Data Integration

Steven Barlow
University of Passau
barlow@forwiss.uni-passau.de

July 24, 2000

Abstract

The growing use of computer networks has lead to more complex and dispersed data sources. Where once a single database system provided the information needs of an organisation, today collections of several different types of systems are being used. In order to still provide consistent and effective use of the information contained in these varied sources, some sort of higher abstraction is needed. This abstraction is provided by data integration systems which can unify the logical data models of various sources and provide query processing across all of these sources simultaneously. We describe the attributes that a data integration system must have, and then present three example data integration projects.

Contents

1 Introduction

1.1 Backgound Information

Modern information systems and the applications that run on them have moved beyond a simple one source, one data model. The increasing use of computer networks; a trend made possible by both sinking telecommunication costs and the spread of effective network standards, has lead to a wider distribution of information and an ever increasing number of available sources.

Today nearly anyone can provide some set of information through the Internet and allow geographically widely dispersed people nearly transparent access to that information. Businesses with widely scattered interests can use the same technologies to provide better access to internal information through a company-wide intranet. By using such an intranet, business decisions can be made based on more complete and up-to-date data. For example if a company was deciding on how much to charge for a product it might want to access production information from several different factory locations at the same time in order to have a more accurate picture of what the manufacturing costs are. If this information was not available through a computer network the decision might have to be delayed until the needed data could be collected and sent, or it might be made based on a smaller subset of the information.

In addition to simply providing better access to separate information sources the computer network makes it feasible to combine several different types of data into a more powerful composite object. For example, in a traditional environment an insurance policy might be saved in a database as a record. However using modern (multimedia) data it would be possible to define a policy object that is composed of several different data types. A picture object might provide a visual record of the insured items, a document management system might contain accident or theft reports from the police, an accounting application might provide payment history of the premiums, and the traditional database record could still provide the actual policy information. A policy object of this sort obviously provides a much richer set of semantics than the simple databse object alone.

Composite objects provide a much richer semantic base for application developers and users, but not without additional costs. These costs occur primarily at the programming level since the developer must have explicit information about the location and attributes of each data source he wishes to incorporate into the final application. In a dynamic environment it would then be necessary to constantly modify the applications to reflect new data sources, or changes in how the sources provide information. It has proven hard enough to write and maintain good applications based on knowledge of one system. The alternative is that the application is written to expect extensive user input during use as to the location and form of the needed data. This is clearly not a good solution. To continue with our insurance example, an employee at an insurance company still needs to view a policy as an atomic unit irregardless of how many data sources it actually combines. Their job is advising costumers and selling insurance, not defining methods to access databases. The goal is to provide more information that allows people to work more effectively without adding complexity.

The solution to both problems lies in unifying the logical view of the data and the means required to access it. This is achieved by adding a higher layer of abstraction above the data sources that encapsulates the specific knowledge needed for each source. Such a system is said to provide data integration since it combines lots of different information into a logical whole. In much the same way that traditional databases provide a consistent interface to the underlying information that can then be used to write applications, a data integration system provides a consistent interface to many data sources with very diverse internal structures. Using this interface the programmer is insulated from the specifics of each individual source, but can still provide flexible access for the application users.

1.2 Requirements of Data Integration

In order to achieve the kind of flexibility discussed above, a data integration system must address several problems. First, it must know where the data is located and how it can be accessed. These problems are typically solved through the use of wrappers. A wrapper is a software layer that translates the information from a specific data source into some standard form understood by the data integration system. This form could be defined through some object model, described through XML, or in any other standard way the developers of the data integration system decide. In this way the programmer defines the methds used to actually get to the information once, and in only one place. If the data source changes its internal model, then only the code in the wrapper needs to be modified and the rest of the data integration system and all applications written based on it remain untouched. If a new information source is to be added, then all that needs to be done is the implementation of an appropriate wrapper.

