Hyperdatabase Infrastructure for Management and Search of Multimedia Collections

Michael Mlivoncic, Christoph Schuler, and Can Türker

Swiss Federal Institute of Technology Zurich
Institute of Information Systems, ETH Zentrum
CH–8092 Zurich, Switzerland
{mlivonci|schuler|tuerker}@inf.ethz.ch

Abstract. Nowadays, digital libraries are inherently dispersed over several peers of a steadily increasing network. Dedicated peers may provide specialized, computationally expensive services such as image similarity search. Usually, the peers of such a network are uncoordinated in the sense that their content and services are not linked together. Nevertheless, users expect to transparently access and modify all the (multimedia) content anytime from anywhere not only in an efficient and effective but also consistent way. To match these demands, future digital libraries require an infrastructure that combines various information technologies like (mobile) databases, service-oriented architectures, peer-to-peer and grid computing. In this paper, we sketch such an infrastructure and illustrate how an example digital library application can work atop it.

1 Introduction

Future digital libraries shall provide access to any multimedia content anytime and anywhere in a user-friendly, efficient, and effective way. One problem of nowadays digital libraries is that they are dispersed over a network in an uncoordinated fashion such that each peer of the network often works isolated without regarding other peers that manage related content. Another problem is to efficiently handle the steadily increasing amount of requests for multimedia content. Besides, the distributed content should be kept consistent and provided in a personalized way. Hence, an infrastructure is required for digital libraries that is a reliable, scalable, customizable, and integrated environment.

To build such an infrastructure, the best aspects of (mobile) databases, service-orientation, peer-to-peer and grid computing must be combined. We refer to such an infrastructure as a *hyperdatabase* [1]. Within such an environment, databases and special servers provide basic services, such as efficient and reliable storage for various kinds of multimedia data like image, audio, and video. Service-orientation [2] helps to describe, customize, and deploy complex services, such as sophisticated search of images with respect to their content and annotations. The peer-to-peer paradigm [3] allows a loosely-coupled integration of digital library services and ad-hoc sharing of information, such as recommendations and annotations. Since certain services within a digital library are computationally intensive, e.g., the extraction of features from multimedia objects to

support content-based similarity search, grid technology [4] supports the optimal utilization of the given computing resources.

In the recent years, we have developed OSIRIS (**O**pen **S**ervice **I**nfrastructure for **R**eliable and **I**ntegrated process **S**upport) [5], which is a prototype of a hyperdatabase infrastructure. As common standards like WSDL and SOAP, OSIRIS helps to access a wide range of service types and allows isolated service calls. In addition, OSIRIS support processes (compound services) as a means for combining existing services and executing them under certain (transactional) guarantees. In a digital library, such services can be used, for instance, for the maintenance of dependencies among the various data repositories of the digital library. In this way, a sophisticated search may benefit from always up-to-date indexes.

In this paper, we describe how to organize such a self-contained application. The ISIS (**I**nteractive **SI**milarity **S**earch) application demonstrates an efficient management and organization of large multimedia collections atop OSIRIS. It features a wide range of functions that allow for metadata management as well as efficient and effective content-based similarity search on multimedia objects. The implementation of ISIS consequently follows the idea of service-orientation. Specifically, all digital library functionality is encapsulated by services, such as storage services for arbitrary types of media objects or feature extraction services for given media types. Such services are used to compose the entire ISIS application.

The infrastructure monitors the service execution as well as the routing of the corresponding services. Whenever necessary, it distributes and replicates certain services on multiple grid peers in order to boost the performance.

In principle, any service can be executed locally in a stand-alone manner, as with mobile devices, for example. However, due to several reasons, e.g., resource or license restrictions, some services might be locally unavailable. In such cases, the service execution depends on external services, which have to be bound and invoked on demand. On the other hand, in some cases the locally available services are sufficient to completely execute a compound service. For instance, even a content-based image similarity search does not require a feature extraction service provided the reference image is already part of the collection (i.e., is already indexed).

