by name. Agents can register and retrieve services by name (subject to access control). Before migration, each agent is piped through outgoing filters again.*

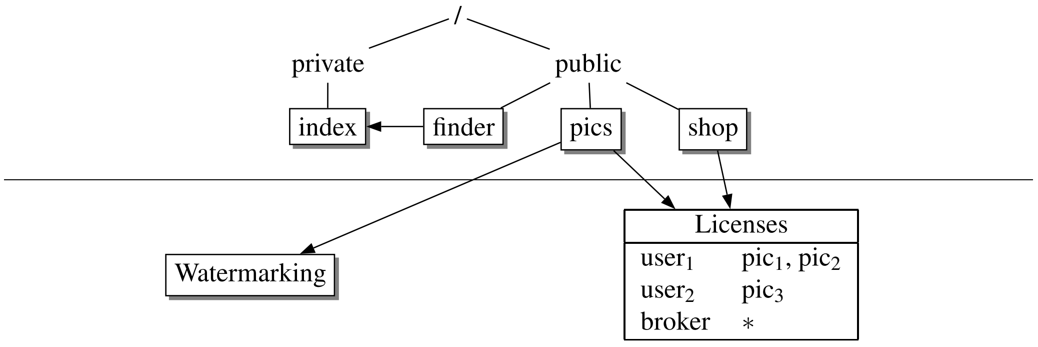


FIG. 3.2. *Agents can publish and retrieve services in a hierarchical name space. For instance, an image provider publishes the* pics *service under the path "/public/pics". The* pics *service iterates image names and thumbnails without restriction, but retrieves full quality images only if it is invoked by an agent whose owner has purchased a license. Agents can negotiate and purchase licenses on behalf of their owners by the* shop *service.*

Services are published in a hierarchical name space (similar to a hierarchical file system), which simplifies the grouping of services and the definition of access control policies. Agents may publish services dynamically at runtime in an allowed subspace of the name space, and the server may publish services or launch daemons statically at boot time. For instance, an image broker provides a static *index* service that his agents (and only his agents) can access in order to merge collected feature vectors with previously collected ones. In our implementation, the *index* service is backed by a file system and provides concurrent read/write access to the stored information. The image broker also publishes a static *finder* service which, on input of a query, returns matching image entries. This service is backed by the *index* service (as illustrated by the horizontal arrow in Fig. 3.2) but restricts access to the index to a limited set of operations and can therefore be made accessible to search agents (by placing it in the public area of the name space). In the incubator model, an index agent publishes the *finder* service dynamically at the image provider, and it keeps a private (unpublished) *index* service as the back end of its *finder* service. In that case, the image entries are stored by which ever resource backs the storage of the index agent[3]. Figure 3.3 gives a simplified view of the interface and class design in the UML [17] notation.

Image providers publish the static *pics* and *shop* services. The *pics* service iterates image IDs (*e.g.*, a locally unique image name) and thumbnails without restriction, but retrieves full quality images (based on the image ID) only if the

[3]Typically a file system or RAM of the host computer, the middleware provides abstractions for the actual type of agent storage which could also be backed by a database.

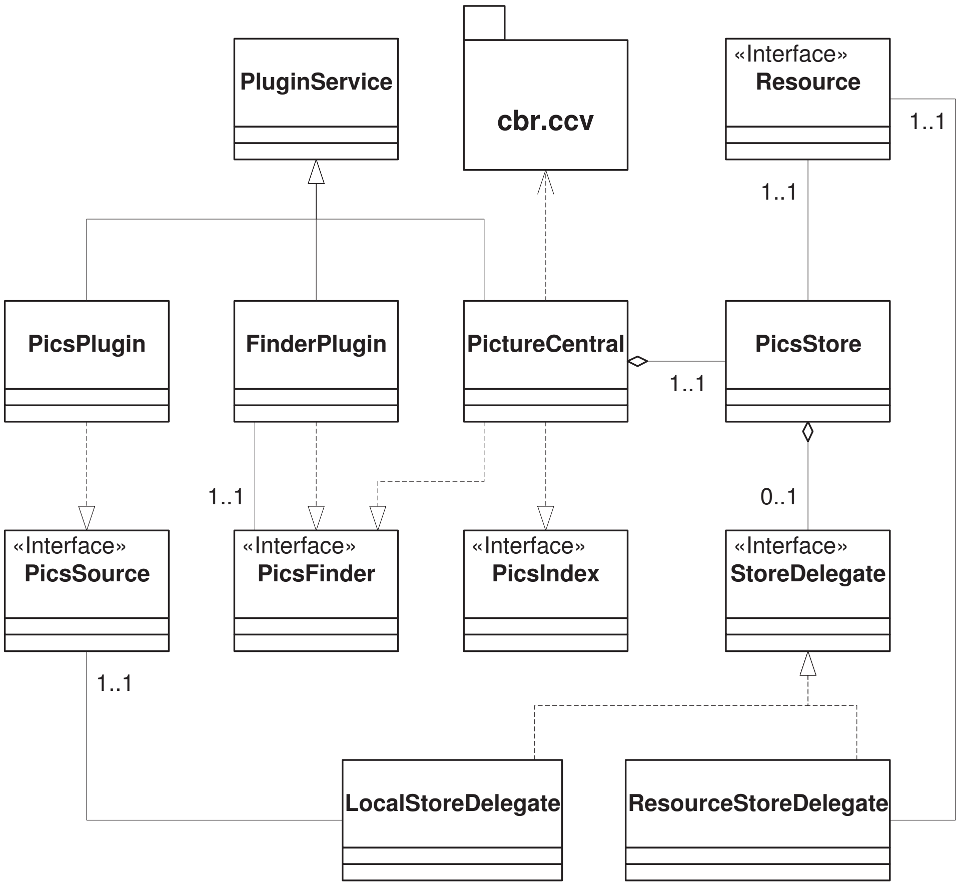


FIG. 3.3. *A simplified view of the interface and class design related to the CBR services is given above. Storage of information is handled through the "Store" abstraction for which we implementated instances that map to RAM or file systems.*