Second, it must provide some means for rectifying the logical differences that even very similar data from different sources contain. Consider an application to sell used cars online. Such a system might collect car offers from the classified ads of several newspapers. Although the information from each ad is very similar; i.e. the seller's name, the type of car, the model year, etc. it can be very difficult to combine this information into a logical whole. Even such simple issues as the attribute name of a record can cause trouble. One newspaper might use an attribute "Name" for the name of the seller, while another uses an attribute with the same label to describe the car manufacturer's name. Further, what happens if a seller places ads in two different papers? Such inconsistencies need to be at least reduced as much as possible. These problems are addressed by the standardized internal model of the data integration system, combined with the wrapper for each data source. An application programmer need only be concerned with what such a field is called from the view of the data integration system. If the system used has the fields "sellerName" and "manufacturerName" the application can be programed with those values and the wrapper transforms the labels back and forth as needed.

Finally, the data integration system must provide some effective means to define queries over all of the participating sources. This is the layer that really provides the data integration system with its power and flexibility. A query is defined by the user without regard for where the information needed is located, or how to access it. This is achieved by the definition of a generic query language designed to run together with the internal information model used by the data integration system. The queries are then defined using the notation of the internal model of the data integration system and depend on the system (i.e. the wrappers) to convert the subqueries into a form understood by each information source. This effectively frees the users from learning more than one query language, but still allows access to varied resources.

Historically data integration has been provided by a middleware layer [6]. The name arises from the logical location of the software between clients (users and applications) and the data sources used by those clients. Such a system has typically had a very monolithic design with one large middleware server. This can obviously cause scalabilty problems if the data set that is to be managed is very large or widely dispersed. In such cases the servers need to be combined in a tree-like hierarchy with each server managing a smaller subset of the data, and queries being passed from one server to the next in order to access the appropriate information. A data integration system must therefore scale well, and be able to be combined in a building-block manner.

1.3 Overview of the Following Sections

In the following sections of this paper we will be discussing three systems that were designed to address the needs of data integration. The developers of each system have made different choices about the best way to provide the needed services. First we talk about the Garlic System from IBM's Almaden Research Center which is structured as a traditional middleware layer and has a strong emphasis on integrating multimedia data contained in specialist systems. The second system, TSIMMIS, comes from Stanford University and emphasizes less the unification of diverse logical schema and more the use of an internal object model in order to access the data simply and consistently. The last system discussed is ObjectGlobe from the University of Passau which has a very network-centric, practical model with emphasis on flexibility, security and expandability. The final section is a short evaluation of the three systems.

2 The Garlic System

2.1 Overview of the Garlic System

The Garlic[1] Project originated from IBM's Almaden Research Center and is primarily concerned with integrating heterogeneous multimedia data [3]. Due to this emphasis Garlic deals more with the second problem domain described above; the integration of different data types into complex composites. Garlic therefore tries to support not only complex queries, but also new interaction models for multimedia data.

A primary assumption is that multimedia data is most likely to be stored in separate special-purpose multimedia systems (i.e. a video server for video data, a document management server for document data, etc.) which then provide the query and management of that specific type of information. This frees general-purpose systems from having to be good at everything, therefore (hopefully) allowing more efficient and extensive access to the data. The effectiveness of content-based retrieval is very type specific. Years of research have lead to good methods for text retrieval and evaluation, but methods for other types such as video or picture data lag considerably behind. By providing a seperation of servers for each type it is possible to use the best technology when it becomes available without having to rebuild the entire system for each change.

As mentioned in the introduction the traditional solution to this integration problem is the use of a middleware system. The developers of Garlic agree, but feel that the currently available systems are too restrictive in their support of data sources [5]. For example most middleware systems do not support access to the file system, the exact location where most multimedia

[1]"Garlic" is not an acronym - the developers of the system just really like the taste of garlic

data is currently stored. Garlic was therefore designed to grant access to any data source, irregardless of its location and allow queries across all sources.

