DataCyte

Document Revision

April, 2009

1 Introduction

This document is intended to serve as a brief description of DataCyte technology, a patented technology developed within the DataCyte Group of Companies. First, a high level background overview of the framework is presented in Section 2, followed by a deeper analysis of the feature sets of the framework and how the framework relates in Section 3. Next, the suitability of the framework to some of the functional requirements of a typical Information Grid (IG) is considered in Section 4.

2 DataCyte Technology

DataCyte is a reality-modeling environment that enables a developer to model the problem domain of an application in an interactive and dynamic way. These interconnected DataCyte models can be distributed over multiple remote servers with all communication, distribution, execution and deployment being performed seamlessly in an event-driven manner by the underlying engine.

The DataCyte engine manages storage elements known as Cytes that are persisted, self aware, secure, mobile and morphic. A Cyte has memory, intelligence, is self describing, can directly communicate with other Cytes and constantly enrich themselves by creating associations with other Cytes.

Multiple DataCyte servers operate in a peer-to-peer manner and the actual distribution of the intelligent entities over the various DataCyte servers is totally transparent to the user. This means that the user of DataCyte does not have to worry about any of these aspects when developing a DataCyte solution, resulting in a fully distributed information structure that is able to perform seamlessly as a computational grid. Furthermore, the DataCyte engine can be configured by means of policies to customize the distribution strategy employed.

Cytes can contain both logic and data which are integrated into DataCyte modules that form the basis of the power of DataCyte. Autonomous DataCyte applications can be combined in a seamless manner, enabling them to operate in concert to create a single fully integrated application.

Communication between DataCyte modules is performed by means of a proprietary protocol which enhances the efficiency of communication above that obtained using HTTP, XML, SOAP, WSDL, and UDDI standards, although communication using these standards is also supported. The efficiency of communication between DataCyte modules becomes more and more relevant as (i) the level of interaction required

between the DataCyte modules increases (ii) the number of associations between DataCyte modules increases.

When things change in reality, the Cytes grow with reality and through keeping history, can be rolled backward (or forward) to any point in time. Cytes can replicate themselves on their local server or on any other server to which the DataCyte has access based on an arbitrary policy. A Cyte can also mutate and create other Cytes and Cyte structures, provided it has the correct security clearance.

Some of the acute challenges which the world of information technology is facing is:

- Slow growth of computing power, compared to other areas
- Inflexibility of data structures which makes the creation, maintenance or rather evolution of systems slow, expensive and filled with risk
- Memory and process data hungry retrieval systems
- Reduced cost of storage which results in more data being stored
- Inefficient lookup tables

The implementation of software applications based on object-oriented programming techniques is well known and has been in widespread use since the late 1980's. Although providing several noteworthy advantages over classical top-down programming techniques, it is acknowledged that applications based on object-oriented programming techniques are inefficient for a number of reasons:

- traditional object-oriented applications require an underlying database structure to be designed and provided in order to support the application;
- the objects which form the components of an application are local to that application only, without any relationship to extraneous objects or applications;
- the object characteristics such as attributes, properties, inheritances and the like are static in nature. A change to any of these characteristics renders it necessary to recompile the affected objects and rebuild the application;
- there is no global classification of objects, making it very difficult, if not impossible to construct applications which make use of objects which are distributed across a number of remote servers; and
- the separation of objects and data necessitates the provision of separate facilities for data management and streaming of data to physical hardware devices.

Object Orientation in its current format has brought great benefit to the application developer as well as ensuring the reliability and performance of applications, however Object Orientation got side tracked by the notion that data should be stored in a separate layer from the application structure.

It has been predicted since the early 1980's by academics that the relational database

model will not be able to cope with the emerging requirements of future large scale systems. And today, Object Oriented Database (OODB) already outperform relational databases by a significant margin in many large scale applications. DataCyte is not a relational database, neither is it an object-oriented database. However, it does derive many of the same performance benefits of OODB systems without incurring any of the drawbacks of those systems, such as class versioning and the schema mismatch problems that accompany it.

DataCyte has done away with the separation of the data layer from the application layer and has created an environment where the developer is able to model the real world in real time, making changes at any time while the system is running live without any disruption to the functioning of the application.

DataCyte converts super linear time complex searches to linear space and sub-linear time complex searches, by exploiting the direct relationships between data and by using localized indexes stored within the DataCyte space.