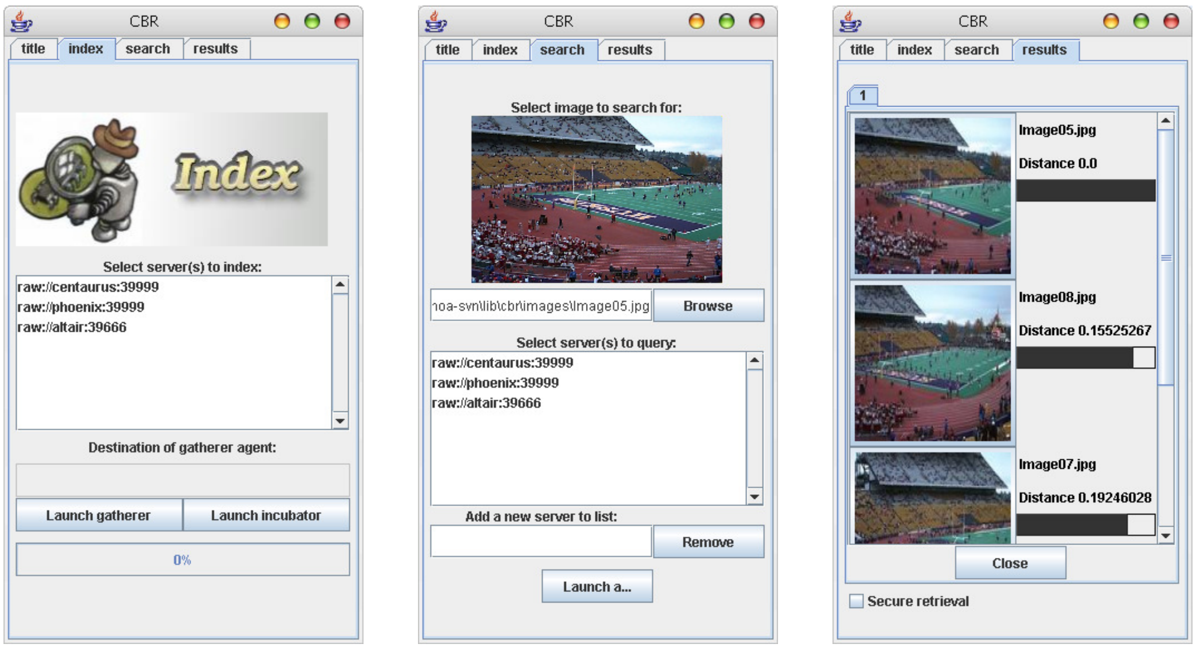
owner of the invoking agent already purchased a license for the image in question. In this case, the *pics* service may also embed the client ID as a digital watermark[4] into the retrieved image so that unauthorized usage of copyrighted material can be traced. Clients (resp. their agents) can negotiate and purchase licenses by the *shop* service[5]. Brokers can purchase a license for all images for the purpose of indexing (presumably at a low price and under the legally important condition that no full quality images are illicitly exported). Alternatively, image providers can grant brokers access to their images based on prior offline agreement.

The *pics* service provides a simple and sufficient interface so that feature extraction algorithms can iterate and extract features from existing images, irrespective of the heterogeneity of deployed image databases (*e.g.*, the schema of the database or the fact that images are simply stored in a file system).

3.1. Implementation Details. The prototype implementation uses *Color Coherence Vectors* [36] as feature extraction and comparison algorithm. Feature vectors consist of 128 float values; each vector is computed as follows: the image is blurred using a simple 3×3 convolution filter which averages the color values of all horizontal and vertical neighbors of the filtered pixel. The blurred image is then quantized to a color space of 64 colors. In the last step, the pixels of the image are classified into coherent and incoherent pixels. Coherent pixels are pixels which are part of a horizontally and vertically connected pixel area of the same color whose size exceeds a certain threshold τ (a fixed percentage of the total image area). Incoherent pixels are pixels which are not coherent pixels. For each of the 64 colors, the coherent and incoherent pixel counts are summed up separately and normalized with regard to the total image area. This results in

[4]The term "digital watermarking" refers to steganographic means of embedding copyright markers in multimedia data so that the marking is imperceptible, undetachable, as well as robust against a variety of adverse and inadvertent manipulations of the media such as lossy compression, format conversion, *et cetera*. See *e.g.*, [1] for an overview over digital watermarking.

[5]Together with Siemens Corporation we engineered and evaluated a licensing system for images retrieved by mobile agents. The system founds on Kerberos tickets and was used with both retrieving mechanisms described above [16].


CBR
title
index
search
results
Index
Select server(s) to index:
Destination of gatherer agent:
Launch gatherer
Launch incubator
0%
Select image to search for:
Browse
Select server(s) to query:
Add a new server to list:
Remove
Launch a...
1
Close
Secure retrieval

raw://centaurus:39999
raw://phoenix:39999
raw://altair:39666

noa-svn\lib\cbr\images\Image05.jpg

Image05.jpg
Distance 0.0
Image08.jpg
Distance 0.15525267
Image07.jpg
Distance 0.19246028

FIG. 3.4. *Three shots of the prototype GUI. The panel used to launch index agents is left, in the middle is the panel used to start search agents for a given example query image, and the panel which shows the query results after the search agent returned is right.*

a 128 dimensional vector. The L_1 distance is taken as a measure of similarity between two color coherence vectors. Let $\langle(h_1,\bar{h}_1),\ldots,(h_n,\bar{h}_n)\rangle$ be a color coherence vector where h_i is the percentage of coherent pixels of color i and $\bar{h}_i$ is the percentage of incoherent pixels of color i then the L_1 distance is defined as: $\|h-h'\| = \sum_{i=1}^{n}(\mid h_i - h'_i \mid + \mid \bar{h}_i - \bar{h}'_i \mid)$. The Color Coherence Vector algorithm has the advantage that it is easy to implement, reasonably fast, and achieves a high compression rate.

Images are reduced to a vector whose encoding is less than 600 bytes. For feature extraction algorithms whose output has a length comparable to the size of the images no volume transfer savings are achieved. Although in this case the processor utilization is still distributed among the image servers (this results in a speedup linear in the number of image servers, given a uniform distribution of images). Here, we assume that *number of search engines* $\ll$ *number of image servers*.

The graphical user interface of the demonstrator is shown in Fig. 3.4. The left view shows the panel which is used to launch index agents. From a list of known servers, a subset can be chosen. On pressing the button titled *start indexing*, index agents are created and dispatched to each selected server. On the target server, each index agent looks up the *pics* service which must be registered in the target server, and starts the feature extraction process. Upon completion, it publishes an instance of the *finder* service in the target server, which search agents can look up and query.

The middle view shows the panel which is used to dispatch search agents. Again, a number of servers can be chosen from a given list. The search agent takes a user-provided example image (which can as well be a sketch or prototypical image), hops in turn to all selected image servers, and collects image entries with a distance less than a given threshold and up to a given maximum number. The overall best matches (thumbnails but not full images) are reported back and presented in the results panel.

The results panel shows the retrieved thumbnails and each entry's distance to the query image. If the user clicks on a thumbnail then a fetch agent is created and dispatched to the host where the image was retrieved, and returns the corresponding full image. The check box in the lower left corner of the results panel enables secure retrieval. If it is checked then retrieved images are transported in encrypted form as explained in [40].

Although we implemented only one feature extracting and matching method so far, the interfaces are designed to support multiple and alternative implementations transparently. All implementations have been developed in the Java programming language (Java Version 2)[6].