2.2 Garlic Architecture

The architecture of the Garlic system can be seen in 1. At the bottom of the figure are a series of Data Repositories. These constitute the information providers and can be any type of information server including (but not restricted to) traditional databases, document management servers, video servers, and traditional file systems. Each repository has its own internal data model and methods for accessing and querying the data it contains.

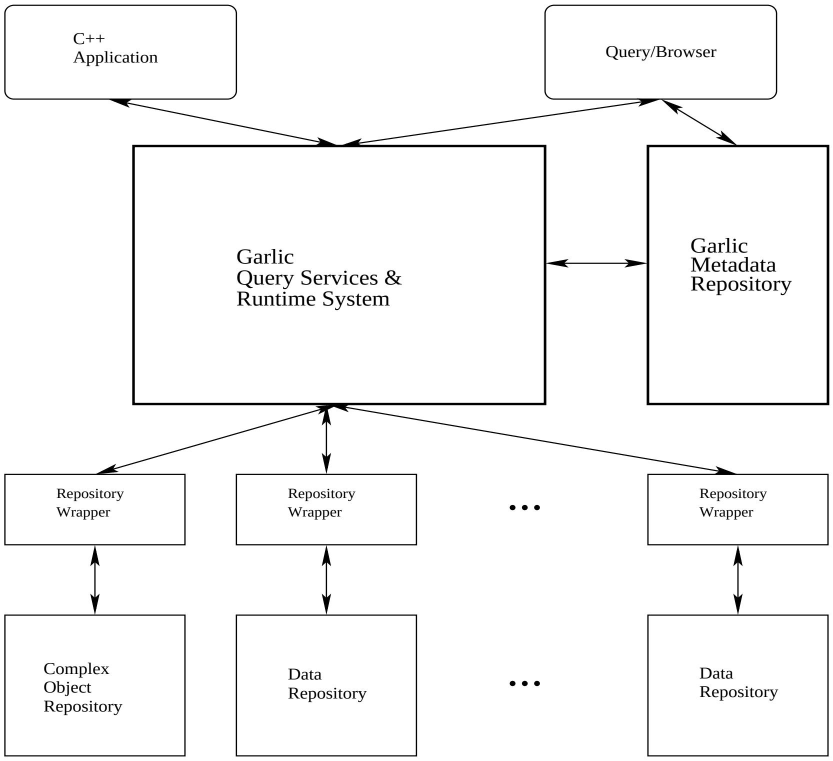

Fig. 1: Garlic System Architecture

Above each data repository is a repository wrapper. This wrapper is responsible for the communication between the rest of the Garlic system and that data repository. Included in this communication is the transformation of the repositories' local schema into the unified Garlic schema, and the translation of queries from the Garlic system back into the native query language of the data repository. The wrapper therefore acts as a mediator between a data repository and the Garlic system as a whole.

Each Garlic system has at least one data repository, the Complex Object Repository (seen at the bottom left of the figure). This Complex Object Repository contains all information about the composite objects used by Garlic applications, or the methods needed to contruct such objects out of several different data sources. The Complex Object Repository can be contained in any database system (currently Garlic uses the ObjectStore Database) and therefore needs a wrapper of its own through which it communicates with the rest of Garlic.

The first of two core elements of Garlic is the Garlic Query Services and Runtime System (as seen in the middle of the figure). The Query Services and Runtime System provides the unified, object-oriented view of all data contained within the Garlic system. It is this component that controls the processing of all queries and therefore has transactional control over Garlic application programs.

The second core element (also in the middle of the figure) is the Garlic Metadata Repository which contains the Unified Data Schema, a collective model of all data contained in the current Garlic system. This information is passed to the Metadata Repository by the wrapper for each data repository in the internal Object Model format where it is collected into the overall schema used by the client applications to describe their queries.

Garlic client applications are written in C++ using Garlic's external API and the Garlic object query language. The combination of these two programming methods allow client applications to access both dynamic information contained in the Metadata Repository through the C++ API and to define their queries in the traditional way using the Garlic Query Language to describe the sort of information being sought.

