Towards an information infrastructure for the grid

by S. Bourbonnais
V. M. Gogate
L. M. Haas
R. W. Horman
S. Malaika
I. Narang
V. Raman

In this paper we present our vision of an information infrastructure for grid computing, which is based on a service-oriented architecture. The infrastructure supports a virtualized view of the computing and data resources, is autonomic (driven by policies) in order to meet application goals for quality of service, and is compatible with the standards being developed in the technical community. We describe how we are implementing this vision in IBM today and how we expect the implementation to evolve in the future.

Grid computing offers the power to address some of the world's most challenging problems; for example, struggles to prevent cancer and cure smallpox, to reliably predict earthquakes and global warming, and many others. Computationally intensive analytic applications could also benefit: accurate risk computations could help investment companies minimize losses; insurance companies could more rapidly detect fraud. Two key benefits of grid computing would enable these advances. First, grids harness heterogeneous systems together into a megacomputer, and hence, can apply greater computational power to a task. Second, a grid virtualizes these heterogeneous resources, so that applications for the grid can be written as if for a single, local computer, vastly simplifying the development needed for such powerful applications.

Of course, these wonderful applications depend not only on computing power, but also on data—and often on vast volumes of heterogeneous, distributed data, collected or generated by various groups, and stored in diverse systems. The data sources might be files, databases, or applications, and the data might be structured (e.g., relational), semi-structured (e.g., Extensible Markup Language—XML—documents) or unstructured content (e.g., images). For the promise of grid computing to be fulfilled, not only must we harness and virtualize multiple computing resources, but we must also abstract and hide the diversity and distribution of these various information sources to provide applications with a single, powerful virtual-information store for their virtual computer.

In this paper, we propose an information infrastructure for grid computing that will meet this lofty goal. This information infrastructure will have three key characteristics; in particular, it will be:

- *Virtualized*—allowing a collection of distributed information resources to be shared and managed as if they were a single information store, although they may in fact remain fully distributed.
- *Autonomic*—ensuring that the interconnected information systems can be managed effectively and efficiently through self-management just like the human autonomic nervous system.
- *Open*—utilizing open interfaces and agreed-upon standards to enable highly interoperable systems and processes.

These characteristics will enable the information infrastructure to produce information on demand, in the spirit of IBM's on demand initiative. More specifically, they will help customers derive the benefits they expect from a grid. Customer motivations for using grid computing, besides the hope of meeting some of the grand challenges outlined above, include such pragmatic desires as:

- Enhanced collaboration within and across enterprises by rapid integration and sharing of distributed heterogeneous information
- Scalability by adding new copies of data to offload swamped servers
- Faster response times by efficient processing of data-intensive queries
- Better availability through transparently exploiting alternative copies of information
- Lower cost through leveraging existing resources and easier administration
- Faster time to value by simplifying the task of application development

These motivations have been echoed by numerous customers in a range of industries, including life science, finance, manufacturing and the scientific community. Our proposed infrastructure for information will be the backbone the grid needs to meet these demands.

An information infrastructure to meet these needs requires a flexible architecture rich in function. We believe these needs will best be met by a service-oriented architecture in which each service is self-configuring and self-maintaining, guided by user-provided policies. The services will, together and individually, provide certain transparencies, or abstractions, that shield applications from the complexities of a distributed, heterogeneous environment. Core services of this architecture include data access and integration, discovery, meta-data, data placement, change-publish (for publishing changes to data), replication, and caching. Services can build on (use) other core services as well as other grid services such as registration, billing, policy management, and so on. The architecture will be an open one, supporting the existing and evolving grid standards[1] so that many implementations of individual services and of the collective infrastructure will be possible.

This paper proposes a service-oriented, policy-driven information infrastructure for the grid. We detail our vision and the services required in the next section, and illustrate how these services could be used to facilitate data-intensive applications on the grid. In the third section, we look at how these services are being implemented today. The following section focuses on the likely future evolution of our implementation. Standards are critical to a service-oriented architecture, and we describe the relevant standards efforts in the section "The role of standards."

An information infrastructure for the grid

For a large-scale, distributed grid to be successful, the grid infrastructure should make application development easy. Ideally, all the resources used in computing—processors, storage, databases, applications—should be virtualized in such a way that the application developers, administrators, and users are shielded from the details and dynamics of how the necessary services are provided. Specifically, a requestor of grid services should not be affected by the number of data and computing resources, their locations, their failures, and their specific hardware and software configurations.

The grid infrastructure should transparently provision the right data and computing resources for each application. We also want application programmers to be able to specify end-to-end goals for the quality of the provisioning. Those goals, often referred to as *quality of service* (QoS) goals, may include goals for the system availability, response time, throughput, number of concurrent users supported, currency or accuracy of the data, and so on. The grid infrastructure needs to support the definition of policies that set QoS goals and define the conditions under which they must be met.

Thus, our vision for the information infrastructure consists of a set of services that individually and collectively support a set of *transparencies*. By upholding these transparencies, the services virtualize the underlying resources, simplifying application development. A set of *policies* governs the functioning of these services. This vision extends the work done in the Global Grid Forum (GGF) on the Open Grid Services Architecture[2] (OGSA) to the information infrastructure, building on the work being done in the data-oriented working groups of GGF (e.g., References 3 and 4). In this section, we first provide an example of a grid scenario that we want the information infrastructure to handle. We then identify the set of services we believe are required for the information infrastructure, followed by a description of the transparencies we expect our services to provide to ap-

plications and users. A fourth subsection provides a brief overview of policies and illustrates their use in our example scenario. Finally, we illustrate how the services will interact to maintain transparencies and meet policy goals.

An example. To understand the extent and power of our information infrastructure, consider a worldwide grid of hospital information systems, containing patient records such as hospital visits, medication history, doctor reports, x-rays, symptoms history, genetic information, and so on. Transparent access to such a grid with QoS guarantees could enable a variety of useful tasks. We outline a few examples below.

Patient Health Overview: Many health-related applications would benefit from an integrated view of medical records for individual patients. Today these records may be scattered across various hospitals and doctors' offices. For example, a doctor planning surgery could use such views to provide better, safer care. If some of the records for the patient were unavailable at a given time, the doctor would still like to get as many as possible to continue to plan the surgery to the extent possible.

Computer-Aided Diagnostics: To diagnose diseases, a doctor could compare a given patient's symptoms with those of other patients around the world. This would be especially helpful for diseases that are uncommon in a region and therefore unfamiliar to the local doctor. Again, a partial result set would be better than no information although the doctor might want even the partial results to span a representative subset of the data. Further, when certain symptoms are found, they may be propagated to the Centers for Disease Control and Prevention (CDC) to allow tracking of potential epidemics, or disseminated to other physicians in the area to alert them to the increased likelihood of a particular disease.

Pharmaceutical Research: A researcher could study patients with common characteristics to study the efficacy of various treatments on classes of people. The analysis would be both computation- and data-intensive, but the data and computation would be dynamically distributed among multiple nodes on a grid. Further, the researcher would need to link data about patients from hospital records with the pharmaceutical companies' own experimental results. As several researchers in a company may be working on related areas, the researcher would also like to be informed when new results on particular biological or chemical substances are made available.

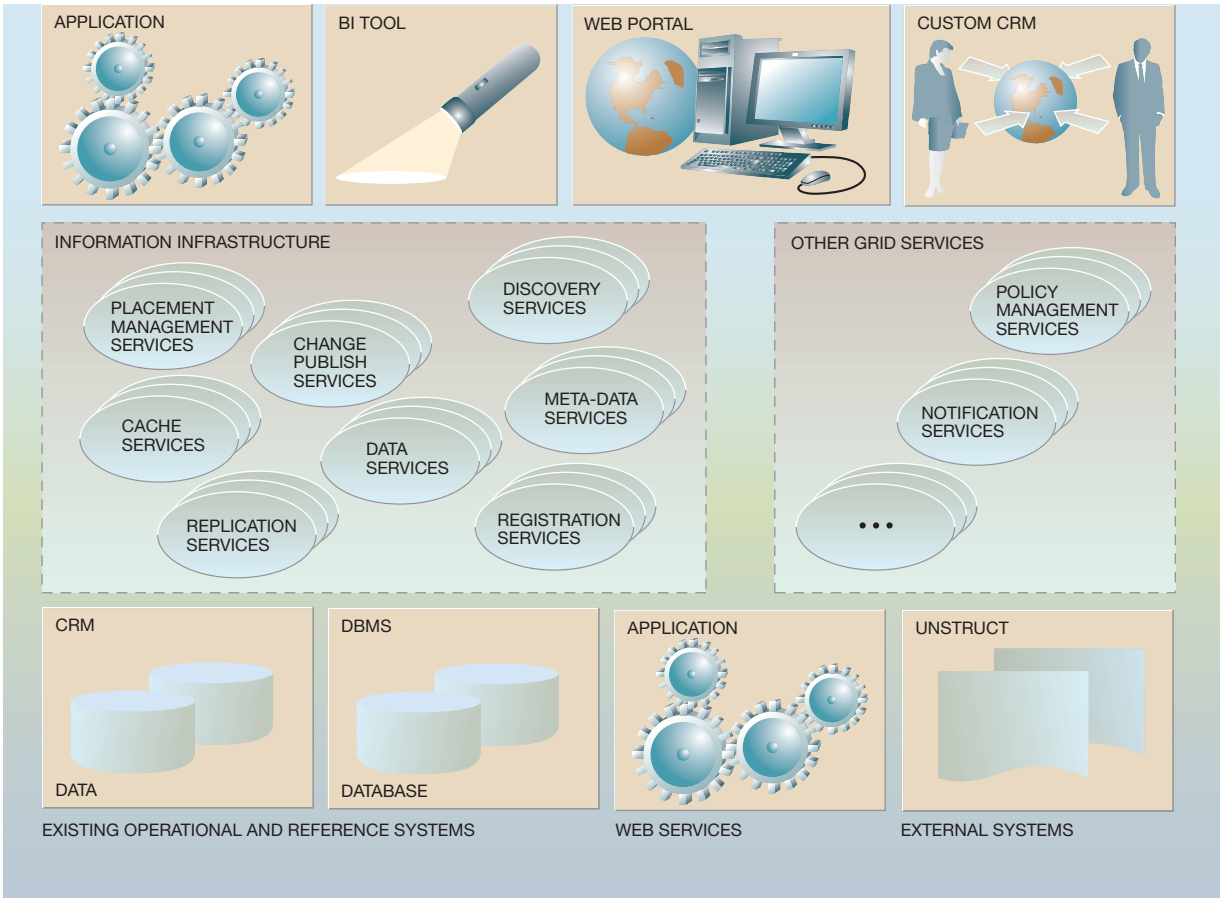
The challenge in performing these tasks is that medical information systems are distributed, heterogeneous, and autonomously administered. Patient information is independently entered at different hospitals, which bear responsibility for the security and privacy of this data. Data sources may come and go, due to events such as new medical centers joining, hardware and software failures, or even password expiration; thus, it is very difficult for application developers to program these tasks directly against data sources. Because the proposed information infrastructure presents a unified view of these diverse data sources, application developers can write their programs as if all the data were centrally located and always available.