[6]The middleware SeMoA and the implementation of the agent based search engine are distributed as open source software at `www.semoa.org`

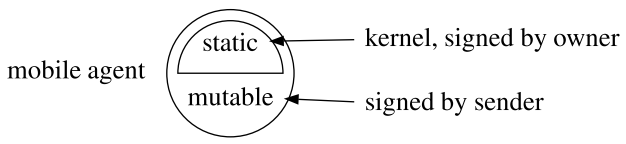


FIG. 3.5. *A mobile agent consists of static data (which does not change during the agent's lifetime* e.g.*, code and a random agent ID) and mutable state. The owner assumes responsibility for her agent by signing its static part (the* kernel*), and the most recent host of an agent signs the entire agent to assume responsibility for the agent's most recent state.*

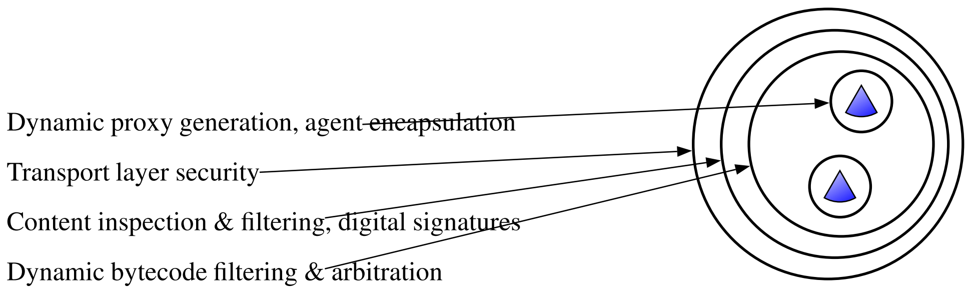

Dynamic proxy generation, agent encapsulation
Transport layer security
Content inspection & filtering, digital signatures
Dynamic bytecode filtering & arbitration

FIG. 3.6. *The security architecture resembles an onion. Agents have to pass all layers successfully to be admitted to the system. The outer layers keep threats out of the system. The innermost layer encapsulates and confines agents.*

3.2. Content Protection. It has been argued that mobile agents achieve a greater level of control over the media being searched on [6]. This is only part of the truth, though. In practice, various covert channels [26] as well as direct means of cheating can be used *e.g.*, by malicious index agents and colluding search agents to subvert image export restrictions. The billing schemes proposed by Belmon and Yee (which account for projected losses due to covert channels) punish thieves and ordinary clients likewise and will hardly be accepted. Still, using *e.g.*, the incubator model can improve confidence that image contents are not exported illicitly from image servers. Evidence of stealing images on the part of index agents can be established by reverse engineering the agents' code which is present at the image server, if it comes to the worst. SeMoA requires that each sender of an agent digitally signs the static parts of his agent (including the code), which establishes a non-repudiable proof of ownership. This signature yields a unique and unforgeable agent *kernel*. Furthermore, each server must sign the entire agent before transport. This signature binds the new state of the agent to its kernel and protects the agent against tampering during transport. Thereby each server documents its responsibility for any state changes that the agent may have undergone while being hosted by it (see Fig. 3.5 for an illustration of signatures).

Nevertheless, an agent must also have means of protecting itself against a malicious host as soon as for-profit services are involved. If search agents pass by multiple competing image providers there is a certain chance that a provider does not play by the rules. One possible attack that a dishonest provider may launch on an agent is to manipulate the accumulated search results or to replace or degrade the quality of images retrieved from a competitor. Thereby the perpetrator increases the likelihood that images are purchased from him the next time. Other attacks involve tampering with the agent's code in order to alter its negotiation strategy, decision making, or simply to cause harm at the servers of competitors.

SeMoA prevents tampering with the code by requiring that code must be digitally signed. Any such tampering invalidates the kernel of the agent and hence the agent subsequently fails to "speak on behalf of its original owner." Tampering with accumulated images is prevented by establishing encrypted subsections in the agent so that each subsection can be accessed only by the agent's owner and by hosts that he authorizes [40]. For instance, the fetch agent is programmed so that it stores each retrieved image in the section that is dedicated to its current host. This section is automatically encrypted by the *encrypt* filter when the agent departs.

This setup has a drawback, though. For security reasons agents do not carry private keys for the encrypted sections. Hence they cannot access Agents cannot access results that they collected prior to reaching the current host (these results are encrypted and by assumption the current host does not have a matching private key). This prevents an agent from pruning its accumulated results *e.g.*, to a fixed upper number of overall best results (we referred to this problem in §2.2). One workaround is to program agents so that they regularly migrate to a trusted and authorized host where the agent can access and prune all results that have been accumulated up to this point.

3.3. Server Safety and Security. For practical purposes it is essential that agent servers are protected against attacks by malicious agents. Agents must not be able to disrupt or otherwise negatively effect the operation of an image server or other agents hosted by it. For mobile agent servers based on Java this is currently an elusive goal—the Java runtime system is vulnerable to a variety of *denial of service* attacks [7].

However, SeMoA makes a best effort to protect the runtime system against malicious code, and provides pluggable bytecode filtering and arbitration modules. Before a class is loaded into the name space of an agent each module may inspect, reject, or instrument the bytecode of that class. Currently, SeMoA includes a module that rejects classes which *e.g.,* override the `finalize` method, a well-known and simple way to attack the garbage collector thread of the virtual machine (a more subtle variant of this attack may be directed at the `close` method of some I/O classes). The same extension mechanism can be used to add resource control by bytecode arbitration [23] as well as additional security checks.

Each agent is run in a separate name space with a separate class loader and in a separate thread group. Thereby, interference of agents is prevented. A special security manager filters and sorts newly created threads so that for instance threads of the *Abstract Window Toolkit* are not accidentally placed in the thread group of an agent. Marshalling and unmarshalling is done by the initial agent thread from within the agent's sandbox so that an agent cannot exploit callbacks in the *Java Serialization Framework* to hijack server threads. The server transports an agent only after all threads of that agent have terminated, which prevents the adverse or inadvertent creation of clones or zombies (or, for that matter, widespread infection of servers by a worm).

Once running, an agent may publish and retrieve service objects (such as the *finder* service) by name in the server's object registry if the agent has appropriate permissions. Published objects are automatically wrapped in a proxy object which is created dynamically. The proxy prevents uncontrolled aliasing of the service object and automatically invalidates references to it as soon as the agent terminates. This makes the service object available for garbage collection. Agents cannot share classes (by virtue of separate name spaces the classes would not be type-compatible) but they may share interfaces for the purpose of method invocation and communication. However, two interfaces are shared across name spaces only if their implementations are mapped to the same value by a cryptographic hash function. In such a case, a super-ordinate class loader assures type-compatibility.

Before an agent is loaded and run, it must pass a configurable pipeline of pluggable filter modules (see also Fig. 3.1). Each filter may reject the agent in the case of errors. SeMoA provides a variety of security related filters some of which transparently handle digital signatures, certificate chain validation, and encryption of subsections of an agent. Agent transport is possible both in the clear and over a mutually authenticated *Secure Sockets Layer* (SSL) connection. A schematic illustration of SeMoA's security architecture is given in Fig. 3.6.