3 Analysis

It should be noted that DataCyte technology is a general purpose software development frameworks for distributed applications. This section evaluates the framework based on criteria that are not specific to a any given type of application, while the section that follows presents a suitability analysis more focused on the requirements specific to IG.

3.1 Data Storage

The DataCyte core is a general purpose database system and it is a true distributed database. DataCyte technology can model arbitrary relationships (associations) between data elements and have the data seamlessly distributed on a grid of computing resources. Then, dependent on policy configuration, any information can be accessed at any location in a single global view. Furthermore, this data can be accessed and modified in a transactionally aware ACID compliant environment.

3.2 Security

DataCyte is able to communicate via a web service interface using PKI and TLS/SSL, however, the security framework enables a richness of definition for authorization policy. DataCyte technology places the authorization policy in the same DataCyte space in which the application's data and logic is stored. Thus, security authorization information, encapsulated in Policy Based Cytes (PBC's), is accessed orthogonally to the data and

services they protect. Any PBC's which are traversed on the path to accessing a protected Cyte are added to the transaction context, after which those PBC's may intercept any operation on the protected Cyte. Thus, security policies can be very flexible and include implementations for standard role and permission based authorization as well as various encryption policies. Furthermore, these policies can be composed in DataCyte space, providing a least privilege intersection of the policies by simply manipulating the associations between Cytes and their policies. Finally, it would be relatively straight forward to construct a PBC which defers authorization to a third-party based community authorization service, if integration is necessary.

3.3 Disaster Recovery

DataCyte technology provides more than one solution to the disaster recovery problem. Replication policy is defined by PBC's stored in DataCyte space, in the same way that security policies are defined. These replication policies define the replication and data integrity rules, which are implemented by intercepting read and write operations to the Cytes managed by the policy. Thus, a consistency policy can be defined to automatically create new replicas elsewhere in the grid based on known good copies of the data in a failure situation.

Furthermore, DataCyte provides two separate mechanisms for backup and recovery. In a DataCyte environment, consistent snapshots of a replica can be taken at the file system level (for local site backups) and DataCyte space aware exports of data to XML can be performed (for distributed backups across multiple sites).

3.4 Query Approach

In terms of query approach, traditional databases deploy SQL. Many third party middleware tools exist to integrate SQL based solutions, to the extent that such tools are a present day commodity.

Relational databases, however, do not scale well under conditions of extremely high data volumes. This lack of scalability stems from the fact that in order to perform a foreign key lookup (ie, navigating a relation), the query processor must perform a search through the index of all the related entities of the same type. This problem is further compounded in the case of complex joins where query back-tracking becomes required.

Queries in DataCyte space offer extremely high performance, since direct associations between related DataCyte instances can be navigated. That is, there is no table in which entities of a given type are grouped which must be searched in order to find the subset of related entities. In DataCyte space, selectivity is performed via filters that constrain the

result set of a given association to certain attribute values. High speed key based lookups can also be performed via indices stored in DataCyte space, which indices do not degrade in performance relative to the number of data records, but rather in order of the complexity of the key which is utilized.

Unlike clustered SQL databases, which typically offer relatively primitive partitioning of tables over a number of computer resources connected via a local area network, DataCyte replication policies are both customizable and flexible. DataCyte queries are inherently distributed and can function efficiently over a wide area network. This is due in large part to the direct associations between Cytes, which enable the query processing to be focused, directing connected hosts to perform the query in sub-parts while fully exploiting locality of data during the query pursuit. Thus, only the results of a query are transported over the network before being centrally composed. This distributed nature extends to the indices as well, since these are stored orthogonally in the same DataCyte space.

In the traditional relational world, services such as OGSA-DAI (Open Grid Services Architecture, Database Access and Integration) can be used to provide a federated view of multiple disparate SQL databases (as well as XML based data sources) for centralized query orchestration. However, OGSA-DAI is a heavy weight XML based protocol, and does not, in general, provide the kind of selectivity within the data repositories that is provided for by DataCyte. Thus, potentially large amounts of data must be transported over the network to a central orchestration point, before the data can be transformed, composed and queried centrally. The fact that the DataCyte space can be represented in a hierarchical fashion can be exploited in order to integrate DataCyte with industry standard middle-ware tools and even OGSA-DAI. Although this is an area still to be investigated, it should be possible to treat this hierarchy as an analogue of an XML document and use standards based XPath expressions and XMLQuery as a query language.