A set of services for the information infrastructure. Figure 1 illustrates the set of services we imagine for the information infrastructure for the grid. While no system provides all of these capabilities today, prototypes and even commercial versions of some services do exist. Standards activities in GGF will ensure that as more of the pieces are created, they can be put together to form the information infrastructure we envision here.

The information infrastructure provides applications, such as those shown at the top of Figure 1, with transparent access to heterogeneous, dispersed information sources, like those that appear at the bottom of the figure. To add an information source explicitly, it is first registered via the Registration Services. This step provides information pertaining to that source to the Meta-data Services, which know about all available sources and how they ought to be represented within a unified view to the consuming applications. Discovery Services can be used to automatically identify possible information sources and to help knit them into a unified view by depositing the required meta-data into the Meta-data Repository (not shown) using Meta-data Services. A Discovery Service might use a Registration Service to enter sources it has found in the Meta-data Repository.

Arguably the most essential of the services shown are Data Services. Data Services handle requests for information from applications or from other services. A particular Data Service may represent one specific information source, for example, a file (mydata.xls) or a relational database (a single Oracle** in-

Figure 1 Services for an information infrastructure for the grid



stance). However, it could also be implemented by middleware that encapsulates and translates among several data sources. Distributed file systems, gateways, mediators, and federated systems are examples of such middleware. In this case the Data Service represents the collection of information accessible by the middleware that implements it. By using the other services of the information infrastructure, it is possible to build a very sophisticated Data Service that can handle complex queries, locate relevant information sources for a query, and ensure that QoS goals are met.

Depending on the access patterns and locality of the consuming applications, a Placement Management Service can improve response time or availability by creating caches or replicas. In effect, a Placement Management Service automatically distributes copies of the data to optimize performance. The Placement Management Service provides intelligence to determine what data to copy to meet the QoS goals. It relies upon Replication or Cache Services to actually do the work, creating the data copies, registering their existence with a Meta-data Service, and then populating the copies. Of course, Replication and Cache Services can also be used independent of the Placement Management Service. For example, an administrator might use a Replication Service to ensure the availability of certain data for a disaster recovery scenario. Change-Publish Services can detect changes in data and deliver them to a consumer, providing the ability to "publish" changes. A Replication Service might use a Change-Publish Service to know when to propagate and apply changes for a replica, or an application might subscribe to changes in data it particularly cares about.

Finally, these services provided by the information infrastructure can use other grid services as indicated on the righthand side of Figure 1. For example, the overall grid architecture[2] provides a Notification Service, which could be used to inform an autonomic Meta-data Service of relevant changes in the state, meta-data, or location of information sources. Another important grid service is the Policy Management Service, which is used by most of the information infrastructure services to discover what QoS goals they must sustain.

We believe this set of services, working together as described above, is an information infrastructure that can fully support the needs of grid computing and its users. In the next two sections we first discuss the transparencies that we expect these services to provide, and then how policies are used by the services. With that background, we present a scenario using several of the services, elaborating on how they can be used together in the grid context to provide these transparencies and to meet QoS goals.

Transparencies needed for the information infrastructure. We characterize the extent to which an information infrastructure supports virtualization through the notion of transparencies.[5] There are a number of different kinds of transparency that will have to be provided before we realize a complete information infrastructure for the grid. Each transparency masks from the user some type of difference, idiosyncrasy, or implementation detail of the underlying data sources.

The most fundamental type of transparency for the grid is *location transparency*. Location transparency shields the user from awareness of where data are stored. Without location transparency, users must make requests directly of a particular data source. Location transparency is usually implemented via some middleware (often called a mediator or broker) that is responsible for interpreting each information request and directing it to the right location. True location transparency is not really possible without *heterogeneity transparency*. This transparency protects the user from the details of how data are stored and accessed by the actual source systems, including the language or programming interface supported by the data source (and the dialect the source supports), how the data are physically stored, whether the data are partitioned, or the networking protocols used. The user should see a single uniform interface, complete with a single set of error codes.

However, even with location and heterogeneity transparency, users typically need to know the name of the data they need. *Replication transparency* extends the concept further: one name can now be used to access any of several copies, depending on which is available or more cost-effective. *Name transparency* is a generalization of these transparencies that suits the dynamic, large-scale, grid environment. With name transparency, a user does not have to know the name of the data, only the logical characteristics or QoS properties. In the hospital grid example, a pharmaceutical researcher may want to find the records of all patients in a specific age group, having a specific symptom. This researcher does not know or care about the location of these patients, or which medical facilities provide their records. The researcher would like to qualify his request by only the logical attributes of the desired patient records. The alternative solution of hard-coding the tables or documents in the query or application would mean that every source entry, exit, or failure would result in rewriting the query or recompiling the application. With name transparency, the sources can be migrated, cached, or replicated without changing the applications because the application does not specify the data sources explicitly. As part of the data access request, applications can specify QoS goals such as proximity, staleness, and cost, and suitable replicas are chosen automatically.

If grids are successful in the long term, they will evolve to span organizational boundaries (as suggested by the hospital grid example), and will involve multiple autonomous data and computational resources. As far as possible, applications should be spared from separately negotiating for access to individual sources, whether in terms of access authorization or in terms of resource usage costs. *Ownership* and *costing* transparencies address these needs. This will require protocols beyond the scope of this paper to address.

If our infrastructure is to autonomously meet QoS goals, two other transparencies are needed. *Parallelism transparency* gives applications processing data on a grid the benefits of parallel execution over grid nodes without explicit coding. The application should only have to specify the dependencies among the tasks it needs to execute and the policies that will affect scheduling, such as priority of the job or response time required. A workflow coordination service should automatically orchestrate this workflow in a parallel fashion, taking care of data movement, node failures, and so forth, to meet the response time

goals. In cases where the processing consists of traditional data management tasks like online transaction processing or online analytical processing (OLTP or OLAP), the system should automatically expand (or shrink) by adding (or removing) nodes in response to workload fluctuations to meet QoS goals such as transaction throughput and response time. Finally, applications should be able to maintain distributed data in a unified fashion, as if the data were stored in one central place (*distribution transparency*). This maintenance involves several tasks, such as ensuring consistency and data integrity, auditing access, taking backups, and so on.

Policies and autonomic computing in the information infrastructure. "Policy" is a general concept, used not only by the information infrastructure but by many other grid services. For example, other aspects of system operation, such as workload management, are guided and constrained by policies. Further, these other services interact with the information infrastructure services to maintain the system autonomically in accordance with these policies. In this section, we focus on the kinds of policies needed to govern the information infrastructure, and on how the information infrastructure works with other services to comply with these policies autonomically.

A policy is a prioritized set of QoS goals. For the information infrastructure, QoS goals have to do with the availability and latency of data, with query performance, replication throughput, and so on. QoS goals have terms that must be met (also known as criteria) and conditions that constrain the goal. A *service level agreement* (SLA) is a formal contract (both monetary and legal) that is struck between provider and consumer. This contract specifies minimum expectations (or obligations), known as terms and conditions, that the provider must meet. It usually includes penalties or refunds if the terms and conditions are not met. Although the process for arriving at the terms and conditions, that is, the negotiation among selected parties, as well as the process for compliance monitoring, are both important aspects of an SLA, they are no different for the information infrastructure than for other grid systems, so we will not discuss them further. (As an example, the WSLA project[6] is an attempt to formalize this notion for Web services.)

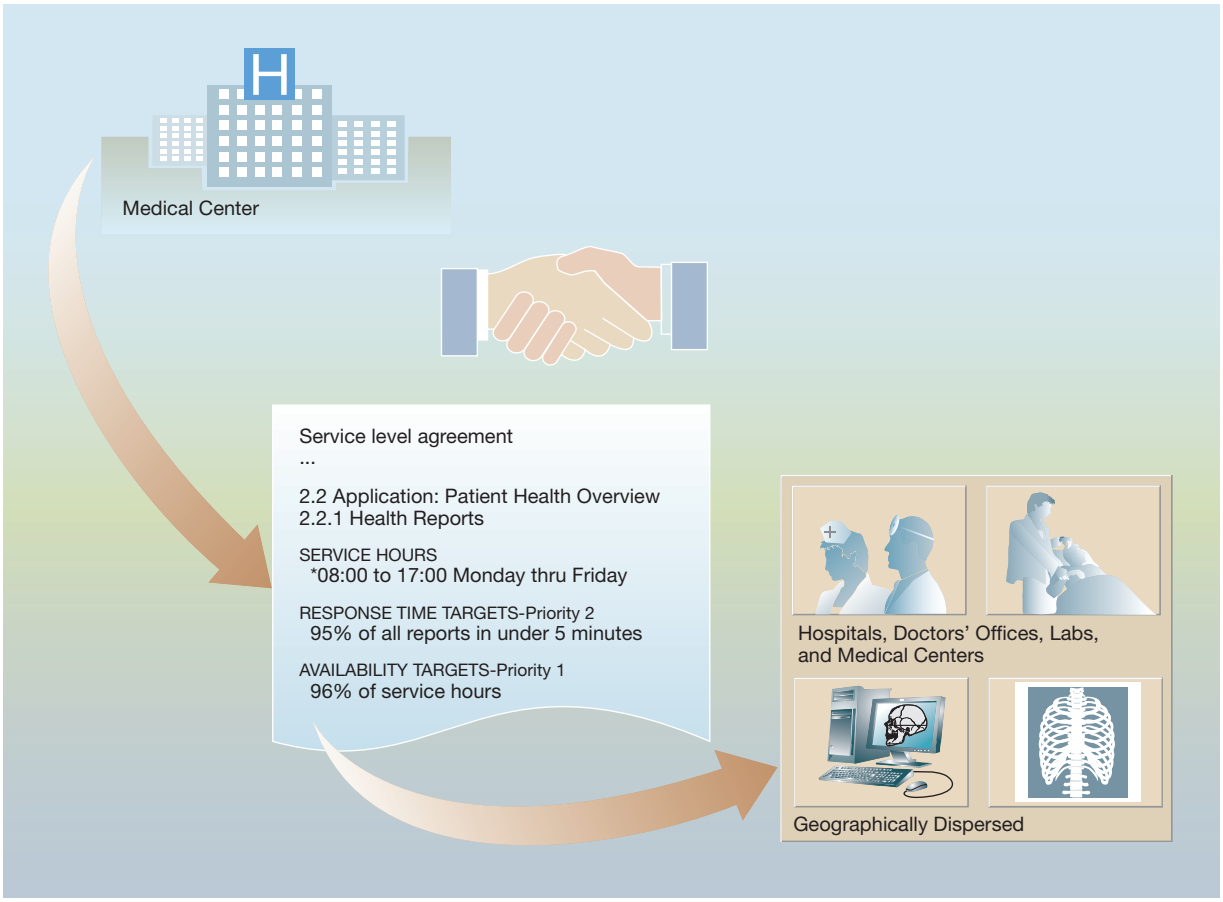
Figure 2 uses the Patient Health Overview scenario introduced earlier to clarify the concepts of SLA, terms and conditions, and policy. In this example, the SLA contains two sets of terms and conditions, or a policy with two goals. First, there is a response-time goal of less than 5 minutes for 95 percent of all reports, and second, there is the availability goal of 96 percent of the stated Service Hours. These two policy goals capture the QoS requirements of the users of the Patient Health Overview application, and in particular, the generation of health reports using this application. Having come to an agreement on these two policy goals, the service provider is obligated to meet these goals or incur a penalty. More formally:

A policy is a prioritized set of criteria for making decisions that guides the operational behavior of a system.