Moreover SeMoA allows the execution of agents from different agents system by means of an interoperability layer [42], such as JADE agents. The security measures are designed in a way that they apply fully to any kind of dynamically loaded code executed within the SeMoA middleware.

4. Integration of Mobile Agents and Web Services. According to the Web service architecture as defined by W3C [51], a *Web service* is an abstract notation which is implemented by a concrete *agent* as computational resource, owned by and acting on behalf of a person or organization. An agent realizes one or more services and may request other services, in turn. The Web service specification ensures the interoperability between systems through machine-to-machine interaction over a network, whereas it avoids any attempt to govern the implementation of agents.

On the one hand, this agent concept is defined on a high level of abstraction neglecting detailed characteristics of the numerous existing software agents systems. Furthermore, most current applications base either on Web service or on software agent technology, but not on both simultaneously. On the other hand, the Web service specification already indicates the integration of both technologies resp. hints at similarities and possible extensions/improvements of one technology through the other.

Similar to Web services, software agents can encapsulate business or application logic. Rather, software agents can dynamically discover, combine and execute such processes, and further offer multiple services or behaviors that can be processed concurrently. In order to move from one system to another or even to communicate with each other, mobile agents currently need a common platform on which they operate. Thus, they are useful for business partners only if these actually share a common platform. The consistent use of Web service standards for description of capabilities, communication, and agent discovery would establish interoperability not only between different agent platforms but also between agent platforms and traditional Web services, combining the advantages of two worlds.

To dynamically deploy both technologies without much overhead for the application developer mobile agents and Web services have to be integrated in a seamless manner, whereas mapping of processes have to be fully automated.

4.1. Requirements and Definitions. To integrate mobile agent and Web service technology in a seamless manner, components have to be designed, which map between the different mechanisms for *service description*, *service invocation*, and *service discovery*, in both worlds. In other words, messages resp. representations from the according Web service protocols (WSDL, SOAP, UDDI) have to be translated into corresponding requests resp. data types of the agent system, and vice versa.

When we talk of a *Web service engine* we mean the summary of these components, which can be integrated into an existing agent system, and thereby extend this system with Web service abilities. Especially in some communication-centered agent systems, this engine further has to bridge the gap between the synchronous behavior of SOAP, and the asynchronous messaging paradigm as it is specified by FIPA with FIPA-ACL, for example (cf. [15]). Describing the level of integration, we want to define the following features for the integration of Web services and agents:

self-contained The Web service engine integrates all components for a seamless transition from one technology to the other, without the need of additional external resources.

bidirectional The Web service engine supports the invocation of Web services by agents as well as the provisioning of agent services by means of Web services. Thus, it allows cross boundary interaction in both directions.

automated After being properly configured and started, the Web service engine does not require any further manual steps executed by the user during runtime.

transparent The components of the Web service engine are transparently integrated into the agent system resp. the Web service infrastructure. Neither agents nor Web services as the acting entities recognize the existence of the Web service engine.

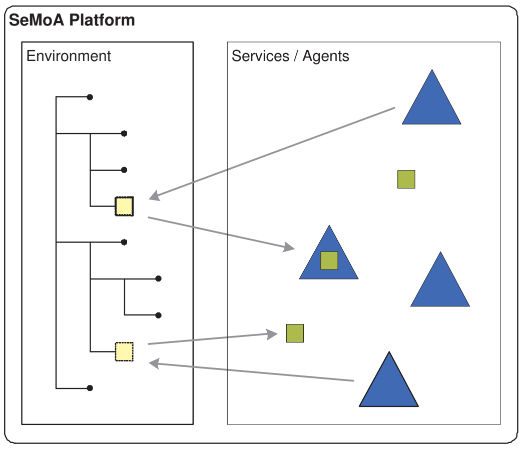


FIG. 4.1. *Hierarchical service environment.*

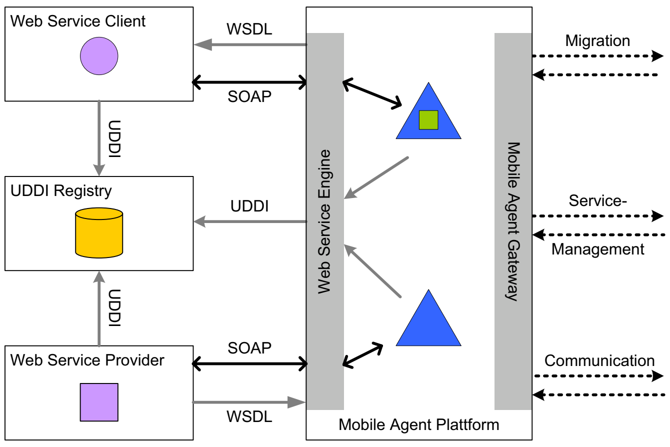


FIG. 4.2. *WSE Integration Architecture.*

4.2. System Integration. As already mentioned in §3.3, the only mean of interaction between mobile agents and/or services on the local server is based on direct method invocation through shared Java interfaces (cf. Fig. 4.1). Services are registered locally as entities within a hierarchical namespace (the service *environment*). A service requester searches for an appropriate service interface within a defined sub hierarchy of the environment. If successful, a Java object is returned which implements the given interface.

To integrate Web service technology in SeMoA, the local service management has been extended by a Web service engine (WSE) fulfilling all the requirements defined above. Presupposing that either a certain *Web service flag* has been set when publishing a service, or a service has been published in a certain *Web service enabled region* of the environment, this service is automatically wrapped by a Web service stub which is accessible by remote clients via SOAP requests. Further a corresponding Web service description is generated and registered on a remote UDDI-compliant directory server. The WSDL-conform service description allows both, a broad search for services according to a given category, and a concrete search for services implementing a given interface and/or having a specific name in a hierarchical namespace which corresponds to SeMoA's local environment. Thereby, a service interface is unequivocally identified by its name and the cryptographic hash code of its normalized method signature. When a service is requested from a Web service enabled region of the environment which does not exist locally, the WSE tries to find a corresponding Web service and returns a transparently generated Web service client stub, implementing the desired interface(s). Figures 4.3 and 4.4 show the corresponding extended processes for (Web) service deployment and undeployment resp. (Web) service discovery.