It is important to note that the entire service execution is transparent to the service designer. She therefore can fully focus on specifying and implementing the core service functionality itself. Service runtime environment tasks (startup, updates,...) and communication with the outside world are handled by the hyperdatabase infrastructure. As first experimental evaluations show, this infrastructure does not only support dynamic changes of the execution environment but is also able to scale with the expected huge number of users, services, and grid peers of future digital libraries.

The rest of this paper is organized as follows: Section 2 gives an overview of ISIS together with the requirements for the underlying infrastructure. Section 3 describes such an infrastructure, OSIRIS, and discuss various important issues like service execution and service registration. Finally, Section 4 concludes.

2 ISIS – A Service-Oriented Application

ISIS (Interactive SImilarity Search) is a powerful service-oriented application for efficient management and organization of multimedia collections. The application features a wide range of functions that allow meta data management as well as efficient and effective content-based similarity search on multimedia objects. The realization of ISIS consequently follows the idea of service-orientation. All basic functionality is encapsulated by a set of services.

We distinguish five classes of such services:

Storage Services. These services provide storage for arbitrary types of media objects. A storage service at a certain peer may be dedicated to a certain content like video clips or images. Storage services can also act as web-caches for remote content in order to speed-up access to often used data. A storage service closely monitors its attached repositories for changes. If, for example, a new object is added to one of its repositories, it will issue a "new object" event. Likewise, a (local) deletion of an object will lead to an "object-deleted" event. These events can be handled by the digital library infrastructure, for example, to keep the consistency between object data and indexed information.

Metadata Services. Such services maintain keyword annotations, textual descriptions, as well as other object properties, such as the membership in several (sub)collections, or predicates like "copyrighted". As an media object might be available in different versions at different locations (e.g, as thumbnail of the image at thumbnail-storage, primary high-resolution image copy at remote storage location), metadata services also keep track of those locations and corresponding properties of the object versions at the various locations (e.g. resolution or thumbnail/primary-role). There is no fixed vocabulary for describing such properties. Hence, ISIS can store arbitrary information about the objects. For example, one might want to store textual information gathered from the surrounding web pages together with an image or the artist and the song title together with a piece of music. Besides such "per-object" information, metadata services also maintain general knowledge about the object types themselves. This includes existing feature descriptors for an object type, the availability of feature extractors and indexing containers within the digital library infrastructure that are able to manage a given feature descriptor.

Feature Extraction Services. Content-based retrieval depends on features that describe the raw content of media objects in a certain feature domain. For example, the pixel information of an image can be described in the color or texture feature domain [6–8], while a piece of music can be described in terms of beat and pitch [9]. A feature is therefore a kind of descriptive fingerprint for an object. In content-based retrieval, object similarity is expressed as similarity of their descriptors in a given feature domain. For a given media type, there could be several feature extraction services computing various kinds of feature descriptors. Also, there is more than one way to

describe a concept like color. For example, we could use a histogram with 64, 256 or whatsoever bins. We could also use color moments. There are many meaningful descriptors and variants around. One might want to use several of them — even in combination. In many cases, the extraction of features is a computationally very expensive task. Therefore, feature extraction can profit from grid infrastructures.

Indexing and Search Services. Indexes allow for efficient search of objects. Different indexes have to handle data from different domains: Besides often high dimensional feature descriptors, object predicates and numerical values as well as keyword annotations of the objects should be indexed efficiently. Searching over an arbitrary *combination* of those attributes efficiently is a non-trivial task. Efficient search strategies on this level are however beyond the scope of this paper. For further information please refer to [10–12]. At this point, we only state that concept and design of such a service should be carefully chosen in a way that it allows for massive scalability through dynamic partitioning of a query onto a set of several search services. Those services will be spread all over the grid infrastructure, similar to the extraction services.

Presentation Services. These services provide frontend functionality to browse and query the multimedia collection and also to initiate some maintenance and administrative tasks. In ISIS, this task is shared among services for the interactive part (frontend), the layout part (XML-to-HTML rendering) and supporting services (session management and template repository).

The entire ISIS application consists of a set of compound services over these basic services. Once the application logic is divided into such basic services, they can be distributed and replicated without changing the description (implementation) of any (compound) service. Figure 1 shows the insertion of a multimedia object as an example for a compound service. The given service is triggered by the "new object" event of a storage service mentioned above when a new media object physically enters the repository.