A criterion is anything that can be used to help make a decision or come to a judgment. In the example of Figure 2, the criteria are the policy goals of availability and response time, together with the conditions under which those goals are applicable, such as the Service Hours. The grid services use these criteria to guide their decisions on how to provision information automatically in response to consumer demand. The final piece of the policy definition is *priority*. The criteria themselves are prioritized, so that trade-offs can be made when the system is unable to satisfy all goals due to limited resources. In our SLA example, the availability goal has a higher priority than the response-time goal; hence, the autonomic system would favor keeping the Health Reporting function running and accessible over ensuring most reports had a response time of less than 5 minutes. For example, the overhead of maintaining a recovery strategy that would satisfy the availability criterion may negatively affect the reporting subsystem, resulting in only 85 percent of the reports having a response time of less than 5 minutes.

Given these two high-level policy goals, expressed in terms very relevant to the users of the Patient Health Overview application, grid services are obligated to configure and manage data access as necessary to satisfy the users' goals. There may be several different ways to meet these goals, possibly involving several different services. To meet a response-time goal, for example, remote data might be cached locally, or data might be hosted on storage devices with appropriate performance characteristics (e.g., striped across multiple disks for parallel access), or network bandwidth might be increased to reduce delays. Note that satisfying these goals is not up to the information infrastructure alone. Other grid services that monitor for policy

Figure 2 A service level agreement for the Patient Health Overview application



compliance must detect potential violations and choose the appropriate actions, then orchestrate the response by calling the necessary services. In this case, a workload management service might detect the exception and drive the need to improve response time.

Because a patient's health overview is an integrated view of medical records that may be scattered across various hospitals, doctors' offices, and medical centers, response time for the Patient Health Overview application could be unacceptable due to certain congested segments of the network. The workload management service might conclude that the only viable way of improving response time would be to cache some or all of the data to avoid the congested segments, and invoke the Placement Management Service to decide which data should be cached and where. Given the response-time goal of 5 minutes, the Placement Management Service could decide to introduce a replica or cache close to the Patient Health Overview application. It could then invoke the appropriate service to configure and manage the replica or cache in a manner consistent with the response-time policy goal.

In summary, an autonomic grid will be driven by policies and SLAs. It will require the collaboration of multiple grid services, some part of the core information infrastructure, others part of the general grid environment. Cooperation in this autonomic environment will have a number of benefits for our information infrastructure, including:

- More elastic data repositories, that is, database clusters that can grow and shrink automatically in response to demand

- Improved access to data through caching or replication, automatically determined based on access patterns, locality of consumers relative to providers, and locality of data relative to the processing of the data
- More transparent access to heterogeneous, dispersed data sources (because automatic data caching hides the distribution and heterogeneity)

Autonomic management is essential to maintaining the transparencies that we need for our information infrastructure.

A scenario using the Information Infrastructure Services. In this section we present a scenario using several information infrastructure services, to give a better understanding of how we envision these services working and interacting. We focus on the critical Data Services and Placement Management Services, which are at the heart of our information infrastructure. We first introduce the notion of *data equivalence*, which ties these two services together, then comment briefly on the workings of both services, and conclude this section with an illustration of how the services work together.

Data Equivalence: The previous section provided an example of how a Placement Management Service could help realize response-time policy goals by optimizing the location of data in the distributed system. The Placement Management Service could also help achieve availability policies for information such as those of hospitals that want their records available "24×7," by creating and maintaining multiple copies of the information on discrete physical machines. However, this will only help if the Data Service that answers requests for information knows about the copies and knows that they are sufficient to answer a particular information request.

Thus, the Data Service needs a way to recognize equivalent data sets and a means for choosing between equivalent data sets according to user policies. Likewise, the Placement Management Service needs to be able to create equivalent data sets that the Data Service will use. *Data equivalence* is the knowledge by the information infrastructure services that more than one copy of a data set exists and can be used as an alternative for satisfying a data query under a given set of runtime conditions and policies. Examples of data sets include a set of rows resulting from the execution of a query, a subset of a table, or a document. To illustrate data equivalence, assume there are two servers with a price list that is *usable* for a given e-commerce application. It could be that each price list is an exact copy of the other and that they are updated simultaneously, or one might be the master price list and the second could be a subset maintained by replication that is sufficient for this e-commerce application. Data equivalence applies to the scope of a query and does not imply content equivalence. As another example, imagine the PATIENTS table contains one million records, while another table PATIENT8 contains a single record for the patient with an ID of '8'. Despite the obvious differences between these two tables, these two data sets are equivalent for the query "where PATIENTID = '8'." As a final example, tolerance for using an out-of-date copy of the data would be specified as a policy of the application requesting the data. For an application willing to accept data less than an hour old, a cached copy of a table that was last refreshed (updated from the master version) 58 minutes ago is equivalent to the master, regardless of how many updates have happened at the master in the meantime.

This notion of equivalence shared by the Data Service and the Placement Management Service allows us to support location and name transparency. Applications do not need to modify their queries or connect to a different database to benefit from data placement. If the Placement Management Service creates new copies of subsets of the data, they are registered with the Meta-data Service, and applications will automatically benefit from these subsets without having to be recompiled because the Data Service uses the meta-data and discovery services provided by the overall infrastructure to find and exploit data equivalences.

Data Service: The Data Service is, first and foremost, a provider of information. It takes requests coming from applications (or from other services), and returns the information requested. Different Data Services may specialize in different types of request. One, for example, may handle only XQuery,[7] another may handle a subset of structured query language (SQL), a third may only handle files. A simple Data Service may encapsulate a single data store; whereas, a more powerful Data Service may be able to access any of several, and a yet more powerful Data Service may integrate multiple other Data Services, providing a uniform schema over all of them. We anticipate that many, if not most, Data Services will have their own local storage, which could hold some or all of the data to which they provide access and could also be

used for processing requests or as a target for caching data.

The Data Service works with other services to provide the transparencies mentioned earlier, across the set of information that it provides and by means of the interface that it provides. Some will be able to provide more transparency than others. Consider a very simple Data Service: a local file system. This service can provide only limited transparency; typically, users would be aware of exactly what operating system they are using and maybe even what disk drive their file is on (think of Microsoft Windows**). However, a distributed file system such as the Andrew File System (AFS*) may provide much greater transparency by providing a single namespace for files from several different systems, and, with the use of aliasing, even name transparency may be achieved. Likewise, a simple Data Service that encapsulates a single relational database by means of an SQL interface can be easily imagined, and again, provides limited transparency: users must be aware of which database system and even sometimes which version they are using, to which database name they should connect, and which client they must have installed. Yet, once the user is connected, the relational database provides most of the transparencies—within its limited domain. The Meta-data Service can be implemented as the relational catalog, and the Data Service itself as the relational database management system (RDBMS); views can be used to create logical domains to support name transparency; policies are met by self-tuning or (not so autonomically) by adding resources.

At the other extreme, a powerful Data Service providing information from a dynamically growing (and shrinking) set of data sources needs much assistance to maintain the transparencies. A more powerful Meta-data Service will be needed to provide mappings between sources and the unified schema; Caching, Replication, and Placement Management Services may be needed to meet policy goals, as well as a Workload Management Service to monitor compliance and then call the appropriate service—or that could be built into the Data Service or the Placement Management Service (different implementations for each of these are clearly possible; in the section, "Implementing an information infrastructure for the grid: Today," we will look at concrete implementations as they appear today in IBM's products).

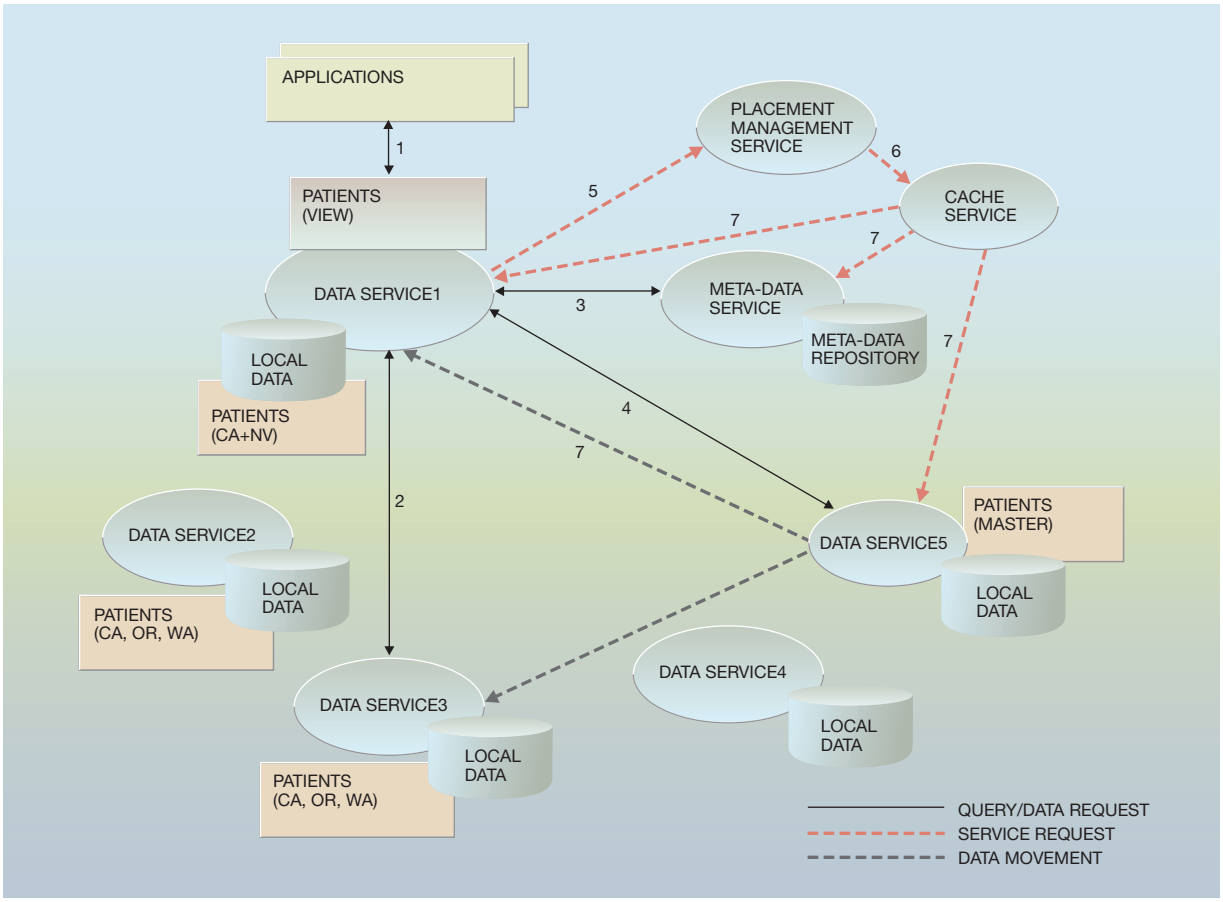
Placement Management: The Placement Management Service is concerned with the location and movement of data within the information infrastructure, to deliver a QoS that meets the policies of an SLA. Placement management functions address the need for improved response time by reducing network costs in remote geographies, say, or by precomputing queries into an equivalent data set. They also support high availability (by allowing the use of an equivalent table when a primary source is unavailable) and increased scalability (by, say, offloading a server before it gets saturated and distributing its data to a pool of perhaps less expensive servers). Workload characteristics, system and user policies, available hardware and software resources, and security considerations determine the type of placement:

1. Caching (on-demand transient copy),
2. Replication (synchronized long-lived copy),
3. Extracting and transforming (on demand copy, typically long-lived, possibly with transformations),
4. Federation (access in place, no copy), or
5. Archiving (move to long-term storage).