The final element is the Garlic Query/Browser which is a client application designed to test new models of interaction and provide a means for users to interactively navigate through the available data. This becomes increasingly important as the amount of data grows, and the user does not exactly know what types of data are available. For an extensive example of the Query/Browser see [5].

2.3 Unified Schema and Queries

In order to unify data from such diverse sources an unifying data model is required. The basis of the unified Garlic Schema is an object-oriented data model based on the ODMG-93 specification. Due to the heterogeneous environment (as opposed to the centralized model envisioned by the ODMG) several extensions to the Garlic Data Model were needed such as expanded object identity, better handling of legacy references, support of views, and better general expandability of object attributes [3].

The Garlic Data Model is built by describing the data objects using the Garlic Data Language (GDL). The GDL is an extension of ODMG's Object Definition Language (ODL), and is used to descibe all data objects contained in a Garlic system and also to define queries across this data using the Garlic Query Language. As such it builds the standard interface among all parts of the Garlic system.

The Garlic Query language is an object-oriented extended SQL dialect. The object-oriented additions are needed in order to support the object notions in the Garlic Data Model. In addition, since Garlic can store data in any information server, the query language must be able to be extended to support different types of search requirements. While traditional databases use exact match predicates, other data sources may use more "inexact" matching, i.e. "Retrive all the pictures with a red circle in the middle third." Such open requirements can then be dealt with by adding new methods as needed using the object-oriented extensions.

Once a query has been defined using the GDL to access the Garlic Object Model, the query processor breaks it into smaller pieces and sends each subquery to the wrapper responsible for the appropriate data repository. It decides what piece to send where based on the information contained in the Metadata Repository and using a cost estimate defined by each wrapper. The support for the cost estimates is still very incomplete, but could eventually be used to optimize the run-time of each subquery. Each wrapper is then responsible for converting the subquery into the native query language of its corresponding data repository, and then converting all results back into the Garlic Object Model so that all subquery results can be reunified and the (hopefully) correct answer for the complete query can be found. For an in-depth example of queries defined with the GDL and their use, see again [5].

3 The TSIMMIS System

3.1 TSIMMIS Architecture Overview

The TSIMMIS[2] Project from Stanford University also has the integration of any type of data source as its goal, with a special emphasis on unstructured or semistructured data. Such unstructured data is typically encountered on the Internet in the form of HTML pages, but can also include such applications

[2]TSIMMIS stands for "The Stanford-IBM Manager of Multiple Information Sources."

as a Newswire feed. The TSIMMIS Project has been successfully used to access Internet data sources [7].

The TSIMMIS architecture can be seen in 2. At first glance it has a similar structure to the Garlic System. At the bottom of the figure are a collection of several heterogenous Information Sources. These sources are not specified nearer and could in fact be almost any data source. Similar to Garlic the interface between these data sources and the rest of the TSIMMIS system is a collection of Translators (in Garlic they were called wrappers), one for each Information Source.