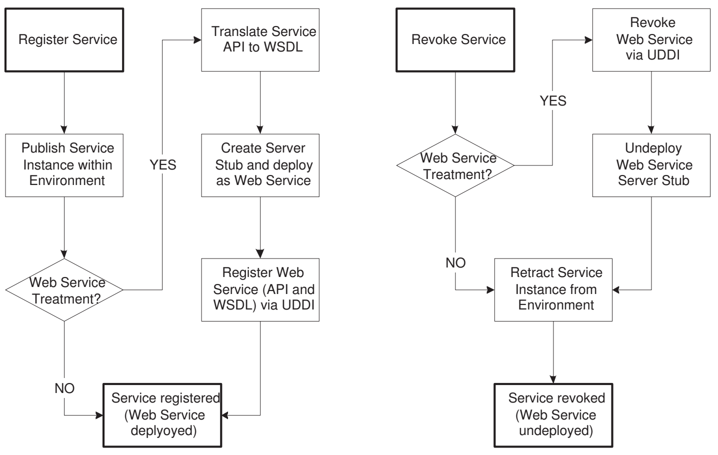

Register Service
Publish Service Instance within Environment
Web Service Treatment?
YES
NO
Translate Service API to WSDL
Create Server Stub and deploy as Web Service
Register Web Service (API and WSDL) via UDDI
Service registered (Web Service deplyoyed)
Revoke Service
Web Service Treatment?
YES
NO
Revoke Web Service via UDDI
Undeploy Web Service Server Stub
Retract Service Instance from Environment
Service revoked (Web Service undeployed)

FIG. 4.3. *Deployment/Undeployment of a (Web) service.*

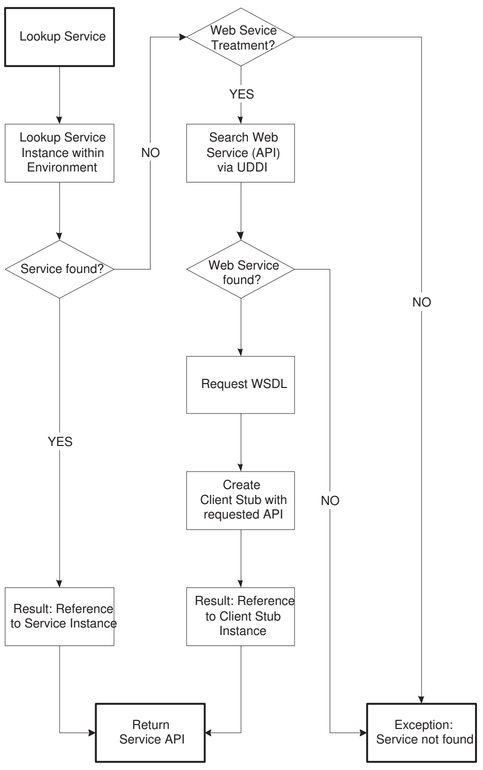

Lookup Service
Lookup Service Instance within Environment
Service found?
YES
NO
Web Sevice Treatment?
YES
NO
Search Web Service (API) via UDDI
Web Service found?
NO
Request WSDL
Create Client Stub with requested API
Result: Reference to Service Instance
Result: Reference to Client Stub Instance
Return Service API
Exception: Service not found

FIG. 4.4. *(Web) service discovery.*

Thus, in addition to the functionality provided by an existent mobile agent gateway, agents (△) are enabled to request and interact with external Web services (□), and external Web service clients (○) are enabled to request and invoke services encapsulated by agents (see Fig. 4.2). Further details about the internal WSE modules can be found in [37, 38].

4.3. Secutity Aspects. The current use of Web services to share information and services across organisations make necessary a new mean of access control. Besides, the use of Web service protocols imply a couple of Denial-of-Service attacks against XML parsers resp. new SOAP-specific attacks, *e.g.*, with respect to command injection or session hijacking.

Figure 4.5 gives an overview of existing Web service security mechanisms within the Web service protocol stack: *Transport Layer Security (SSL/TLS)* [19] offers a secure channel (point-to-point) between Web service client and Web service provider. *XML-Encryption* [22] and *XML-Signature* [4] provide basic security mechanisms for XML messages. These specifications are used within the *WS-Security* protocol family [3] to create appropriate SOAP security headers. Furthermore, high-level protocols as there are the *Security Assertion Meta Language (SAML)* [33] to exchange authentication and authorization information, or the XML *Access Control Markup Language (XACML)* [34] to define role-based access control policies allow specialized security functionalities. Currently, even a bunch of new high-level protocols are specified and standardized, such as *WS-Policy*, *WS-Trust*, *WS-Privacy*, *WS-Authorization*, *WS-SecureConversation*, etc. (cf. [21]).

At the latest when combining basic Web services to composite Web services with added-value, the simple interaction paradigm between an explcit client and one service provider as server becomes obsolete. Rather, a dynamic tree models temporary bidirectional requestor/provider associations as part of a dependency hierarchy, established by a client at the root of the tree (compare [35]). Since a requestor then accesses a remote resource through a provider's Web service on behalf of another entity (the client), authentication and authorization mechanisms have to be established, which support this kind of delegation principle, and concurrently minimize the overhead for the requesting client. In this context, [28] compares current authentication and authorization infrastructures, while [12] discusses and proposes a new framework for Web service access control based on the above presented standards which decouples authentication and authorization, and further takes dynamic aspects (as there is the request context by means of an access history) into account.

In our opinion, appropriate standard-based security mechanisms for Web services have to be selected and combined according to specified security requirements. Especially in the context of integrating mobile agents and Web services, the

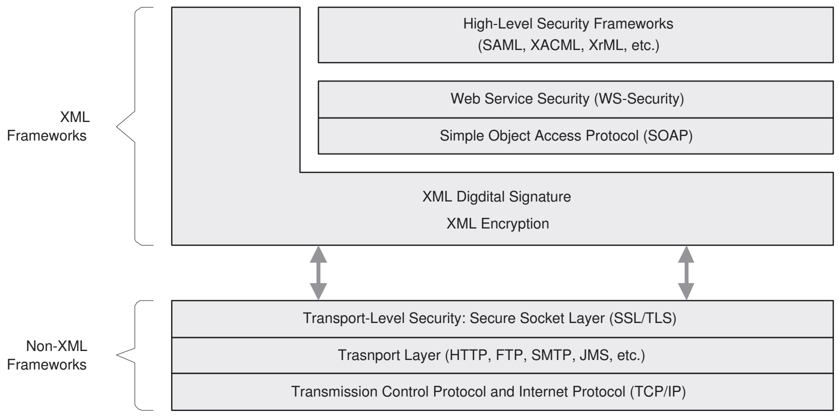

XML Frameworks
High-Level Security Frameworks (SAML, XACML, XrML, etc.)
Web Service Security (WS-Security)
Simple Object Access Protocol (SOAP)
XML Digdital Signature
XML Encryption
Non-XML Frameworks
Transport-Level Security: Secure Socket Layer (SSL/TLS)
Trasnport Layer (HTTP, FTP, SMTP, JMS, etc.)
Transmission Control Protocol and Internet Protocol (TCP/IP)

FIG. 4.5. *Basic Web service protocol stack including security layers [10].*

existing security frameworks for both technologies have to be thoroughly combined without enabling new side-channel attacks to the integrated system. Currently, we are combining transport layer security with enabled client authentication in point-to-point connections with SAML-based authentication resp. authorization tokens as part of the transmitted messages transport information about the security context of the acting entity from one host to another. Thereby, the cumulated security information is used to enforce the local security policy on the target platform.