The difference between a cache and a replica is subtle and needs some explanation. The creation of a cache is primarily an optimization decision; a cache is a transient copy by nature, and applications should never be aware of or depend on the existence of a cache beyond their QoS requirements. The creation of a replica is primarily an availability decision; a replica is generally long-lived and static, and applications can be aware of its existence. However, replication technologies also provide data movement mechanisms that may be used for a Cache Service, such as changed data capture. Other implementations of a Cache Service that do not use replication are also possible, of course, and, likewise, if we had a pre-existing caching capability, we might be able to use it to build replication. This independence is one reason we believe there should be separate services for these tasks.

Figure 3 illustrates the relationship between the Placement Management Service and the Data Service in our grid information infrastructure. The Placement Management Service may cooperate with several Data Service instances. In the figure, the master copy for a PATIENTS table resides in the local store for Data Service5. (Note that because our infrastructure upholds heterogeneity transparency, we do not know what the actual system is that provides the data. It might be a DB2* for z/OS* database, IMS*, an XML

Figure 3 Patient Health Overview application: Interactions among services



table, or an SAP** system. In fact, each of the Data Services shown might be a different system "under the covers.") At the start of the action, a partial copy of the PATIENTS table (for patients in three western states—California, Oregon, and Washington) resides in Data Service3, and requests coming in for patients from those states from the applications to Data Service1 (1) are being routed to Data Service3 (2) (we assume that Data Service1 has previously discovered that Data Service3 can satisfy these requests).

Then Data Service1 receives a request for data for patients from Nevada and California. Nevada data is not available at Date Service3, so Data Service1 asks the Meta-data Service where it can go for the answer (3). The Meta-data Service provides Data Service1 with a list of Data Services. The list provides patient data, along with sufficient information on the granularity and currency of each Data Service so that Data Service1 can determine whether a Data Service is equivalent to the request in its query. In this case, Data Service1 finds that the best alternative is to go to the master copy of the PATIENTS table at Data Service5 (4). (It could have chosen to "union" data on patients in Nevada from Data Service2 with California data from Data Service3, but that requires multiple requests, and perhaps the currency of the data at Data Service2 is insufficient for this application.)

However, Data Service5 is heavily loaded and soon cannot keep up with the requests coming in from Data Service1, which is now in danger of violating

its SLA with its application. Therefore, Data Service1 might ask the Placement Management Service for help (5). The Placement Manager analyzes the requests with which Data Service1 is dealing and finds that most of them are for patients in California or Nevada. It therefore requests the Cache Service to cache data for those states at Data Service1 (6). The Cache Service works with Data Service5, Data Service1, and the Meta-data Service to create and populate the cache (7), possibly using a Replication Service not shown, and to record the information on the new cache in the Meta-data Repository. Once the initial population of the cache is complete, Data Service1 can satisy its application locally.

Note that there is a close collaboration between the Data Service, which makes routing decisions (selects the source or equivalent data set) and the Placement Management Service. The latter analyzes the workload and acts according to the placement policy rules. The Placement Management Service might, for example, automatically set up replication or archive data. Data placement can be used to support multiple distributed scenarios. For example, a medical group with several different locations might have a centralized computing facility. To offload the centralized server allowing better scalability, they could deploy several Data Services and Cache Services, one at each site, to act as mid-tier caches for data accessed from both their facility and those of other medical groups on the grid.

In this section, we have presented our vision for an information infrastructure for the grid. The infrastructure we envision consists of a set of core information services that cooperate to maintain an important set of transparencies and meet a set of policy objectives. They work with and depend on other grid services as well. Many implementations of each service are possible. In the next two sections, we discuss an implementation for the infrastructure described earlier. The first section describes the current state of our implementation as manifested in our products. We present the various services, as well as our overall autonomic capability, and show which transparencies are supported. Although our infrastructure already provides rich capabilities for the grid, there are still areas that need work before our vision is fully realized. In the following section, we first discuss the evolution that is needed to make the infrastructure more dynamic and scalable, and then revisit a few key services to show how they may change moving forward.

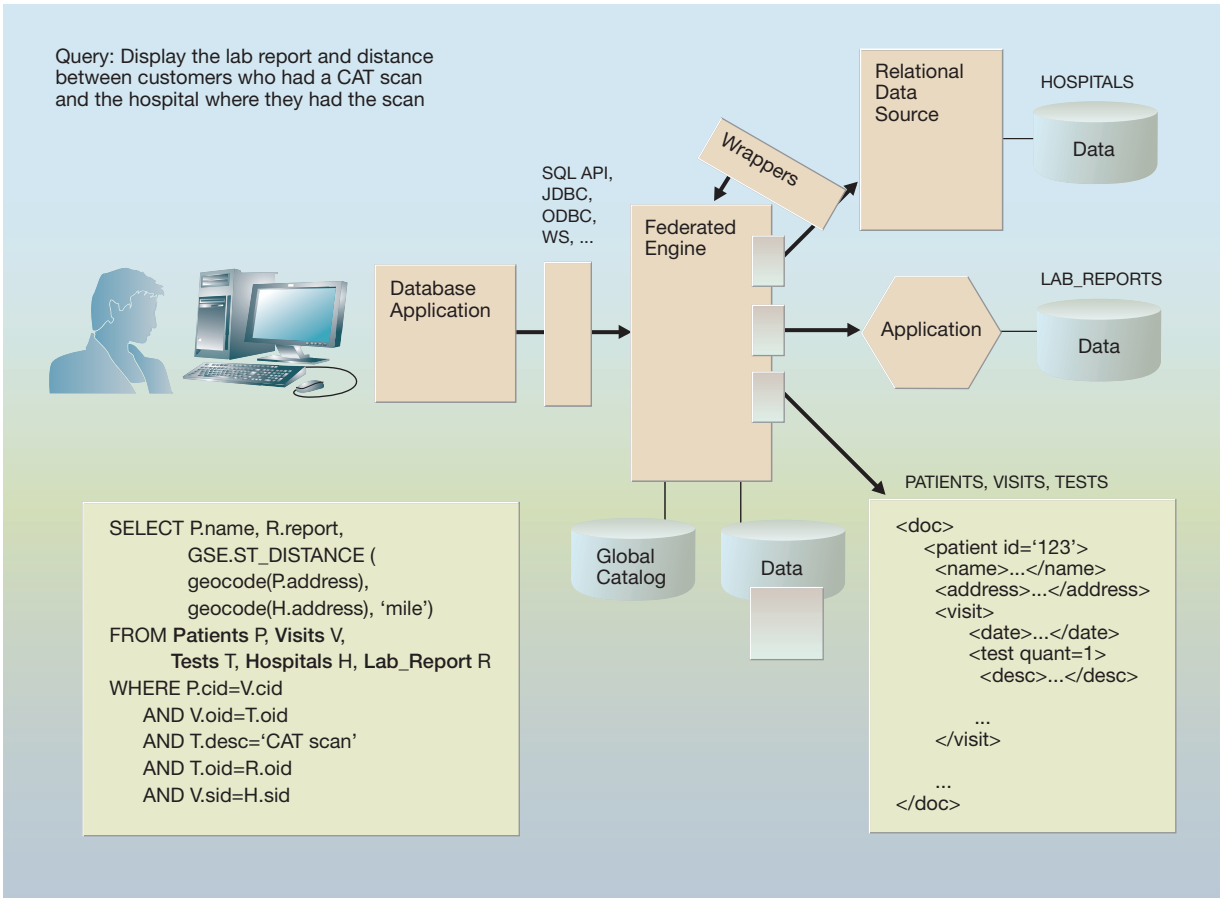
Implementing an information infrastructure for the grid: Today

IBM has long been concerned with providing information virtualization. Products such as DB2 DataJoiner*[8] and research projects such as Garlic[9] pioneered federated database technology nearly ten years ago. Today, DB2 Information Integrator*[10] is taking further steps towards full virtualization, bringing together federation, replication, transformation, publish, and search technologies into a powerful platform for information integration. Many of the capabilities we imagine for the grid information infrastructure exist at least in part in DB2 Information Integrator and other DB2 products today, though not always accessible via services—yet.

We will describe DB2 Information Integrator's capabilities using the framework of the service-oriented architecture introduced in the previous section. We will use the term "service" loosely, here, to indicate the relevant capability, and indicate which of these capabilities is actually accessible as (Web) services in the text. All are accessible via programming interfaces, so they can be called from another program, at least. In the following, we focus on each of the services in turn, starting with the Data Service and the placement-related services that are at the core of our infrastructure.

The Data Service. A key component of DB2 Information Integrator is its *federation engine*, which we will look to as our Data Service. This federation engine is a query-processing engine (built on the DB2 Universal Database* technology) that provides transparent access to a number of heterogeneous, distributed data sources. It provides location and heterogeneity transparency, and even some amount of replication transparency today. Even though the data sources are distributed, the federated system looks to the application developer like a regular (relational) database management system. Users can run queries to access data from multiple sources, joining and restricting, aggregating and analyzing the data at will, with the full power of SQL (including SQL/XML[11]). While access to individual data sources is also provided by Java** Database Connectivity (JDBC**), Open Database Connectivity (ODBC), Microsoft** ActiveX Data Objects (ADO), and so forth, federated systems allow correlation of values in different sources (e.g., cross-source joins), without any work in the application. Users can also update data for some sources if they have the appropriate permissions at the sources. Yet the data sources in a federated system need not be databases at all, but

Figure 4 A federated query performed by DB2 Information Integrator



in fact could be anything ranging from sensors to flat files to application programs to XML documents, and so on. The data in these heterogeneous sources are modeled as tables, and specialized capabilities of the sources are modeled as functions.

A typical query in a federated system architecture is illustrated in Figure 4. Applications can use any supported interface (including ODBC, JDBC, or a Web services client) to interact with DB2 Information Integrator. When an application submits a query to DB2 Information Integrator, the federated engine identifies the relevant data sources and develops a query execution plan for obtaining the requested data. The plan typically breaks the original query into fragments that represent work to be delegated to individual data sources, as well as additional processing to be performed by the federated engine to further filter, aggregate, or merge the data. (For the example in Figure 4, the federated engine would have to calculate the distance after the join.) The ability of the federated engine to further process data received from sources allows applications to take advantage of the full power of the query language and any functions defined at the engine, even if some of the information that they request comes from data sources with little or no native query-processing capability, such as simple text files. The federated engine has a local data store to cache data or query results, if desired, as well as to provide temporary storage for partial results during query processing.

In Figure 4, the federated engine accesses diverse data sources that are shown on the right: a traditional database system such as Oracle, a specialized application, such as the Documentum** document man-

agement software, and an XML file. A single federated query can perform a join between hospital data stored in Oracle, patient and visit information maintained in the XML file, and test results in the Documentum data source.