3.5 Performance and Scalability

As discussed in the previous section, DataCyte technology provides for greater query performance than is possible with relational databases in circumstances where there are extreme amounts of data being managed or where there is a multiplicity of relationships. This performance gap is further widened as the distributed nature of the data increases, since silos of relational databases must be indexed by a single centralized index which grows in size proportional to all data records managed by the system.

DataCyte has no single global registry and thus no single bottleneck (nor any single point of failure for that matter). In theory, DataCyte technology is infinitely scalable in

this regard, since individual Cytes need only be aware of the specific Cyte instances to which they are directly related. That is, Cytes do not need to be aware of the global universe of other Cytes to function in a meaningful way. Yet, these Cytes are still globally discoverable within a single DataCyte space and can be associated with any other Cytes within that universe.

In addition to this fundamentally different architectures with respect to scalability, DataCyte makes further use of optimized network communication protocols and efficient storage mechanisms. In particular, light data weight in DataCyte is one aspect for which some empirical benchmark data does exist. In this particular example, multiple blood peptide datasets from Cedars Sinai were imported into a DataCyte system for analytical processing. The relational database they had previously been using had a data weight of approximately 1.3TB, on which they could not perform live queries efficiently. Whereas DataCyte, was able to represent in a space of 60GB, while still supporting efficient querying for processing purposes. Queries that took 1 minute to 11 minutes in the relational context took under 1 second in the DataCyte context. Complex queries that took 24 to 36 hours in the relational database took 11 to 44 minutes in the DataCyte system. All of this being accomplished using commodity laptop hardware, where massive database servers were required in the relational case.

Finally, DataCyte does not have to pay the performance costs, both in terms of application performance and in terms of developer efficiency, of the well publicized impedance mismatch between object oriented programming languages and the legacy relational storage model, where traditional development approaches require inefficient and complex Object Relational Mapping (ORM) technologies to be used.

3.6 Discovery

Discovery, in the context of this analysis, is defined as the process by which resources can be located in a grid environment. DataCyte discovery is based on the concept of a DataCyte space. A DataCyte space is the universe defined by how Cytes are connected to each other. Thus, any two Cytes which are mutually reachable by some path of associations between them (however long that path may be) are said to be in the same DataCyte space. Therefore, having a reference to any DataCyte within a given DataCyte space enables all other Cytes within that space to be discovered and accessed, provided that security policies defined by any PBC's along that path are satisfied. This is an extremely powerful concept, because any given logical node (Cyte) in a DataCyte network, need only know about the neighbors which are important to it. For example, a hospital would only need a reference to a patient's Cyte representation in order to access all the other hospitals previously visited by that patient, because they too are associated with that particular patient's Cyte representation.

Cytes are self describing, providing a common interface for discovering their properties and associations. Furthermore, data and services are accessed orthogonally and are defined in the same DataCyte space. DataCyte treats data and services uniformly, in exactly the same manner. Finally, all that is required for discovery is a single starting point somewhere within the global DataCyte space. Thus, in a DataCyte environment, there is no central registry, and hence no single point of failure, all while providing a global namespace which solves the discovery problem.

3.7 Configuration

Configuration management is an important issue in contemporary systems, even more so in distributed applications. DataCyte technology addresses configuration issues in the same manner as everything else is handled, by incorporating the information into a Cyte representation in DataCyte space.

In DataCyte, server side configuration is minimal, requiring literally only enough information to enable the server to connect with some other nodes in the DataCyte space and to allocate server side resources (file system locations, local resource limits, etc.). Everything else is configured in DataCyte space exclusively, which configuration information is self describing through the same single common interface described earlier for data and services. Most of the site specific local configuration is also accessible and can be overridden in DataCyte space. Furthermore, the configuration data benefits from the same replication and centralized view features that apply to all Cytes. Finally, configuration is required for exception cases only, with sensible defaults being defined for virtually all configuration parameters.

3.8 Deployment and Monitoring

In DataCyte there limited systems monitoring has been implemented, however, the global DataCyte space presents some unique opportunities for doing so that are being investigated. Hardware devices can be represented as special purpose Cytes with self describing attributes corresponding to the measurements that can be monitored remotely. As these devices are exposed in DataCyte space, they would benefit from the discovery features as well as the Cyte event interface described in Section 3.9 to provide functionality similar to MDS triggers. Long running transactions are represented by Cytes that expose the status of those transactions except that the Cytes have the advantage of being self describing, whereas the WS-RF properties of interest would have to be known a-priori.