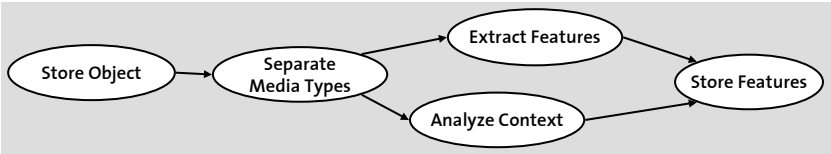

Fig. 1. Compound Service Insert Multimedia Object

The first activity of the compound service is to store the object, i.e., the location and available meta information about the object is stored by the metadata service. Depending on the media type, further information is extracted. In case of a web document, the object will not only contain an image, but also some text surrounding this image on the page. By analyzing the HTML source, it is

possible to gather some layout information and applying some heuristics in order to determine textual descriptions that might be relevant for the image. This text can be indexed later on. Independent of the analysis of the context of the image and its surrounding, the extract features activity uses the raw pixel information of the image to extract several descriptors for color and texture. Note that it is transparent to the user, whether this activity is a single service or a compound service which is composed of single feature extraction services. The distinction between a single and a compound service is important for the infrastructure. By explicitly knowing about the semantics of the various activities of a compound service, the infrastructure is able to further parallelize the extraction to achieve better performance. The store features activity hands all gathered object descriptors and metadata information over to the metadata service, which will in turn care for the indexing and replication of each data item in a suitable way.

Infrastructure Requirements

ISIS performs some complex and sometimes computational expensive tasks. As ISIS may change over time, e.g., when new feature descriptors are becoming available, we need an infrastructure that provides us with flexibility in order to define and modify the logic of the digital library applications, i.e., the corresponding compound services, as necessary. In order to focus on the core digital library tasks, the infrastructure should support a transparent way of communication among services. For example, while designing the extraction service, we do not want to *explicitly* deal with workload distribution, it should be sufficient to "solicit" that a certain task, e.g., a feature extraction, should be executed on any one of the potential service providers. Thus, the infrastructure must also support service discovery. Service providers should be able to declare what kind of functionality they are offering and service users should be able to find them.

As mentioned before, it can be useful, if application logic specifies compound services over the basic services (cf. the feature extraction example). In such cases, feature extraction as well as search can profit even from a higher degree of parallelism. As multimedia content tends to be storage intensive — especially with large-scale general purpose digital libraries — one might also want to distribute the storage services and searching facilities over the peers of the grid. Besides this complexity, we still demand that the overall digital library should be reliable, i.e. (temporary) unavailability of single service *instances* shall not jeopardize the operation of the overall system. Services once invoked (like the insertion of an object) should lead to a guaranteed execution of the defined activities and thus ensure the consistency of the data within the metadata and indexing services. Mentioning consistency, we also want to ensure highest possible "freshness", i.e., changes should be propagated "immediately" instead of monthly updates like in Google and other major web search engines. On object deletion, references to that object should be removed immediately from all indexes. Also, changes of feature data should immediately be propagated to *any* indexing service related to that data.

3 Hyperdatabase Infrastructure for ISIS

In the following, we sketch how a hyperdatabase infrastructure provides us with all the required functionality as elaborated so far.

3.1 Overview of the Infrastructure

A hyperdatabase supports applications following the ideas of a service-oriented architecture. Using a distributed peer-to-peer network, service requests are transparently routed along the connected peers. Following the concept of service-oriented architecture, a service can be *atomic* or *compound*. Compound services are composed of existing services. The composition is realized using the notion of a transactional process [13]. By defining a data and control flow for processes, the involved service calls must appear in the specified application specific invocation order. Transactional processes allows for providing execution guarantees similar to transactions in classical databases. While databases focus on basic data querying and manipulation operations, a hyperdatabase orchestrates service calls. These service calls are executed according the description of the compound service.