The federated engine communicates with the data sources by means of wrappers. A wrapper is a piece of code, packaged as a shared library, which can be loaded dynamically by the federated server when needed. Often, a single wrapper is capable of accessing several data sources, as long as they share a common or similar application programming interface (API). The wrapper serves to translate between the federated engine's data model and internal data structures and the interfaces and data models of the data source. Once wrappers are written, applications can seamlessly integrate data from relational databases, application programs, and even XML data sources, without consideration of the details of data formats and programming interfaces in each source.[12] Hence, the wrappers provide heterogeneity transparency for our Data Service.

DB2 Information Integrator provides a wide range of wrappers to structured and unstructured sources. One important wrapper for our service-oriented grid infrastructure is a wrapper that allows data to be incorporated from Web services. As the grid standards for data access (DAIS[13]) solidify, the Web services wrapper will be used as the basis for a grid services wrapper. DB2 Information Integrator also includes a Web services provider, which allows the federated engine to be invoked via a Web service. Thus our Data Service is ready to plug into a service-oriented information infrastructure today, using Web services. In fact, all of the DB2 database products are similarly enabled. As more and more commercial data sources provide similar capabilities, it will be easy for our Data Service to reach out to them transparently.

A key feature of DB2 Universal Database, and one that is inherited by DB2 Information Integrator, is the ability to define materialized views, called Materialized Query Tables (MQTs) in DB2. The query compiler understands these views and can automatically substitute them for use in a query in place of the base tables, providing a measure of replication transparency for our Data Service. (In essence, the view definitions serve as data equivalences for the query-processing engine). Materialized views are particularly powerful when used by our federation engine to materialize distributed, heterogeneous data, as they can greatly speed up query processing. Materialized views can be simple selections or projections on a single base table, or they can be complex aggregations and joins.

The Placement Management Service. The 2004 release of DB2 Information Integrator includes our first implementation of some intelligence around data placement. The *Design Advisor*[14] has, as one of its features, an MQT advisor that can recommend what data to materialize, given a workload of queries and priorities. In its first release, the Design Advisor will only give advice; but it will provide all the commands needed for a human (or program) to implement that advice. Hence, we can build a Placement Management Service based on this intelligence, which, given a workload of queries, will be able to recommend and create materialized views that the federated engine will recognize and use.

The Cache Service. It is possible to use MQTs to build a simple declarative cache for individual tables by creating an MQT defined as a simple select-project query on that table and maintaining the MQT by using replication. For example, assume PATIENTS is the local nickname for a table at a remote server. The SQL statement 'create table CACHEPATIENTS as select * from PATIENTS' defines an MQT for caching rows from that remote table. When compiling a query that references the remote table PATIENTS, the query processor looks for a materialized version that could be substituted based on existing MQT definitions. If an application issues the query, "select name from patients where name like 'Bou%'," this query will be executed against the CACHEPATIENTS MQT defined above. No remote access is required. Of course, any MQT can be used as a cache in the same way, but when the MQT is a materialization of an aggregation or a join over several source tables (especially tables from different sources), it is maintained by doing a full extract and load. (Simple aggregations from a single source can also be maintained via replication if desired.)

To use an MQT as described, the application must indicate a willingness to tolerate potentially stale data. In DB2 Information Integrator, the policy on staleness is specified with the REFRESH AGE parameter. REFRESH AGE is a binary parameter: an application either tolerates arbitrary data staleness, or none. It is the responsibility of the system administrator to ensure that the MQT is properly loaded with data at the right time as per the definition (possibly using replication as described above). Data

writes against MQTs are always redirected to the data source. Since refresh of the cached data is asynchronous, subsequent reads in the same transaction may not see their own updates. The mechanism for routing a query to an MQT, or substituting a table or nickname specified in this query with an MQT, is a compile time decision. That decision is made by comparing the cost of executing the query against the actual data source table or tables to the cost of executing the query using the MQT. As such, it does not actually take into account availability of the data source. In other words, if the query would execute faster against the actual data source, that is the execution strategy which will be chosen. If the source happens to be down at that moment, the query will fail, even though a local copy exists. Still, for most situations, local data will be preferred; therefore, MQTs do provide for more availability as well as better response times and scalability. Another issue that arises due to the compile-time routing decision is that the query engine cannot always prove that the cache has all the rows needed for the query. For example if a query uses parameter markers to select a subset of the table rows and the MQT is a proper subset of the table, the query engine does not have sufficient information to decide, and so must conservatively use the actual data source.

To recapitulate, today DB2 Information Integrator includes the ability to cache individual tables or complex query results in MQTs. Routing is done at compile time and does not take into account changes in data sources, such as availability. Caches are declaratively loaded and maintained either by change-capture replication or by full refresh. Writes are transparent to the cache, meaning that applications might not see their own update if a subsequent read is directed to the MQT.

The Replication Service. DB2 Information Integrator includes a replication engine (formerly DB2 Data Propagator). This replication capability addresses scenarios where data needs to be provisioned to a variety of applications, often across geographies, allowing users to maintain copies of the data across heterogeneous database systems. Several topologies are supported; for example: data distribution from a primary server to several distributed servers; data consolidation from distributed servers to a primary server; master/slave configuration, where a named master server prevails in the case of conflicts; and peer-to-peer, where servers can each update the same data. Changes to a database are captured from the database recovery log (or by using triggers for non-DB2 databases) and staged into relational tables, from which they are applied to one or several replicas (by using federation for non-DB2 tables). This architecture has interesting advantages: staging the data into relational tables allows transformations via SQL by using database triggers, stored procedures, or SQL statements, and the data changes only need to be inserted once into the staging area for delivery to several targets.

In 2004, a high throughput, low-latency replication engine joined the set of replication alternatives, also as part of DB2 Information Integrator. Initially limited to the DB2 database family, this new engine also captures changes from the database recovery log and puts the changes onto a queue for delivery to the target, using point-to-point messaging. At the target, the changes received from the queue can be applied in parallel, using multiple agents. Conflict detection and resolution are also much improved over earlier technology.

In addition, to address disaster recovery scenarios IBM is developing High Availability Data Replication (HADR), a feature that is already available for IBM Informix* Dynamic Server databases. This replication solution addresses the need to maintain a standby server for failover in case of a disaster when the primary server is disabled. With HADR, the database recovery log records are sent to the standby server via TCP/IP (Transmission Control Protocol/Internet Protocol) when a transaction is committed. As log pages are received at the secondary server, database recovery logic is executed to apply the changes and maintain consistency. Synchronization between the two servers can be ensured. The secondary database must have the same physical structure as the primary database, and because it is performing recovery on the database, it cannot be used for queries until a failover occurs. HADR tolerates intermittent loss of connectivity between the two servers. When the connection is lost, the primary server stops sending log records to the secondary server until the connection is restored, at which point the secondary server requests missed log records from the primary server, and a resynchronization takes place.

The Change-Publish Service. Our Change-Publish Service uses the same framework as the new, queue-based replication previously described. This publishing function makes it possible for an application to subscribe to data changes that will be delivered as XML messages over an MQSeries* message queue.

The use of the replication log capture mechanism ensures that a change is delivered only after it has been committed to the database. The changes are captured and the messages sent asynchronously, so that publishing does not impact the response time of the application committing changes to the database. A subscriber can be connected to the server intermittently: changes will be accumulated in the MQSeries queue while a connection is unavailable. Changes from a variety of sources including the DB2 family, but also IMS and VSAM*, can be captured.

The Meta-data Service. Today, the federated engine's catalog stores the operational meta-data needed by the various services described above. By querying the catalogs, information about remote sources, tables, caches, and subscriptions can be retrieved. New in 2004, we are adding support for an XML Meta-data Repository (based on the XML Registry[15]) as well, where, for example, XML artifacts used in application development may be stored and queried.

The Registration Service. To register a new object to the DB2 Information Integrator, new entries must be placed in the Meta-data Repository. These entries can be made via a graphical user interface and saved in scripts for later replay. Programmatic interfaces are also available for many object types. Typical objects that might be registered include wrappers (the code modules that connect the federation engine to a particular type of data source), sources themselves, data sets from a source (where a data set is a unit of data that will be modeled as a table to the federated system), replication subscriptions, and so on.

The Discovery Service. Today, we have only the rudiments of a Discovery Service—but even those rudiments may be lifesavers for users of the federation engine. Today our discovery capability is only available through the control center, the graphical user interface for administering the Data Service, and only for tables and data source instances. The user has to explicitly specify what type of source to look for; the Discovery module returns instances of that source type and data sets at that source. For example, if the user indicates an interest in XML files, the Discovery module can present the user with a list of possible files. Once the user picks a file of interest, Discovery can parse that file, and pick out the element types and their definitions that could be modeled as tables to the federated engine.[16]

Summary: The information infrastructure today. To summarize, DB2 Information Integrator already provides many of the capabilities needed for our information infrastructure, albeit not all are available as services today. The federated engine (our Data Service) provides heterogeneity transparency and location transparency. It understands some types of data equivalences, and hence also supports a degree of replication transparency. While not fully autonomic as yet, there is the basis for a Placement Management Service (the Design Advisor), and the means to accomplish placement, namely, replication and caching. Publishing capability is being built on the change-capture capability used for replication. Substantial meta-data is captured in catalogs that can be queried, and new objects can be registered or discovered, causing entries to be made in the same catalogs.

Powerful though this infrastructure is, it still needs improvement. The infrastructure described does not yet provide all the transparencies we believe are needed. One key limitation is that there is no name transparency; applications have to explicitly name the individual data sets (modeled as tables) that they wish to query (unless an administrator defines views to cover them—and even then, the views must be explicitly named and maintained). Thus, applications (or view definitions) may need to change when the set of data sources changes. For example, if a new hospital joins our grid, a query looking for patients does not find this hospital's data unless the query is modified to look at the new hospital's sources. Likewise, failure or removal of data sources along with the data sets they provide may require rewriting queries to omit the missing site's sources so that the queries do not fail. Another issue has to deal with ease of registration: data sources must be explicitly registered to the federated engine, along with their wrappers; this constrains the frequency of source addition and removal, and hence the system's ability to exploit new data in a timely fashion. We have already mentioned the need for a services interface (Web today, grid tomorrow) for some of the implementations just described. Some of the individual service implementations also need more function. In the next section, we look at how we expect the infrastructure to evolve to better support the needs of the grid.

The information infrastructure in the future

To create the ideal information infrastructure for the grid, the implementation just described will need to

be enhanced in an number of ways. All services need to evolve to be more directly policy-based and autonomic. They need to be made more dynamic. Although we will not give details here, all the services of the information infrastructure need to exploit the general grid services for policy, billing, security, and so on. Many also need additional work to support the scaling required by large grid environments and to allow more heterogeneity.

One key area that must be enhanced to ease administration and increase performance is support for data equivalence. This will require changes to a number of our services. The Discovery Services, Meta-data Services, Data Services, and Placement Management Services will all need to cope with a richer set of equivalences. Discovery Services will need to find and recognize them, Meta-data Services will need to represent them, understand their characteristics and allow them to be examined, Placement Management Services will need to maintain them dynamically and automatically, and Data Services will need to choose which one to use, also dynamically and automatically.