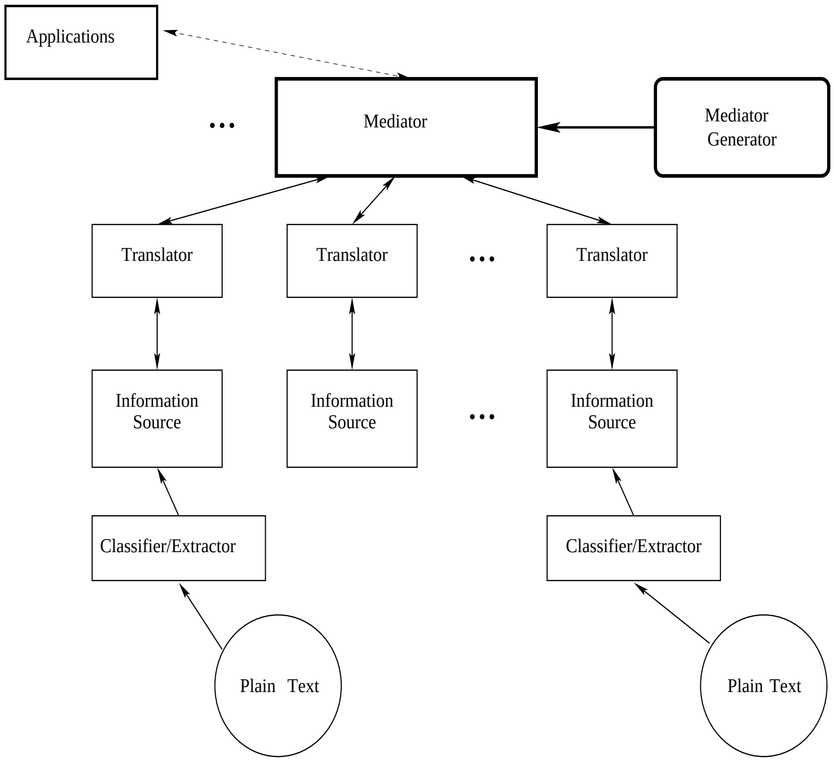

Fig. 2: TSIMMIS System Architecture

The Translators are the part of the TSIMMIS system that needs specific knowledge about each Information Source, and provide the translation between the data representation used internally by the Information Sources and the object model used by TSIMMIS to intergrate data from several sources. Further, the Translators are responsible for converting query requests from TSIMMIS into the locally used query methods.

Under the Information Sources we see one of the first major differences between Garlic and TSIMMIS, the Classifier/Extractor. TSIMMIS places special emphasis on the inclusion of unstructured data sources. The Classifier/Extrator is responsible for converting plain text into a structured form that can then be included in TSIMMIS queries.

TSIMMIS is unique in that it does not try to intergrate the local data schema of all of the managed sources into a global data schema [8]. Therefore there is no global Metadata Repository as we have seen with Garlic. Instead TSIMMIS provides for the definition of a set of software modules named Mediators (seen in the middle of the figure). Each Mediator contains the knowledge needed to access information from one or more sources and process that type of information. For example, we might want to describe a Stock Information Mediator that can pull up-to-date stock prices from an online ticker, and combine it with relevant news articles from a source such as the Wall Street Journal Online. Later queries are then directed to the relevant Mediator (i.e. one that knows about the type of information being looked for).

3.2 OEM and Querying in TSIMMIS

TSIMMIS adopts a very simple unifying object model called the Object Exchange Model (OEM). This model is self-describing since every object includes a tag that describes its meaning. For example an object representing the number of employees at a company location (in this case 80) might be expressed as:

[number-of-employees, int, 80]

The advantages of such a simple system are that a human reader can immediately understand what this object is used for, and that it can be combined with other objects to contruct more complicated composite objects. For example, an object representing the number of employees at two different locations could be constructed like this:

[total-number-of-employees, set, num1, num2]
num1:[number-of-employees, int, 80]
num2:[number-of-employees, int, 200]

By building on this simple foundation OEM can be used to describe very complex object types and can act as the unit of exchange between Translators and Mediators. Once a Mediators has been defined it exports an interface of OEM objects which can then be accessed through the use of the OEM-Query language (OEM-QL). Client queries are written using OEM-QL to describe the desired result set and are then directed to a Mediator that knows how to access relevant information. Which Mediator should handle which query is decided by a human operator. Once the query reaches the Mediator it is broken into subqueries and passed on to the appropriate Translator for processing at the Information Sources using their own query methods. The results of this processing are then passed back through the Translators to the Mediator as OEM objects where they are incorperated into the global result set and returned to the client application. An extensive example of OEM-QL queries can be found in [4].

Since OEM acts as both the input and output format for Mediators it is possible to chain a series of Mediators together to provide more complex information processing abilites. In addition, once a Translator is defined for a particuler Information Source any number of Mediators can access that data.