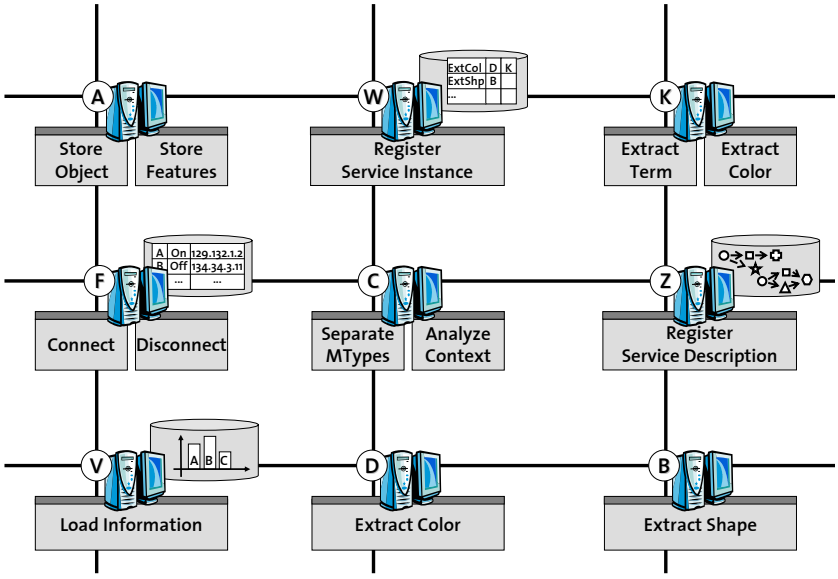

Fig. 2. Hyperdatabase Infrastructure

In order to provide this functionality, the hyperdatabase infrastructure consists of a software layer installed on each peer of the grid (dark gray layers in Figure 2). This layer integrates the peer into the overall hyperdatabase network.

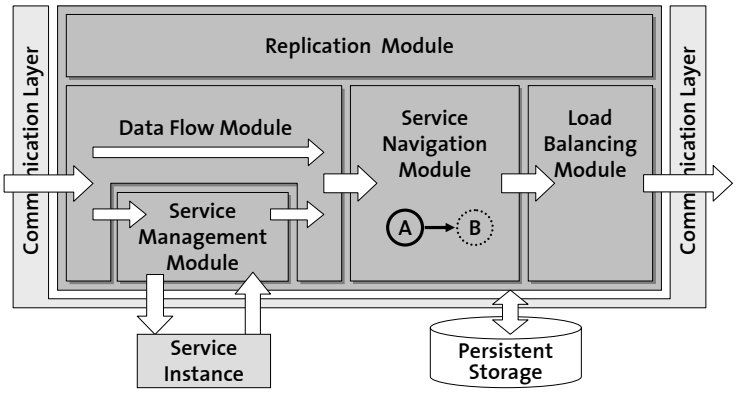

Fig. 3. Hyperdatabase Layer

Ideally, this layer comes together with the operating system like TCP/IP stack does (comparable to the .NET framework). By this integration, local available services can be registered and used by all peers in the grid. Moreover, the local hyperdatabase layer is now enabled to transparently call remote services.

Figure 3 shows the architecture of the hyperdatabase layer, which we will discuss in more detail later. Beside communication, the layer also supports management of compound services and load balancing. This functionality depends on grid common knowledge accessible via the replication manager available at every peer.

OSIRIS [14] as an implementation of a hyperdatabase is realized as a service-oriented application itself. Besides the core hyperdatabase layer, additional system functionality is needed to organize the grid and to provide service transparency and the execution of compound services. OSIRIS implements this functionality also in terms of services. As depicted in Figure 2, OSIRIS provides core services like Register Service Description, Register Service Instance, Connect, and Disconnect. The figure also shows that there is no distinction between core services and atomic application-specific services such as Extract Color or Extract Term.

3.2 Peer-to-Peer Service Execution

As mentioned before, OSIRIS distinguishes two classes of services: atomic versus compound services. The execution of an atomic service consists of one call to a function provided by a service in the grid. A call to a compound service however initiates a peer-to-peer execution. According to the description of a compound service, the contained services, i.e., the process activities, are executed sequentially. Although the execution semantics of the two service classes are different, the invocations of both services are not distinguishable to the caller. In other words, the caller does not care about whether the service is atomic or compound.

The same holds for activities of a compound service, which can be compound services themselves.