Evolution of the Data Service. The key areas of focus for the Data Service will be ease of administration (autonomic configuration and management), enhanced performance and scalability, and exploitation of a richer set of data equivalences. In all of these areas, the Data Service will rely on other services to help. The Discovery Service will allow the Data Service to automatically find sources for a query, for example. The Placement Management Service will move or copy data to improve query performance or scalability and will create and maintain the data equivalences that the Data Service relies on. Enhancements to the Data Service federation engine itself will give it more ways to execute queries including more join methods, more asynchrony, and the ability to exploit the parallel potential of a grid for query processing (parallelism transparency). Most important, the Data Service will need to be extended to recognize more equivalences, being sensitive to more characteristics, such as distance in terms of access time, online status, and the user's requirements for equivalence (based on more complex queries).

It will also be critical for virtualization to make runtime decisions about which data set to use. Research is underway for augmenting the routing mechanism of the Data Service with runtime decisions. Examples of policies and workloads that require a runtime routing decision are:

- Data staleness policies that require data to be up to date within a small time interval (the age of the data needs to be verified when the query is executed)
- Tolerance to partial results (e.g., getting any available information on the current patient, because some information is better than none)
- Use of partial dynamic caches that are loaded on demand, based on workload (the contents of the cache need to be verified when the query is executed)
- Queries with dynamic SQL and parameter markers where the match between query and cache cannot be determined until runtime, even for statically declared caches
- Selection of a data source based on the result of a user-defined function (UDF) invocation
- Dynamic negotiation of access speeds and throughputs with autonomous sources

We want the federation engine to be able to pick an alternate data source, as per the policies, if at runtime the source selected during query compilation is not available or no longer adequate for fulfilling the requested QoS.

A method proposed in Reference 17 consists of incorporating a *probe*, in the form of an SQL operation, as part of the query access plan, to test whether a particular data set has the properties required for the query. For example, suppose a query searches for information on some patients in intensive care, and there is a policy that when information for a particular hospital unit is in a cache, the information for all patients of this unit is also in the cache. Then testing that a single row is present in one equivalent table (a copy of hospital unit information) can be sufficient to determine that an entire group of related rows (intensive care patients) is also present in other equivalent tables (those related to patients). The system ensures the integrity of the overall cache according to a policy set by an administrator. To incorporate this probe into the query access plan at compile time, the SQL query compiler generates two data access plans; one is the plan that accesses the actual objects specified in the query (this would normally be the only plan generated for the query); the second plan is a copy of the first one, where each object has been substituted with an equivalent (local) object, if one qualified. The two plans are retained for the final executable and combined into a

single query access plan, called a *Janus* plan, where they are connected by the probe. At runtime, the probe is executed first, to decide which of the two plans is executed.

An alternative approach that we are exploring is to use a *metawrapper* module that resides between the query optimizer and the wrappers to the equivalent (remote) tables. The metawrapper mediates in the optimization process to allow cost-based selection of the best equivalent table based upon the information known at compile time. (We assume that either standard MQT routing or the probe approach is used to deal with local caches.) At runtime, the metawrapper can also change the equivalent-table selection to adapt to changes in the availability, staleness, and speeds of the equivalent tables, or to take run-time query parameters such as host variables and parameter markers into consideration.

Placement management evolution. The Placement Management Service must also grow significantly to enable the future information infrastructure. Today, the Placement Management Service does static analysis of a workload and recommends a set of views to be materialized at the Data Service federation engine. The recommendations do not cover latency characteristics or other desired properties of a materialized view, only the set of data to be copied. Those recommendations must then be executed by an administrator.

In the future, this will change in four ways. First, the Placement Management Service will be invoked automatically when policies are violated (for example, when response times are not within the limits set). Second, the Placement Management Service will be able to recommend materializations at other locations, perhaps at one of the sources (a simple Data Service, not the federation engine). For example, it might be wise to cache data from assorted hospitals about a new epidemic at the CDC, even though queries about symptoms and cases may be coming from Data Services in regional research labs. The alternative, caching that data at each research lab, could be expensive in terms of storage and maintenance. Third, the Placement Management Service will be able to define characteristics of the recommended copies, such as the type of cache to make, latency of the data, and so on. Fourth, the Placement Management Service will be able to invoke other services to automatically create and maintain the recommended caches or replicas.

Cache Service evolution. Today, the Cache Service supports a single type of static (or declarative) cache, but in the future, we foresee many types of cache being needed. We believe the Cache Service (or various Cache Service instances) will support several levels of materialization, each level characterized by the amount of precomputation provided, and consequently, by its reusability (or generality). For example, at the top, materializing the result of an SQL query or a Web service requires much precomputation (making queries against the cache exceptionally quickly) but might be reusable only if the exact query is reissued by the same application, with the same parameters. This full-query caching can be optimal when results are expensive to retrieve and queries repetitive. At the bottom, caching a subset of a table from a single data source is most general but requires recomputing each query, albeit over now-local data. This single-table cache can be optimal when queries are varied and mostly key-based data retrievals. The materialization of a view is a trade-off between these two extremes, and it will be possible to define such view materializations on top of other materializations. A task of the Placement Management Service will be to determine which types of cache will be of most benefit for a given set of workload characteristics and policies.

We would also like to support more policies for cache load, maintenance, and data integrity. Cache load policies include:

- *Declarative*—Data is preloaded as per a static definition; that is, by using a select statement, such as 'select name, condition from patients'. This is what our information infrastructure supports today.
- *On Demand*—Data is loaded when first used. This is common in page-based (operating) systems or on the Web where there are naming conventions for what constitutes a page, but harder to do for a service that must cache data from multiple data sources with no ready equivalent of a page and no agreed-upon naming conventions to define what data are cached. Semantic constraints might be used to identify a group of related data to be prefetched into the cache as a single unit when a datum of the group is first used.
- *Query Results*—Query results are materialized, also on demand, and reused when the same query is re-executed under the same conditions and environment, which generally means by the same user, with the same arguments and the same isolation level, as, for example, in DB2 Query Patroller V7.[18] But the cache can also be implemented so that it

can be used to satisfy queries that return a subset of a previously executed query. Because the entire query is cached, it is easy to determine what data are in the cache for routing decisions. This type of cache is particularly useful to support asynchronous delivery and delivery to third parties—both important concepts for grid computing.

Cache maintenance policies include:

- *Time-To-Live (TTL)*—Keep data in the cache for a preconfigured amount of time, and then invalidate the data (does not detect changes to the original data source version of the data). We use a modified form of this today when we blindly refresh a cache at certain time intervals.
- *Change-Invalidate*—Changes in the data source are detected and used to invalidate data in the cache. This is not supported today.
- *Change Replication*—Changes in the data source are captured and applied to the data in the cache. This can be supported for simple caches today.

Data integrity policies include:

- *Transparent Write*—Make changes directed against the federated Data Service at the data source, but not to the cache. If Change Replication (or TTL) is the maintenance policy, then the application cannot see its own updates immediately and must wait for the replication (or refresh) process to pick up the change from the source and apply it to the cache. This is supported today.
- *Write-Through*—Update the cache and the data source in the same transaction. This requires a two-phase commit between the federated Data Service and the data sources, which can be expensive.
- *Write-Deferred*—Update the cache, and asynchronously update the data source. This policy is also required for supporting updates in disconnected modes.

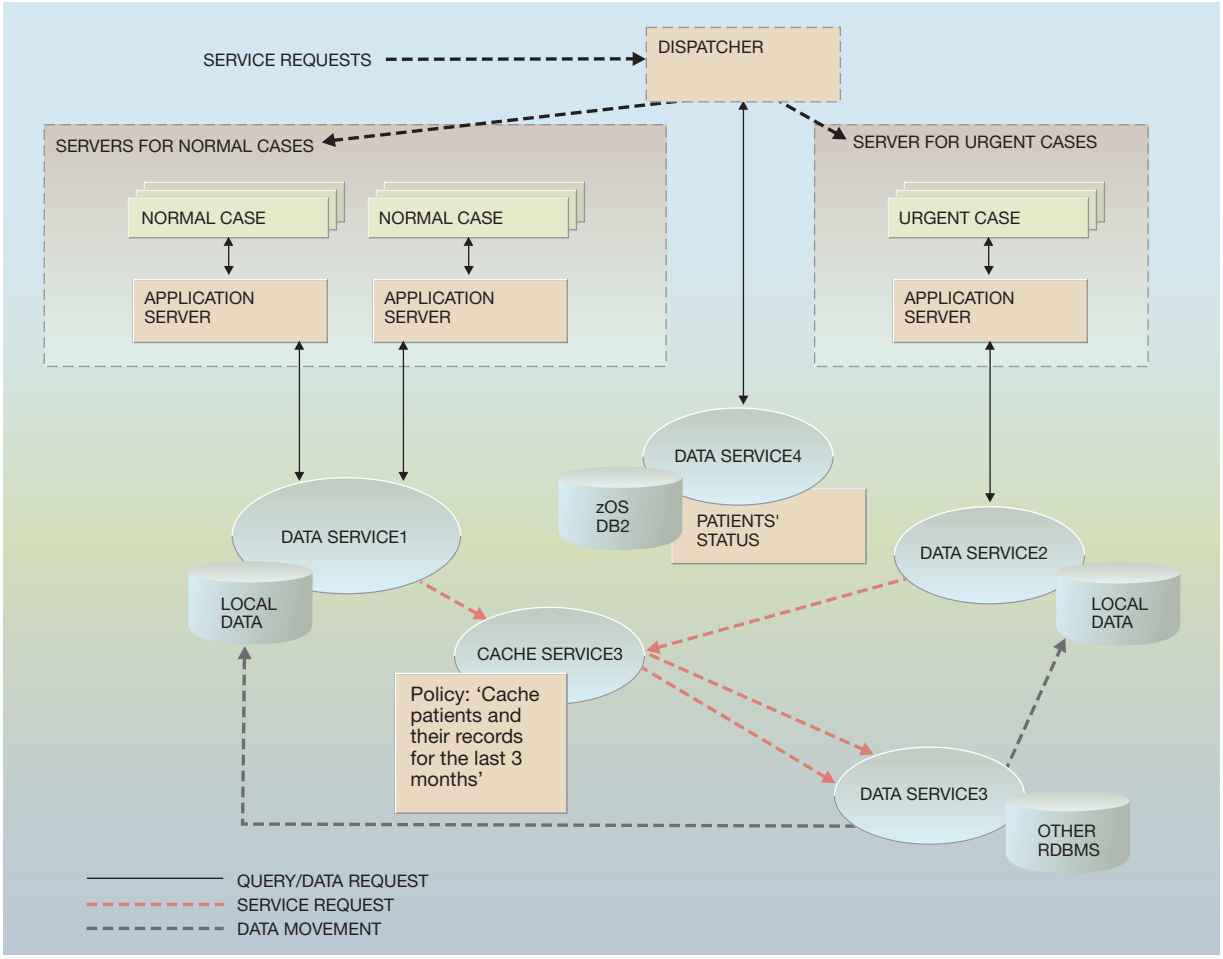
Our first priority for expansion is to support on demand caches. Coupling the ability to load caches dynamically with a Data Service that provides query processing over heterogeneous environments allows the information infrastructure to supply powerful mid-tier caches that provide for QoS for grid applications. Scalability will be achievable by distributing the query workload among a pool of Data Service instances based on data affinity. As each Data Service recognizes that it does not have data locally (a cache miss), it will request the Cache Service to provide the required data. This will effectively partition the cached data, improving both the probability for the data to be in the cache for the queries and the likelihood that the cache will stay small and within budget constraints. In contrast, dispatching transactions randomly between servers will generally result in loading the data required for all types of transactions into each cache, thus creating larger, poorly performing caches.