5. Evaluation. In the incubator models, no feature vectors must ever be transported over the network. Therefore, a quantitative evaluation of this model is superfluous. We evaluated the gatherer model of our prototype for small sets of images in order to get a general idea of the potential savings that can be achieved by mobile agents compared to conventional client/server approaches. The setup of the experiments favored the conventional approach so that our results remain conservative. In practice, we expect that the relative performance of mobile agents is better. We used the hardware and software given below in our evaluation:

- 1 $\times$ Pentium III mobile, 1.2 GHz, 512 MB, FreeBSD 4.7, running Java Version 1.4.1 under the Linux emulation.
- 9 $\times$ Sun Ultra 5/10, 500 Mhz (UltraSPARC-IIe), SunOS 5.8/5.9, 256MB-512MB, running Java Version 1.4.1, *Apache* Server Version 1.3.
- Switched Fast Ethernet (100 Mbit/s)

The nine computers we used were connected by our institute's LAN, which consists of several hundred workstations and PCs, and which is accessed by more than 150 research assistants and a large number of students (although we did our tests a weekend to reduce distortion of measurements due to regular use of the network).

We first measured the performance of the conventional approach. A simple client program loaded and extracted the features of all images, with a varying number of image sources. The client was programmed in the Java programming language; it ran on the 1.2 GHz Pentium III laptop, it used three threads per image source in parallel to optimize I/O utilization (we found by experimenting that this gave the best results), and it was based on the same code that was used by the mobile agents in subsequent testing.

The image sources consisted of 8 Apache servers, each of which ran an Apache server with 48 images. All images were in JPEG format with a resolution of 756 $\times$ 504 pixels, and were loaded by the Apache server from the built-in hard drive. The size of images varied from approximately 280000 to 456000 bytes, depending on the JPEG compression rate. Each experiment was done three times to observe variances.

In the measurement of the mobile agent performance each Sun hosted a mobile agent server. The ninth Sun (without images) played the role of the broker. The other Suns were configured with a simple service that allows to iterate picture names and to retrieve image data for an image with a given name. Again, all images were loaded from the built-in hard drive.

In experiment one, we launched a mobile agent on the broker. This agent migrated to image server one, extracted the features of all images, transported the image entries back (excluding thumbnails), and merged the image entries with the broker's central index. In the second through eighth experiment, we launched two to eight agents in parallel which performed the same operations on the additional image servers. In each experiment, we measured the time from starting the first agent until the last agent returned and completed its task. Again, we repeated all experiments three times.

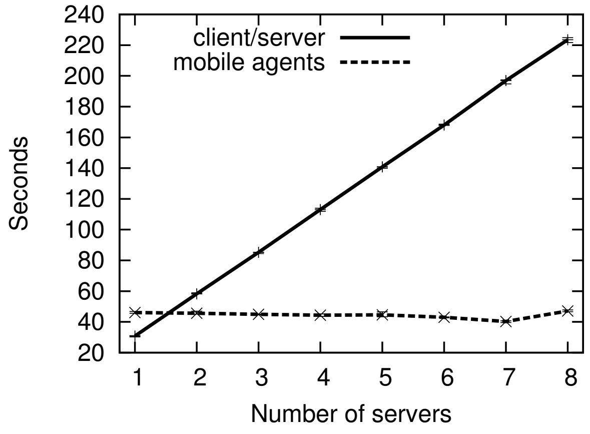


FIG. 5.1. *The time that the client/server based gather needs to complete the feature vector collection task vs. the time required by the mobile agent approach (measured from starting the first agent until completion of the last agent). The point of intersection is the break-even point at which the additional overhead of shipping and setting up mobile code is amortized by the reduced network utilization of the mobile code approach.*

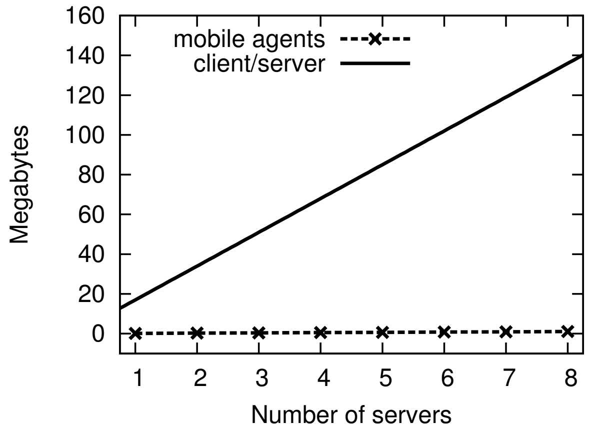


FIG. 5.2. *The amount of data transported by the network in the client/server based gatherer vs. the mobile agent approach. The client/server graph is plotted based on the size of all downloaded images times number of servers. The graph of the mobile agent approach has a slope which is too small to be noticeable compared to the upper graph.*

We also measured the sums of sizes of all image entries (including overheads for the agents) transported per experiment. Comparison of all collected results shows significant savings in favor of the mobile agent approach, as could be expected (see Fig. 5.1 and 5.2, min and max values deviate so little from the median and mean values of the client/server and mobile agent measurements that the error bars are hardly noticeable in the graphs).

Additionally, our image search engine was the subject of a field study with the objective to assess the legal aspects of electronic commerce with mobile agents. Over the period of two days, eleven lawyers used our retrieval system in the roles of the content provider, customer, and image broker, with the objective to trade images. Overall, more than a thousand agents were dispatched. Our prototype proved to be user-friendly, robust, and reliable.

6. Related Work. Considerable work is done in the general area of CBR, see *e.g.*, [27] for the proceedings of a recent conference. Well received work by several authors reports on CBR systems for the World Wide Web [45, 44, 5]. All these image search engines are based on the client/server paradigm of collecting images from the Web. Mobile agent technology is complementary to this work. It remains to be investigated how well the algorithms developed by the authors mentioned above can be adapted to be used within a mobile agent framework. The idea of using mobile agents for content based image retrieval has been mentioned before [39, 2, 54]. Mobile agents have also been applied in related applications. For instance, in [25], Johansen reports on the use of mobile agents in the context of a weather information system (mobile agents process and deduce weather information from satellite imagery). Early attempts to integrate static agents and Web services have been described in [31, 55, 8]. More sophisticated solutions, especially in the context of FIPA-compliant agent platforms, have been presented in [29, 14, 46, 20]. In contrast, the existing contributions based on mobile agents have more diverse goals, and thereby do not completely follow transparent integration of agents and Web services [9, 30, 35, 13, 24].

It is not our intention to claim originality of the idea, but to report unique aspects of our architecture (mobile agent based incubator and the gatherer model; combination with Web services), the results of our evaluation, and our experiences with respect to the usefulness of the application.

7. Summary. Digital images are a valuable commodity and we expect that more and more photo agencies make use of the Internet as the principal platform for advertising, customer relationship management, and—most importantly—content distribution. We presented two models of deploying mobile agents to gather image information from the Internet and a third extended model using Web services. All models take into account that content providers must retain control over their intellectual property. Multiple complimentary retrieval methods can operate in parallel. The models support flexible software distribution, updates, and deinstallation, and they can be extended to account for negotiation of license terms and automatic fingerprinting of retrieved images based on digital watermarks.