OSIRIS provides a transparent way of calling services. A service request that is sent to any peer of the network is transparently routed by the local hyperdatabase layer to an available instance of this service type. For that, OSIRIS implements a distributed service bus that provides routing and load balancing over all existing service instances.

Albeit the infrastructure hides the difference between an atomic and a compound service at the call interface, the execution within the infrastructure layer respects and exploits the differences between atomic and compound services. While the first has to be routed to a service instance, the latter must be driven by the infrastructure itself since its activities can spread over multiple peers of the grid. Therefore, each peer has to provide a minimal service manager that supports a peer-to-peer service execution. This includes the ability to call local services, to navigate to the subsequent activity of the compound service, and to handle execution failures. A compound service consists of a set of ordered services. These services has to be executed according the order defined by the service description. This information is available at the local replication manager. After instantiating a new compound service at any peer in the grid, it is migrated to the hyperdatabase layer of a peer that provides an instance for the first activity.

Figure 3 shows the flow of the service execution of one single step in a compound service. After the service has migrated to the hyperdatabase layer, it enters the *data flow module*. A part of the service data is needed to prepare the call of the service. The *service management module* executes a function at the service instance. After the execution of the service the resulting data is incorporated into the context of the compound service. The next task is to determine the service that has to be executed subsequently. Based on the locally replicated part of the service description, the *service navigation module* decides which service type to call next. The *load balancing module* finally routes the service instance to an available service instance considering system workload. This routing requires grid configuration information. Therefore, the *replication module* has also to provide grid configuration information. During the execution of a compound service, the service instance migrates from one peer to the next in a truly peer-to-peer fashion. After executing the last activity of the compound service, the response is sent back to the caller, as in the case of an atomic service.

Example 1. Assume a multimedia object is inserted into the ISIS digital library by calling our example compound service Insert Multimedia Object. OSIRIS initiates a service instance for this particular service call. After initialization, the instance is routed to the provider of a service for the first activity. In this way, the service instance reaches peer A in our example in Figure 2. In the context of the compound service, then a local call to the service Store Object is performed. After this call, the service instance migrates to peer C in order to execute the service Separate Media Types. This migration is done directly using a peer-to-peer connection between A and C. The next two service calls can be performed

parallel. This requires that the service instance splits and navigates separately along the definition paths. The first part of the instance stays at peer C in order to execute the service Analyze Context, while the second part is migrated to peer K or D. Considering the current workload of the peers D and K, the service instance migrates to peer K. Finally, the service Store Features has to be executed on peer A. Since the original service instance was split and distributed on the network, OSIRIS has to synchronize and join these parts before migrating to peer A. After finishing the local call, the compound service is completely executed and the response is sent to the caller.

3.3 Registration of Service Descriptions

In the previous discussion, we have already distinguished between service types and service instances without explicitly mentioning this. In service-oriented architectures, this distinction enables a transparent routing during service execution. In fact, a service type is called. The infrastructure matches a currently available service instances to perform the execution of the requested service type. To enable this behavior — in the literature referred as *enterprise service bus* [15] — all available service types have to be registered in the system. While calls to atomic services just have to be routed through the system, compound services have to be executed by the infrastructure itself as described in the previous subsection. For that, the description of compound services has to be published to the infrastructure. In OSIRIS, this publication is handled by the service Register Service Description.

Whenever a service description is published using this service, it is analyzed and prepared for replication along the peers. Keep in mind that service execution is done in a peer-to-peer fashion relying on locally replicated information. To provide exactly the information that is required to perform a completely peer-to-peer execution, the service description has to be enriched and divided into parts containing all information concerning the execution of one contained service. These parts of service description is then replicated to the peers hosting an instance of the corresponding service type. This strategy allows peer-to-peer execution to have most information already available at the local replication module.

Example 2. Assume that the description of the compound service Insert Multimedia Object is inserted into the system. The service Register Service Description splits the definition into five parts — one part for each activity of the compound service. The part concerning the service Extract Features contains all information about the parameter handling the call of that service. This information is needed by the *service management module* of the hyperdatabase layer at the peers D, K and B, respectively. In addition, the service navigation module needs to know about the subsequent services. All this information is replicated to the peers D and K immediately after inserting the service description. The peer-to-peer service execution will route every instance of the service Insert Multimedia Object to one of these peers. Consequently, the corresponding part of the service description should be replicated there.