4 The ObjectGlobe System

4.1 Overview of the ObjectGlobe System

The ObjectGlobe Project at the University of Passau [11] begins initially with a much simpler vision of data processing requirements than the other two systems. These requirements can be broken down into three broad groups: first the actual data is needed, second what should be done with that data (i.e. the operations) need to be defined, and finally some computer time needs to be provided to run the operations on the data. Consistant with this model, ObjectGlobe defines three categories of providers.

The emphdata-providers act as the information sources or repositories as in the other models. ObjectGlobe does not make any further assumptions as to the form or structure of the data being provided, and is therefore highly extendable. Each data-provider runs an instance of the ObjectGlobe server which acts as the wrapper/translator between the information contained at a source and the rest of the ObjectGlobe system.

The emphfunction-providers provide operations that can be used on the data. The types of operations that can be defined is not limited in any way. This means that even the "internal" operations such as JOIN can be expanded as better methods become available. This facility is unique among the systems talked about here [2].

The emphcycle-providers contribute the actual computing time needed to run a particular operation. In particular it is possible to scale the amount of machine time needed with the complexity of the operations being run.

Typically the data and function-providers also act as cycle providers (but this is, of course, not required).

By dividing the requirements into these three categories the framework is laid to be able to distribute all three, leading to a very flexible and network-centric model. Since any query needs access to all three resources, ObjectGlobe has a very balanced structure which does not over-emphasize data distribution, but rather the distribution of all resources and depends on using the network to reunite them in order to run. The other systems presented here (and the vast majority of currently-used systems including the Internet) are most concerned with the distribution of data. That the ability to move all resources in order to provide the most efficient execution can improve performace has been shown in [1].

The designers of ObjectGlobe see the possibility of creating a very open information market in which ther exist any number of providers for each of the three needed resources. Such an open market holds many possibilities for new business models. A company may decide to offer the use of "spare" cycles to external users, thus acting as a large "virtual" computer. In addition, it is very easy for owners of information resources to offer external access to their data, and by the use of ObjectGlobe's security restrictions limit access to only "paying" costumers. Perhaps most interesting is the possibility for companies to define new operations and functions, and then offer them through the Internet to any interested user. Coupled with the ObjectGlobe's ability to restrict use of a function based on cycle provider, this would allow the company to demand that the operation run on only their machines, thereby creating two sources of revenue (i.e. the company could sell both the operator and the cycles, or use the amount of computing time needed as the price base for operator use).

4.2 Query Processing in ObjectGlobe

The structure of query processing in ObjectGlobe can be seen in 3, and is divided into four main steps. In the first step the query is parsed, and a list of the needed resources (i.e. how many cycles, what data and functions, etc.) is sent to the lookup service. The lookup service contains a list of all available resources, including cost estimates for their use and authorization lists as to what users are allowed to access which resources. The lookup service returns a list of all resources that could possibly be used to run the query.

This list of resources is then passed to the optimizer which calculates cost estimates of each valid plan that could be run to satisfy the query. Note that the optimizer does not always have to find a valid plan. If very strict cost restrictions are in effect, then it is possible that none of the otherwise valid

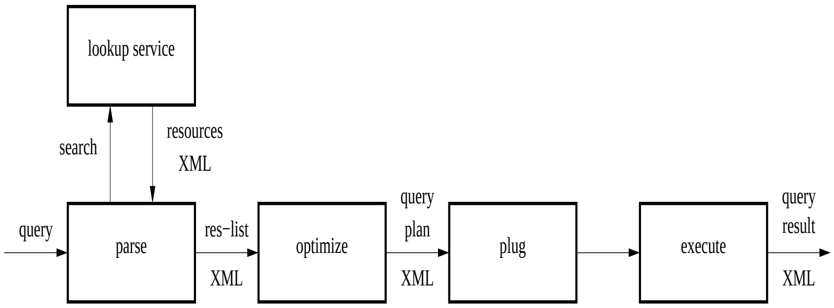

Fig. 3: Query Processing in ObjectGlobe

plans could be run (i.e. they are just too expensive). In this case the query execution is aborted.