As an example, assume the following workload management decision for our medical grid: "Create a new Data Service dedicated to urgent care patients to improve QoS for these patients," where the existing Data Services are already servicing requests from all patients. Figure 5 illustrates this scenario in which the Workload Management Service sets up the new Data Service and invokes the Cache Service to configure a cache for this service. The new cache (on the right, at Data Service2) can be configured exactly like the existing caches at Data Service1. Specifically, with on demand cache loading, there is no need to explicitly declare that the new cache is for urgent care patients, only that it is enabled for caching patients. The query dispatcher can immediately start sending customer queries to this service based upon their status, which perhaps it looks up from a dispatching table (patient status) using Data Service4.

In summary, a Cache Service causes data to be cached in the Data Service's local store. This allows QoS goals to be met without explicit application intervention. In the future, the application's ability to tolerate stale data and the options for data movement will be recorded as policies. These policies will determine what type of copies the Placement Management Service creates and which copies the Data Service selects from the equivalent data sets known to the Meta-data Service. Future work will focus on dynamic or on demand caches first, as well as on looking at alternative policies for cache maintenance and data integrity.

Evolution of the Replication and Change-Publish Services. We noted above that replication and changed data capture and publish are highly related technologies: Replication copies portions of a database, often in a preplanned way, and provides synchronization across the copies at intervals over a long period; Change-Publish captures changed data, formats the data as an XML message, and puts the data on a queue for distribution to other data sources or applications.

Figure 5 Query dispatching and on demand data caching: An example



In future versions of the information infrastructure for the grid, we see these separate technologies and services being subsumed by a more general framework for data movement. This framework will provide a great deal of flexibility to a subscriber in terms of specifying rules for data of interest, transformations, dynamic generation of lists of consumers, and so on. It will support highly heterogeneous sources and targets, and meta-data about copies, subscriptions, publications, and so on, will be made available to the Meta-data Service to serve as the source of information on data equivalences for the Placement Management and Data Services.

In the envisioned framework, the data movement process is divided into four basic components: data publication, data subscription, event and data propagation, and event and data consumption.

Data publication occurs at the Data Service that acts as the logical source of data. The data publication rules may include whether and what data is to be published and information about the *publisher*. Data publication specifications include rules such as "publish this data only to subscribers ages 18 years and older," auditing, and nonrepudiation requirements.

Data subscription occurs at the Data Service that acts as the logical data target. A data subscription specifies a rule or a set of rules that match data of interest to the subscriber with information about the subscriber. Data subscription specifications include

rules such as "deliver data that is published only by certified publishers," auditing, and nonrepudiation requirements.

Event and data propagation rules define how to propagate events and data to a *consumer.* The consumer may be a client or another Data Service, such as a Broker. These rules include propagation protocols and auditing and retention requirements and may also include specifications of how published data is to be delivered and to whom.

Event and data consumption rules define how the consumer consumes events, changed data, and data. These rules include, for example, auditing and retention requirements. Rules are based on the underlying data model query language, such as SQL for relational data services. The composition of these components can support efficient, reliable, and asynchronous information dissemination in the information infrastructure for the grid.

IBM Almaden Research Center has used this framework to prototype a Grid Data Movement and Replication Service (GMR) based on the Globus Toolkit. GMR can replicate data between heterogeneous data sources, such as PostgreSQL to Oracle and vice versa. Currently GMR supports replication of SQL query result sets and files. As a proof of concept, the GMR is deployed by the Geosciences Network (GEON) Grid[19] at San Diego Supercomputing Center (SDSC) to replicate geosciences meta-data across geographically distributed GEON nodes.

These types of change-capture and event propagation features are invaluable in a grid environment. They can be used to provision data to applications, to maintain copies of data at a desired degree of currency, or to disseminate information, potentially triggering additional computations and events. Today, most replication engines work on database data only (including, however, nonrelational data), but some are already being extended to work with arbitrary data sources, including file systems and applications. These developments will greatly improve the virtualization provided by the grid's information infrastructure.

Meta-data, Registration, and Discovery Service Evolution. The Meta-data Service will have to change dramatically to support the needs of the information infrastructure in the future. It will need to store a much richer set of meta-data, including not only information on objects (such as tables, MQTs, and subscriptions) as it does today, but also information on the relationships among various objects, such as mappings from one schema to another, or information on transformations between objects. Likewise, it will need to provide a richer set of services or interfaces to its users, including services to explore meta-data and to create mappings and the means to realize those mappings (transformations). Because it is unlikely that all the meta-data for a grid will be stored in a single repository, the Meta-data Service will have to be distributed; that is, Meta-data Services, each managing a portion of the grid's meta-data, will have to cooperate to help users (or other services) work with meta-data.

Another major change we foresee in this area is the virtual disappearance of a data-specific registration service. In the future, we expect that the Discovery Service can autonomically discover information objects and data sources, as well as relationships among objects, transformations, and so on, recording its findings with the Meta-data Service. While Discovery will still be guidable by users, they should rarely if ever have to locate the data of interest themselves and explicitly register it.

Summary: The future information infrastructure. In this section, we describe the information infrastructure for the grid that we are working towards. The infrastructure will be an open, flexible, extensible set of services accessed through well-defined and standardized interfaces, the result of work described in the section on standards below. The information infrastructure we imagine will be policy-driven, and highly autonomic. Placement management will play an enormous role in ensuring that grid applications receive the QoS they require and will make use of greatly enhanced caching and data movement services to effect the necessary shifting of data. An even more powerful Data Service will exploit the equivalent data sets created by the Placement Management Service and will make more dynamic decisions about which copy to use, relying on the enhanced Discovery Service and Meta-data Service capabilities. This more dynamic and autonomic information infrastructure will use other standardized grid services and be used by them. In short, the information infrastructure for the grid will, in this future incarnation, support the transparencies identified earlier.

The role of standards

In order to fulfill the goals of our information infrastructure for the grid, in particular interoperabil-

ity, we must standardize grid-services interfaces. There is considerable activity towards standardizing various interfaces for the grid in the GGF.[1] The objective of GGF working groups is to produce documents describing best practices, technical specifications, user experiences, and implementation guidelines for distributed and grid-computing environments, with a particular emphasis on interoperability and technology reuse. In order to be independent of programming language, many of the GGF specifications describe interfaces in terms of WSDL (Web Services Description Language) and XML Schemas. The GGF Open Grid Services Architecture (OGSA) Working Group[2] provides a context for structuring grid components and for identifying appropriate Web-services interfaces between components. Various GGF groups work on describing the service-oriented interfaces for grid components in more detail. An OGSA Data Architecture activity is underway that is influenced by a number of sources including the OGSA Data Services paper.[3] The OGSA Data Architecture will identify the key interfaces that should be supported by providers of systems that manage data in order to be part of the OGSA infrastructure. These interfaces are likely to include data access, data management, and data properties (data description). Further documents will drill down into the interface descriptions for data access, management, and distribution in more detail. They include documents for databases in general, relational databases, XML databases, and files. There is much interest in other data sources, such as streams, and in event and data publishing for moving data (information dissemination).

Data access and data virtualization. To achieve data virtualization in heterogeneous environments, we need to agree on common mechanisms for accessing heterogeneous resources. Producing specifications that describe how heterogeneous data sources should be accessed through data services is the primary activity of the GGF DAIS-WG[13] (Database Access and Integration Services Working Group). This group focuses on how SQL, XPath,[20] and XQuery requests should be submitted to relational and XML databases and how the query results should be delivered. Other areas of interest include file access and result delivery and result transformations. The DAIS Working Group strives to ensure that common data sources are supported through data services. In other words, it should not be necessary to modify an existing data management system implementation to enable it to support data services, although of course by enhancing a resource manager, data services might execute more efficiently or could support additional functionality. Requirements that this group is trying to satisfy include the ability to name results for subsequent use, to handle multiple result formats, to return results in chunks, and to deliver results later or to a third party.

There are many relationships between the DAIS Working Group and existing standards activities, including American National Standards Institute/International Organization for Standardization (ANSI/ISO) SQL and SQL/XML, W3C XPath, XQuery, XQueryX (for expressing queries in XML), and Extensible Stylesheet Language Transformations (XSLT) and XQuery Serialization (for expressing XSLT and XQuery results in XML). There is an emphasis on expressing queries and results in XML. However, there is also a GGF working group[21] that is defining a way to describe non-XML data with an XML notation, where it is undesirable to return XML results. OGSA-DAI[22] software implements the DAIS interfaces and provides data service access to DB2, MySQL**, Oracle, and Xindice. The OGSA-DAI implementation currently relies on the Globus toolkit**,[23] which provides a grid infrastructure used by OGSA components and is available with the Globus toolkit as well as independently.

Standards in support of autonomic data management. We have stressed the importance of autonomic computing for the grid information infrastructure, and there are standards to help in this area as well. The Distributed Management Task Force[24] (DMTF), a non-profit collaborative body, has defined a model known as CIM (Common Information Model) to describe the characteristics of computer resources in support of management interfaces. CIM enables heterogeneous management tools to take a common approach to managing heterogeneous resources. The GGF working group called the CIM-based Grid Schema Working Group[4] (CGS-WG) has started collaborating with the DMTF to extend the CIM model to incorporate grid constructs that are necessary to support data access and management and ultimately to enable an autonomic approach to data. The WSDM (Web Services Distributed Management) Technical Committee in OASIS[25] (Organization for the Advancement of Structured Information Standards) is also getting involved with the GGF in the area of management. The collaboration between the three groups provides a helpful step for defining interfaces for interoperable management tools in support of autonomic data provisioning and management.

Replication, caching, and change-publish standards. There are no standards yet for placement management (a unique and powerful part of our vision). Nor are there many standards for the data movement and change-publish technologies. As a result, it is difficult for third-party database tools to provide replication or caching support for heterogeneous data sources, and applications need to be tailored to particular event-publishing mechanisms as well. Standards in this area will be a core part of the necessary information infrastructure for the grid. An Information Dissemination Working Group has been formed in GGF. The group will define interfaces for publishing data, subscribing to data changes, and propagating and consuming changes. As with the standards described in the earlier sections; the Information Dissemination Working Group effort will likely build on some existing standards, for example, Web Services Notification in the Web services area and on prior work such as the Grid Data Distribution (GDD) model.[26]

Data properties (standards for meta-data and data discovery). Standards for describing the properties of data enable an information infrastructure to determine whether particular data resources are suitable for particular applications. This makes it possible to build components that advertise and discover appropriate data in generic ways. XML notation is used for describing data properties. Data property definition takes place in a variety of GGF working groups:

- *DAIS-WG*—XML and relational database properties are exposed in the DAIS-WG specifications. These properties focus mainly on structural information, for example, tables in relational databases, collections in XML databases, and data types.
- *CGS-WG*—This working group has started extending the database model to incorporate properties needed by database systems in grid environments.
- *Meta-data*—There have been attempts to start a meta-data research group to review options for managing meta-data in heterogeneous environments. Readily available meta-data would be very helpful to data discovery operations.