3.4 Registration of Service Instances

A grid infrastructure has to deal with continues changes of the overall system configuration. Service instances may join and leave the system quite frequently. Therefore the infrastructure has to keep track of the current configuration. OSIRIS provides completely transparent service calls by dynamically replicating information about the available service instances to the corresponding grid peers. This replication is based on a publish-subscribe mechanism, which is described in [14]. While the call of a service can occur at any peer, no prior replication can be performed in order to speed up this routing. However, a temporary replication is reasonable since the probability that this information will be needed in the near future is rather high. Beside this temporary replication, information needed during execution of compound services can be predicted by analyzing the service descriptions. In addition, the information about the subsequent services are needed at a particular peer.

Example 3. In the configuration depicted in Figure 3, the peer A holds the replicated information about the current instances of service Separate Media Types since within the compound service Insert Multimedia Object this service must be invoked after the service Store Object, which is provided at peer A. This way a lot of information is replicated along the grid in order to perform and optimize peer-to-peer execution of services. Assume a new instance of the service Separate Media Types is started at peer B and registered via the service Register Service Instance. As a consequence, OSIRIS triggers the replication of additional configuration information to peer B — mainly the Separate Media Types part of the service Insert Multimedia Object and information on available instances of Analyze Context. Extract Features is a compound service and thus has no location associated with. Since the (parallel) activities of this compound service are Extract Color and Extract Shape, the location information about these two services are replicated to peer B.

3.5 Stand-alone Service Execution

The peer-to-peer execution of compound services relies on the transparent routing of service calls. OSIRIS can provide this transparent routing only within the same OSIRIS cell. A cell corresponds to a separate grid environment. Within such a cell, each service of the registered peers is provided to all peers of that cell. As a special case, a complete cell can be installed on one peer. This installation includes OSIRIS core services as well as all services of an application such as ISIS. This peer can also be a mobile device. Albeit local services can be used to realize a stand-alone application, some service types like Extract Feature cannot be performed efficiently by the mobile peer. In order to execute such a service, the peer should join a larger cell to perform this service on a peer having more resources. We also allow peers to join a cell without registering its available services to the cell. In this case, the peer acts as a service user. This kinds of a join can be performed by using a proxy service, which provides a bridge between

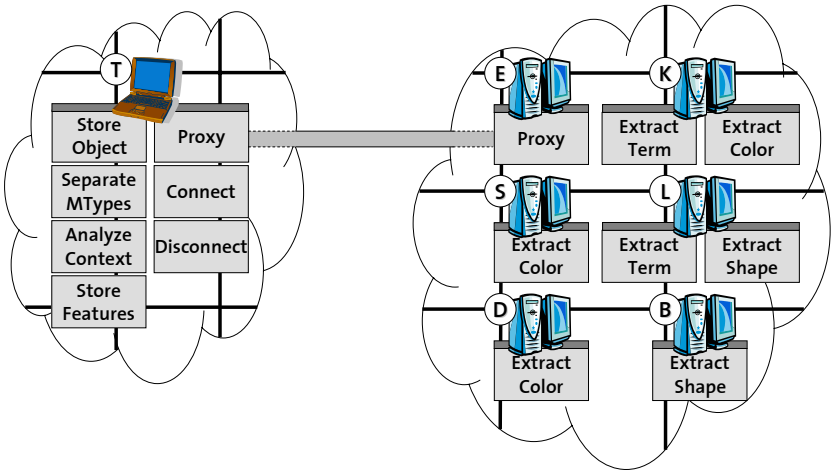

Fig. 4. Proxy Services for Stand-alone Service User

the two cells. As depicted in Figure 4, two instances of the proxy service can be linked in order to establish a bridge. A proxy service provides a service stub and forwards all service calls to the other cell. OSIRIS routes the call to a real service instance.