For deployment issues DataCyte provides a rich mechanism for remote deployment. Updated versions of application code can simply be deployed within DataCyte space

from a central location, by attaching the new version of a Cyte (or multiple Cytes) in the appropriate location. Furthermore, these updates can be conducted on live systems, without having to take the system down for maintenance. More traditional deployments are also possible, where updates are packaged into a compressed XML release file, which can deployed by a local system administrator in a controlled fashion as part of a well defined release cycle.

3.9 Service Interface and Events

Native communication between DataCyte servers is accomplished using a proprietary light weight protocol; however, it is still necessary to inter-operate with other systems. For this purpose a well defined standard is followed for defining service interfaces.

Executable Cytes can be exposed as SOAP web services via modDSA, an integration module provided by the DataCyte server implementation to plug into the open source Apache web server. This module can also be used to expose an HTTP interface on top of Cytes, which can be used for everything from navigating the DataCyte space with a browser to implementing fully fledged web applications (such as portals) within DataCyte technology.

DataCyte also provides a powerful event interface, where observers (implemented by other executable Cytes) can register for event notifications by simply being attached to the event source in DataCyte space. This provides a distributed mechanism for handling arbitrary events which has also been used in previous DataCyte applications to implement a full audit trail. Events in DataCyte are flexible and are tied to the life-cycle of the Cyte to which they are attached, which may be a function of the application state. Or, alternatively, events can also be tied to persisted data resources, providing functionality similar to triggers in relational databases.

4 Suitability to Requirements

Mainstream approaches to creating an information network is based on either the centralization of data or the creation of self contained, organization-centric data repositories. These approaches are not necessarily bad, as they present workable solutions. However, applying these kinds of solutions to data that is inherently distributed by nature presents new problems that would not exist if the data were stored and retrieved in a distributed manner.

Taking into account that there is a high probability that data is generated and stored at different locations and the information required from this data is also required, quite likely, at different locations over any given period of time, at some point, the disparate data would need to be consolidated into usable information in order for decision-makers to perform their functions and query the information network to make any sense and be accurate in real-time. The automatic consolidation of multiple disparate data and infor-

mation into a single federated view is plagued by complexity, reliability and data distribution issues.

4.1 Centralization

In addition to the problems mentioned above, some of the other problems that are created include:

1. Single point of vulnerability. This applies both to availability and security. AVAILABILITY: All users would be affected, should the centralized system be offline. SECURITY: It is more likely that malicious attacks would succeed when they are directed and focused on a single point because multiple secure locations would have the added security of 'safety in numbers'.
2. Scalability. Software and hardware limitations are more likely to become a factor sooner than anticipated because there are higher volumes of data at a single point.
3. Hardware. There is a larger load placed on a single system, which would require more expensive hardware to be installed than would be necessary for a system with a distributed load.
4. Bandwidth. There is an unnecessary shipment of data from the point of capture to the central hub. All requests would require data, which may originally have been captured locally, to be repackaged and reshipped back to the point of capture.

4.2 Organization-centric

As in the case of centralization, an organization-centric approach too has its own serious problems that need to be considered. These include:

1. Consolidation. Parts of patient's medical record will be stored at multiple locations that operate independent of each other. This presents a serious problem as the multiple parts would, at some stage, need to be consolidated and presented as a single federated view of the patient's medical record. It is very difficult to guarantee that manual or automatic processes would be able to accurately differentiate between parts that are from the same record and parts that are not from the same record. A typical approach to the consolidation would be to make use of a global registry or index. Effectively, the global lookup registry would be metadata describing where and how data is stored.
2. Transactions. It is extremely difficult to manage a global registry in a transactionally safe manner, when multiple users may request or change the same data at the same time. This task becomes nearly impossible when the data's physical location can change as well.

4.3 Data-structure-centric – DataCyte Technology

A data-structure-centric solution alleviates most (if not all) of the problems mentioned above by storing data at the source and shipping it to a variety of replication sites. The data is simply presented as a single federated view, although it is automatically spread over a distributed information grid.

DataCyte BOXes act as data collection units that interface with existing systems at the variety of data collection centers. The primary focus of a DataCyte BOX is to gather information and data records during the course of normal operations from existing systems at an institution and present the data to the health information network so that it can be accessed remotely by interested parties.

Data records are presented to the health information network either as a local replicator as a direct link to the data (source / any available replica). Replicas provide the added function of acting as mirrors to the same information.

Clearly, the probability that data will be available from anywhere at any time will increase as more replicas are created on the network. Direct links require data to be accessed from the source (or a replica), which means there is a shipment of data that takes place over the network.