All these groups are working towards keeping their data property descriptions consistent across the groups and consistent with other standards organizations.

Bringing the standards together. A common thread across all these standardization efforts is the need to provide an environment where requests are executed with a guaranteed QoS (through SLA negotiation), and where resource managers adhere to various policies. Standards activities are beginning to appear in the areas of agreement[27] and policies for Web services[28] that attempt to unify computing resources, software, and QoS in a general way. Projects such as the WSLA project[6] address SLAs and management issues in a Web-services environment. Likewise, the data-oriented standards just described will need to be extended to allow specification of QoS and policies. For example, standardizing a common set of terms for data placement policies may be desirable.

Current focus areas in the GGF data area with much activity and discussion include:

- Increased relationships with other standards bodies, for example, with the DMTF for management and with OASIS for Web services and Web-services management.
- Agreed-upon ways for naming bodies of data across data resource managers. There are ongoing discussions on this topic. The resolution may influence how data equivalence issues are handled.
- Handling streams and files; for example, a working group devoted to files is looking at file directory structures and naming, and a working group for file access is proposed.

Areas that are likely to gain interest and focus in the data area in GGF include meta-data, security, transactions, policies, and provenance, possibly through collaborations with other GGF working groups or standards organizations.

These activities are essential for enabling the sort of information infrastructure for the grid that we imagine, and especially, for allowing multiple vendors to participate in building that infrastructure. For a more detailed description of data standards for the grid, see Reference 29.

Conclusions

Grid computing offers many potential benefits within and across enterprises. We envision an information infrastructure for grid computing, consisting of a number of interrelated services that cooperate to virtualize information and to meet QoS goals. This infrastructure will be realized by middleware technologies, such as federation and consolidation of data, that together provide the virtualization of data re-

sources on the grid. We described existing technologies to instantiate our services and presented our vision for their future evolution. DB2 is well positioned to play a leading role in the information infrastructure for the grid. DB2 Information Integrator includes technologies such as federation of heterogeneous, distributed, and autonomous data sources, replication, change-publish and caching, data placement advice, and meta-data storage and discovery. A complete realization of our information infrastructure vision will require enhancements to the technology in order to (1) make individual technologies more dynamic and autonomic, (2) incorporate a general policy mechanism that will be used by all technologies to understand QoS goals and guide their actions towards meeting those goals, and (3) fit all technologies into a standards-compliant services architecture. Other enhancements are needed to individual services (for example, to expand the types of caching available). The net result will be that applications can access the data from diverse and distributed data sources as if from a single virtual data store.

We are actively engaged in standards activities, in technology development, and in working with customers to understand their need for this infrastructure. Whereas we have focused on the core technologies needed for the information infrastructure for the grid, future work will further enrich this infrastructure. Research is ongoing, for example, in the areas of privacy, automatic discovery of meta-data, large-scale efficient meta-data search, information quality, and exploitation of native XML stores for consolidation of data from diverse data sources.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of EMC Corporation, Microsoft Corporation, MySQL AB, University of Chicago, Oracle Corporation, SAP, or Sun Microsystems, Inc.

Cited references and note

1. Global Grid Forum (GGF), http://www.ggf.org/.
2. GGF OGSA-WG (Open Grid Services Architecture), http://www.ggf.org/ogsa-wg/.
3. GGF OGSA Data Services, https://forge.gridforum.org/projects/dais-wg/document/OGSA_Data_Services/en/1.
4. GGF CGS-WG (CIM-based Grid Schema), https://forge.gridforum.org/projects/cgs-wg.
5. V. Raman, I. Narang, C. Crone, L. Haas, S. Malaika, T. Mukai, D. Wolfson, and C. Baru, "Services for Data Access and Data Processing on Grids," *Proceedings of GGF5, Edinburgh*, July 2002. Updated for *Proceedings of GGF7*, Tokyo, March 2003. See https://forge.gridforum.org/projects/dais-wg/document/Services_for_Data_Access_and_Processing_on_Grid-GGF7/en/1.
6. WSLA (Web Service Level Agreement Project), http://www.research.ibm.com/wsla/.
7. *XQuery 1.0: An XML Query Language*, http://www.w3.org/TR/xquery/.
8. P. Gupta and E. T. Lin, "DataJoiner: A Practical Approach to Multi-Database Access," *Proceedings of the International IEEE Conference on Parallel and Distributed Information Systems*, IEEE, New York (1994).
9. V. Josifovski, P. Schwarz, L. Haas, and E. T. Lin, "Garlic: A New Flavor of Federated Query Processing for DB2," *Proceedings of ACM SIGMOD Conference on the Management of Data*, ACM, New York (2002).
10. DB2 Information Integrator, IBM Corporation, http://www-306.ibm.com/software/data/integration/db2ii/.
11. *ISO/IEC 9075-14:2003 Information Technology—Database Languages—SQL—Part 14: XML-Related Specifications (SQL/XML)*, International Organization for Standardization, http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35341&ICS1=35.
12. L. M. Haas, E. T. Lin, and M. A. Roth, "Data Integration Through Database Federation," *IBM Systems Journal* **41,** No. 4, 578–596 (Dec 2002).
13. *GGF DAIS-WG (Data Access and Integration)*, http://www.gridforum.org/6_DATA/dais.htm.
14. D. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden, "DB2 Design Advisor: Integrated Automatic Physical Database Design," *Proceedings of the International Conference on Very Large Databases 2004 (VLDB 2004),* Toronto, August 2004, Morgan Kaufmann Publishers, San Francisco (2004).
15. "XML Registry," *alphaWorks*, IBM Corporation, http://www.alphaworks.ibm.com/tech/xrr.
16. V. Josifovsky, S. Massmann, and F. Naumann, "Super-Fast XML Wrapper Generation in DB2: A Demonstration," *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, March 2003, IEEE, New York (2003).
17. M. Altinel, C. Bornhövd, S. Krishnamurthy, C. Mohan, H. Pirahesh, and B. Reinwald, "DBCache: Middle-Tier Database Caching for Highly Scalable e-Business Architectures," *Proceedings of Federated Computer Research Conference (FCRC) 2003*, San Diego, California, June 9–12, ACM, New York (2003).
18. DB2 Query Patroller, IBM Corporation, http://www.ibm.com/software/data/db2/querypatroller/.
19. Geosciences Network (GEON), http://www.geongrid.org/.
20. *XML Path Language (XPath) 2.0*, http://www.w3.org/TR/xpath20/.
21. *GGF DFDL-WG (Data Format and Definition Language)*, https://forge.gridforum.org/projects/dfdl-wg/.
22. *GGF OGSA DAI (Data Access and Integration) Open Source Implementation*, http://www.ogsadai.org.uk/.
23. *The Globus Alliance Toolkit*, http://www-unix.globus.org/toolkit/.
24. *DMTF (Distributed Management Task Force)*, http://www.dmtf.org/home.
25. *OASIS WSDM (Web Services Distributed Management) Technical Committee*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
26. *GGF Data Distribution in the Grid Environment*, https://forge.gridforum.org/docman2/ViewCategory.php?group_id=49&category_id=517.
27. *GGF WS-Agreement Specification*, https://forge.gridforum.org/docman2/ViewCategory.php?group_id=71&category_id=210.
28. *OASIS WS-Policy Specifications*, http://xml.coverpages.org/ni2003-06-04-a.html.

29. *Standards for Databases on the Grid*, SIGMOD Record (September 2003), http://www.acm.org/sigmod/record/issues/0309/IS16.Grid.pdf.

Accepted for publication June 22, 2004.

Serge Bourbonnais *IBM Software Division, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (bourbon@us.ibm.com).* Mr. Bourbonnais is a product architect in the Information Integration development group where he works on technologies for data placement, replication, and XML data stores. He holds a Master's degree in computer science from the University of Waterloo.

Vitthal M. Gogate *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (gogate@almaden.ibm.com).* Mr. Gogate is an advisory software engineer in the Computer Science Department at the Almaden Research Center where he works on data grid technologies. He received his B.S. and M.S. degrees in electronics from Shivaji University, Kolhapur, India in 1991 and 1994 respectively. He started his career with Center for Development of Advance Computing (C-DAC), a research organization in super-computing where he developed parallel algorithms in image and signal processing on the first Indian supercomputer, PARAM. For his work on DB2 DataLinks technology at IBM Almaden Research Center, he received a Special Contribution Award.

Laura M. Haas *IBM Software Division, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (lmhaas@us.ibm.com).* Dr. Haas is a Distinguished Engineer and the manager of DB2 Information Integrator development. Previously, Dr. Haas was a research staff member and manager at the IBM Almaden Research Center. She is best known for her work on the Starburst query processor (from which DB2 UDB was developed), and on Garlic, a system that supports federation of heterogeneous data sources, a key technology for DB2 Information Integrator. Dr. Haas is Vice President of the VLDB Endowment Board of Trustees.

Randy W. Horman *IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (horman@ca.ibm.com).* Mr. Horman is a Senior Technical Staff Member on the DB2 Universal Database development team at the IBM Toronto Lab. He received a B.A. degree in mathematics, computer science, and economics, as well as an M.Math degree in computer science from the University of Waterloo in 1994 and 1995, respectively. He subsequently joined IBM to work on DB2 Parallel Edition. Recently, Mr. Horman has focused his attention on database manageability, and in particular the applicability of autonomic technology. Mr. Horman is a member of the Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers.

Susan Malaika *IBM Software Group, 17 Skyline Drive, Hawthorne, NY 10532 (malaika@us.ibm.com).* Ms. Malaika is a Senior Technical Staff Member with the DB2 Information Integrator development group. She works in the area of grid computing, is active in the Global Grid Forum and is co-author of a number of GGF specifications and documents. She has also worked on Web services and XML integration in DB2, DRDA, SQL/PSM, and transactions. She also co-edited and co-authored a book entitled, "Web Gateway Tools." She is a member of the IBM Academy Technology Council.

Inderpal Narang *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (narang@us.ibm.com).* Mr. Narang is a Distinguished Engineer in the Computer Science Department at IBM Almaden Research Center and a member of the IBM Academy of Technology. In his earlier career in IBM, he worked on the multisystem clustering of DB2 and OS/390®, also known as DB2 data sharing. He made fundamental contributions to the architecture and algorithms of IBM's coupling facility and DB2 data sharing for which he received IBM corporate and innovation awards. He has published several papers and holds many patents in these areas. He is the inventor of the DataLinks technology for which he received an Outstanding Innovation Award. Currently he is leading the On Demand Information Systems Group in the IBM Research Division.

Vijayshankar Raman *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (ravijay@us.ibm.com).* Dr. Raman is a research staff member in the Computer Science Department at IBM Almaden Research Center. His main interests are in query processing and optimization and distributed systems. Dr. Raman received his Ph.D degree from the University of California at Berkeley. In his thesis he developed several new interactive query-processing algorithms, including the idea of State Modules, which are used extensively in the university's Telegraph adaptive dataflow system. Dr. Raman also developed the open-source data cleansing and transformation tool Potter's Wheel A-B-C. He has been awarded a Microsoft Fellowship and an AT&T Asia-Pacific Leadership Award.