Note that this forwarding of service calls works also for compound services. However, the peer-to-peer service execution cannot use the bridge to migrate. If the mobile device calls a compound service on the remote cell the proxy service will forward the service call and the complete service will be executed at the remote cell. If a local compound service contains an activity for which only a remote service instance is available, the local proxy service provides a stub for that service. In the context of the local grid, the proxy service executes the service. Therefore, the compound service instance will stay at the local device while executing the service at the proxy service. In this way, the concept of a proxy allows for exploiting services of a remote grid cell.

4 Conclusions

As we have seen in this paper, service-orientation helps to describe and implement complex digital library applications like ISIS as compositions of services. A hyperdatabase infrastructure like OSIRIS, which combines service-orientation with peer-to-peer and grid computing, is then able to exploit the knowledge about the services and their composition to execute them in an optimal fashion. Following the idea of grid computing, the execution of service calls respects parameters like the workload of the peers to dynamically select the best fitting service instance. Besides, OSIRIS is able to replicate service instances on demand on any peer of the grid, and thus to optimize the utilization of the given

resources. The peer-to-peer style of service navigation avoids that the navigation becomes a bottleneck of the overall system, and thus provides the basis for a scalable infrastructure. The current implementation of ISIS atop OSIRIS demonstrates this very nicely.

References

1. Schek, H.J., Schuldt, H., Schuler, C., Weber, R.: Infrastructure for Information Spaces. In: Advances in Databases and Information Systems, Proc. of the 6th East-European Symposium, ADBIS'2002. Volume 2435 of Lecture Notes in Computer Science, Springer-Verlag (2002) 23–36
2. Schmid, M., Leymann, F., Roller, D.: Web Services and Business Process Management. IBM Systems Journal **41** (2002) 198–211
3. Curley, M.G.: Peer-to-Peer Computing Enabled Collaboration. In: Proc. of the Int. Conf. on Computational Science, ICCS 2002. (2002) 646–654
4. Foster, I., Kesselmann, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In: Global Grid Forum, 2002. (2002) http://www.gridforum.org/ogsi-wg.
5. ETH Database Research Group: OSIRIS: an Open Services Infrastructure for Reliable and Integrated process Support. http://www.osiris.ethz.ch (2004)
6. Niblack, W., Barber, R., Equitz, W., Flickner, M., Glasman, E.H., Petkovic, D., Yanker, P., Faloutsos, C., Taubin, G.: The QBIC Project: Querying Images by Content, using Color, Texture, and Shape. In Storage and Retrieval for Image and Video Databases. Volume 1908 of SPIE Proceedings (1993) 173–187
7. Stricker, M.A., Orengo, M.: Similarity of Color Images. In: Storage and Retrieval for Image and Video Databases. Volume 2420 of SPIE Proceedings (1995) 381–392
8. Dimai, A.: Spatial encoding using differences of global features. In: Storage and Retrieval for Image and Video Databases. Volume 3022 of SPIE Proceedings (1997) 352–360
9. Tzanetakis, G., Cook, P.: Audio Information Retrieval (Air) Tools. In: Proc. Int. Symposium for Audio Information Retrieval, ISMIR 2000 (2000)
10. Weber, R., Schek, H.J., Blott, S.: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: Proc. of the 24th Int. Conf. on Very Large Data Bases, VLDB'98, Morgan Kaufmann Publishers (1998) 194–205
11. Weber, R., Schek, H.J., Bollinger, J., Gross, T.: Architecture of a Networked Image Search and Retrieval System. In: Proc. of the 8th ACM CIKM Int. Conf. on Information and Knowledge Management, ACM Press (1999) 430–441
12. Böhm, K., Mlivoncic, M., Schek, H.J., Weber, R.: Fast Evaluation Techniques for Complex Similarity Queries. In: Proc. of the 27th Int. Conf. on Very Large Data Bases, VLDB 2001, Morgan Kaufmann Publishers (2001)
13. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation in Transactional Processes. ACM Transactions on Database Systems **27** (2002) 63–116
14. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Peer-to-Peer Process Execution with OSIRIS. In: Proc. of the 1st Int. Conf. on Service-Oriented Computing, ICSOC 2003. Volume 2910 of Lecture Notes in Computer Science, Springer-Verlag (2003) 483–498
15. Chappell, D.: Enterprise Service Bus. O'Reilly (2004)