The models differ in the grade of decentralization. In the gatherer model, the amount of data transported over the network depends on the size of the images and the compression factor of the deployed feature extraction algorithms. In our case, this is less than 1% of the image data transported by regular gatherers. In the incubator model, no image data is shipped over the network at all. Independently of the size of feature vectors, all models achieve a constant speedup, which is proportional to the number of image servers indexed in parallel. Image providers must set up and reserve computing resources for the mobile agent server. They can operate this server in conjunction with a Web server *e.g.,* by attaching the agent server to the Web server by Servlets. Furthermore, the extended incubator model allows to attach non mobile agent clients by means of an arbitrary Web service compliant middleware framework, which is adjusted to the special needs resp. abilities of the client device.

One avenue for improvement is the combination of the incubator and the gatherer model. The index agent that takes residence at the image provider may cluster the feature vectors *e.g.,* as described in [49]. Rather than sending only its *finished* message to the broker it may submit a number of centroids of the densest clusters. Search agents which visit the broker may thus opportunistically prune the search space by migrating only to servers with centroids most similar to the query vector.

Mobile agent infrastructures require a sound security model, which accounts for the various threats. Some progress has been achieved in the area of mobile agent security [47, 48], although a number of hard problems are still unsolved. Yet it is probably fair to say that in principle the attainable level of security is reasonable enough to justify the application of mobile agents in some real-world applications. However, before this can happen, runtime systems must become more robust *e.g.,* Java must become considerably more robust against *denial of service* attacks [7].

Using Web services in the extended incubator model opens a bunch of new Web service specific security risks and disadvantages. And although, there are already a bunch of different and specialized Web service compliant frameworks, some more work and research has to be done, before these Web service frameworks are completely interoperable with each other[7] and reach acceptable performance values comparable with other (proprietary) client/server protocols for distributed systems. Furthermore, the new Web service protocols still have to proof their applicability in the variety of Web service based application scenarios.

Acknowledgments. We thank Patric Kabus for his help in programming the basic search engine prototype. Further we thank Matthias Pressfreund and Dennis Bartussek for their help in programming the Web service extensions.

REFERENCES

[1] M. Arnold, M. Schmucker, and S. D. Wolthusen, *Techniques and Applications of Digital Watermarking and Content Protection*, The Artech House Computer Security Series, Artech House, Norwood, MA, USA, 2003.

[2] C. Arora, P. Nirankari, H. Ghosh, and S. Chaudhury, *Content based image retrieval using mobile agents*, in Third International Conference on Computational Intelligence and Multimedia Applications (ICCIMA '99), 1999, pp. 248–252.

[3] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon, *Specification: Web Service Security (WS-Security)*, Tech. Report Versoin 1.0, IBM developerWorks, April 2002. `ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf`

[4] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML-Signature Syntax and Processing*, W3C recommendation, World Wide Web Consortium (W3C), February 2002. `http://www.w3.org/TR/xmldsig-core/`

[5] M. Beigi, A. B. Benitez, and S.-F. Chang, *MetaSEEk: A content–based meta–search engine for images*, in Proc. Storage and Retrieval for Image and Video Databases VI (SPIE), San Jose, CA, USA, January 1998.

[6] S. G. Belmon and B. S. Yee, *Mobile agents and intellectual property protection*, in Rothermel and Hohl [43], pp. 172–182.

[7] W. Binder and V. Roth, *Secure mobile agent systems using Java – where are we heading?*, in Proc. 17th ACM Symposium on Applied Computing, Special Track on Agents, Interactions, Mobility, and Systems (SAC/AIMS), Madrid, Spain, March 2002, ACM.

[8] F. Büscher, *How Multi-Agent Systems and Web Services can work together*, in Net.ObjectDays 2004: NODe Young Researchers Workshop '04, Erfurt, Germany, September 2004.

[9] J. Cao, Y. Sun, Y. Wang, and S. Das, *Scalable Load Balacning on Distributed Web Servers Using Mobile Agents*, Journal on Parallel and Distributed Computing, 63 (2003), pp. 996–1005. ISSN:0743-7315.

[10] M. Chaniliau, *Web Services-Sicherheit und die SAML*, online article, XML and Web Services Magazin, January 2004. `http://www.entwickler.com/itr/online_artikel/psecom,id,468,nodeid,69.html`

[11] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL), Version 1.1*, W3C working group note, World Wide Web Consortium (W3C), March 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`

[12] M. Coetzee and J. Eloff, *Towards Web Service access control*, in Computers & Security, vol. 23, Elsevier, October 2004, pp. 559–570.

[13] D. Cooney and P. Roe, *Mobile Agents Make for Flexible Web Services*, in Proceedings of The Ninth Australian World Wide Web Conference, Quensland, Australia, July 2003. `http://ausweb.scu.edu.au/aw03/papers/cooney/`

[14] J. Dale, A. Hajnal, M. Kernland, and L. Z. Varga, *Integrating Web Services into Agentcities Recommendation*, tech. report, Agentcities Task Force, November 2003. `http://www.agentcities.org/rec/00006/actf-rec-00006a.pdf`

[15] FIPA, *ACL message structure specication*, Tech. Report FIPA document XC00061E, Foundation for Intelligent Physical Agents, August 2001. `http://www.fipa.org/specs/fipa00061/`

[7] Confer the ambitions of the Web Service Interoperability Organization (WS-I) at `www.ws-i.org`.

[16] K. Fischer and V. Lotz, *Authorization and Delegation of Privileges in Mobile Agent Systems*, Mobile Agents, (2002).

[17] M. Fowler and K. Scott, *UML Distilled*, Addison-Wesley, Reading, Massachusetts, U. S. A., 1997.

[18] S. Franklin and A. Graesser, *Is it an agent, or just a program?*, in Intelligent Agents III, vol. 1193 of Lecture Notes in Artificial Intelligence, Berlin, 1997, Springer Verlag, pp. 21–36.

[19] A. O. Freier, P. Karlton, and P. C. Kocher, *The SSL Protocol, Version 3.0*, internet draft, Netscape, November 1996. `http://wp.netscape.com/eng/ssl3/`

[20] D. Greenwood and M. Calisti, *Engineering Web Service—Agent Integration*, in IEEE International Conference on Systems, Man and Cybernetics (SMC 2004), The Hague, The Netherlands, October 2004.

[21] IBM and Microsoft, *Security in a Web Services World: A Proposed Architecture and Roadmap*, whitepaper, IBM Developer Works, April 2002. `http://www-128.ibm.com/developerworks/library/specification/ws-secmap/`

[22] T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing*, W3C recommendation, World Wide Web Consortium (W3C), Dezember 2002. `http://www.w3.org/TR/xmlenc-core/`.

[23] P. R. C. in Java: Application to Mobile Agent Security, *Walter binder, jarle g. hulaas, and alex villanzon*, in 1st Int'l Workshop on Secure Mobile Multi-Agent Systems at the 5th Int'l Conference on Autonomous Agents, Montreal, Canada, May 2001.

[24] F. Ishikawa, N. Yoshioka, Y. Tahara, and S. Honiden, *Toward Synthesis of Web Services and Mobile Agents*, in Proceedings of the AAMAS'2004 Workshop on Web Services and Agent-based Engineering (WSABE), New York, USA, July 2004. `http://honiden-lab.ex.nii.ac.jp/~f-ishikawa/docs/wsabe2004/camera-ready.pdf`

[25] D. Johansen, *Mobile agent applicability*, in Rothermel and Hohl [43], pp. 80–98.

[26] B. W. Lampson, *A note on the confinement problem*, Communications of the ACM, 10 (1973), pp. 613–615.

[27] M. S. Lew, N. Sebe, and J. P. Eakins, eds., *Proceedings of the International Conference on Image and Video Retrieval*, vol. 2383 of Lecture Notes in Computer Science, Springer Verlag, July 2002.

[28] J. Lopez, R. Oppliger, and G. Pernul, *Authentication and authorization infrastructures (AAIs): a comparative survey*, in Computers & Security, vol. 23, Elsevier, October 2004, pp. 578–590.

[29] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris, *On Software Agents and Web Services: Usage and Design Concepts and Issues*, in The 1st International Workshop on Web Services and Agent-based Engineering, Sydney, Australia, July 2003.

[30] Z. Maamar, Q. Z. Sheng, and B. Benatallah, *Interleaving Web Services Composition and Execution Using Software Agents and Delegation*, in The 1st International Workshop on Web Services and Agent-based Engineering, Sydney, Australia, July 2003.

[31] L. Moreau, *Agents for the Grid: A Comparison with Web Services (Part I: the transport layer)*, in Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002), Berlin, Germany, May 2002.

[32] OASIS, *Universal Description, Discovery and Integration (UDDI), Version 3*, technical committee specification, Organization for the Advancement of Structured Information Standards (OASIS), October 2003. `http://www.uddi.org/pubs/uddi_v3.htm`

[33] ———, *Security Assertions Markup Language (SAML), Version 2.0*, working draft, Organization for the Advancement of Structured Information Standards (OASIS), 2004. `http://www.oasis-open.org/committees/security`

[34] ———, *XML Access Control Markup Language (XACML), Version 2.0*, committee draft, Organization for the Advancement of Structured Information Standards (OASIS), September 2004. `http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-02.pdf`

[35] A. Padovitz, S. Krishnaswamy, and S. W. Loke, *Towards Efficient Selection of Web Services*, in The 1st International Workshop on Web Services and Agent-based Engineering, Sydney, Australia, July 2003.

[36] G. Pass, R. Zabih, and J. Miller, *Comparing images using color coherence vectors*, in Proc. ACM Conference on Multimedia, Boston, Massachusetts, U. S. A., November 1996.

[37] J. Peters, *Integration of Mobile Agents and Web Services*, in Proceedings of The First European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2005), Software Technology Research Laboratory, De Montfort University, Leicester, UK, April 2005, De Montfort University, pp. 53–58.

[38] J. Peters and J. Oetting, *Interoperability with Component Standards and Web Services*, tech. report, SicAri Consortium, 2004. Technical Report.

[39] V. Roth, *Distributed image indexing and retrieval with mobile agents*, in IEE European Workshop on Distributed Imaging, no. 1999/109 in IEE Electronics & Communications, Savoy Place, London, WC2R 0BL, UK, November 1999, IEE, pp. 14/1–14/5. ISSN 0963-3308.

[40] V. Roth and V. Conan, *Encrypting Java Archives and its application to mobile agent security*, in Agent Mediated Electronic Commerce: A European Perspective, F. Dignum and C. Sierra, eds., vol. 1991 of Lecture Notes in Artifical Intelligence, Springer Verlag, Berlin, 2001, pp. 232–244.

[41] V. Roth and M. Jalali, *Concepts and architecture of a security-centric mobile agent server*, in Proc. Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001), Dallas, Texas, U.S.A., March 2001, IEEE Computer Society, pp. 435–442. ISBN 0-7695-1065-5.

[42] V. Roth, U. Pinsdorf, and W. Binder, *Mobile Agent Interoperability Revisited*, in Mobile Agents 2001. Poster Session, K. Marzullo, A. L. Murphy, and G. P. Picco, eds., Atlanta, Georgia, USA, December 2001, 5th IEEE International Conference Mobile Agents (MA), IEEE Society Press, pp. 5–8.

[43] K. Rothermel and F. Hohl, eds., *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, vol. 1477 of Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg, September 1998.

[44] S. Sclaroff, L. Taycher, and M. L. Cascia, *ImageRover: A content-based image browser for the world wide web*, in Proc. IEEE Workshop on Content-based Access of Image and Video Libraries, San Juan, Puerto Rico, June 1997.

[45] J. R. Smith and S.-F. Chang, *An image and video search engine for the world–wide web*, in Proc. Storage and Retrieval for Image and Video Databases V (SPIE), San Jose, CA, USA, February 1997.

[46] L. Z. Varga, *WSDL2Agent: Tool to Help the Integration of Existing Web Services into Agent Systems and Semantic Web Service Environments*. `http://sas.ilab.sztaki.hu:8080/wsdl2agent/`

[47] G. Vigna, ed., *Mobile Agents and Security*, vol. 1419 of Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg, 1998.

[48] J. Vitek and C. Jensen, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, vol. 1603 of Lecture Notes in Computer Science, Springer-Verlag Inc., New York, NY, USA, 1999.

[49] S. Volmer, *Buoy Indexing of Metric Feature Spaces for Fast Approximate Image Queries*, in Proc. Eurographics 2001 Workshop on Multimedia, Manchester, UK, September 2001, pp. 121–130.

[50] W3C, *Simple Object Access Protocol (SOAP), Version 1.2*, W3C recommendation, World Wide Web Consortium (W3C), June 2003. `http://www.w3.org/TR/soap/`

[51] ———, *Web Services Architecture*, W3C working group note, World Wide Web Consortium (W3C), February 2004. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`

[52] J. E. White, *Mobile agents*, in Software Agents, J. Bradshaw, ed., AAAI/MIT Press, Menlo Park, CA, 1997, ch. 18, pp. 437–472.

[53] M. Woolridge and N. Jennings, *Intelligent agents: Theory and pratice*, Knowledge Engineering Review, 10 (1995).

[54] J. You and H. A. Cohen, *A new approach to image retrieval by fast indexing and searching*, in Proc. DICTA'97, Auckland, N.Z., 1997, pp. 425–430.

[55] S.-T. Yuan and K.-J. Lin, *WISE - Building Simple Intelligence into Web Services*, in The 1st International Workshop on Web Services and Agent-based Engineering, Sydney, Australia, July 2003.

Edited by: Henry Hexmoor, Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 16, 2006