Once the optimizer finds the best plan for the query, this plan is distributed to the various resource-providers that are going to take part in the query execution. This is called the plug phase. During the plug phase the actual communication paths are extablished to the various resource-providers. If one of these providers can not be reached (i.e. is has gone offline or crashed), then the plan must be aborted and replaced with a different valid plan when one exists.

In the final phase, the plan is actually executed at the sites of the various participating resource-providers. The progress of this execution is carefully watched over by a monitoring service to ensure that either no failures occur or that if they do the system reacts in a timely and graceful manner. For example, if one cycle-provider aborts in the middle of its execution and it therefore makes no sense to continue the query, the system will then halt the execution of that query at any other cycle providers so that system resources are not wasted.

The interchange format between the various steps is XML. The use of XML allows each step to be used independantly. For example, a human operator could define a query plan by hand instead of using the optimizer, pass this plan to the plug stage in the normal way, and then execute that plan. This can be very useful when the performance of two alternatives is to be tested. In addition, although ObjectGlobe does not place emphasis on an explicit browser function in the manner that Garlic and TSIMMIS do, such a function can be readily realized since the lookup service exports the list of resources in XML.

4.3 Ongoing Development

ObjectGlobe is an ongoing research project and as such has several areas that still need to be addressed, or for which support needs to be expanded [1]. First, the lookup service needs to be improved. Currently the lookup service is a centralized process through which all queries must run. Since ObjectGlobe is a highly distributed system this centralization could easily lead to a performance bottleneck. It is much more desirable to implement the lookup service so that it could also be distributed across the network in the same way as the data, operations, and cycles. In a related effort, the number of lookups during the query parsing process should be reduced in order to avoid any performace hits in this area. Second, much more work could be done in the area of the cost models used for query optimization. The current cost models are very simple but could be greatly expanded to allow more sophisticated and varied optimizations based on the individual goals of each service provider. Finally, the authorization model needs to be expanded to allow finer grain authorization either through the use of encryption or information splitting. This could be important if for example a cycle provider was not allowed to see some specific piece of data. Under the current model that cycle provider would be shut out entirely from participating in plans with that data (even if using that provider would be very efficient for at least a piece of the plan). Such expansions would therefore allow a greater number of plans to be considered, and increase the probability of finding more efficient ones.

5 Evaluation

The Garlic system seems most applicable to use within a company in order to manage complex multimedia data objects. The use of complex composite objects will certainly grow as the management systems for multimedia data become more mature. These objects could then be used in such diverse applications as insurance (as metioned in the introduction), medical records, the management of contruction/manufacturing plans, and many others. Within this context Garlic provides a very consistant programming interface through its object model and object oriented query language, and a high degree of flexibility through the possibility to define new interaction methods such as "query-by-example" to make effective use of such composite objects.

Due to the lack of guarantees about the consistency of data repositories being accessed by both Garlic and other systems simultaneously [6] Garlic could not be used to provide "external" access for interested users in addition to the normal operation of a data repository. Furthermore, the complex objects are stored in the Garlic system itself, and are dependant on all sources being available for proper query execution. Garlic is therfore an "all-or-nothing" proposition at present that does not lend itself to being deployed unless all of the data repositories can be directly controlled. This limitation leads to the conclusion that although good for integrating data within an organisation, it is not appropriate for unifying data from several independant information providers that are not under direct control.

Better in this regard is the TSIMMIS System which has successfully been used to extract information from Internet sources [7] and is therefore much better at integrating external data sources. It does not require the maintenance of a unified data scheme which has advantages in the degree of complexity needed for the underlying object model. The OEM is much simpler then the Garlic Object Model. TSIMMIS is also very good at managing highly dynamic Information Sources since no time is lost constantly reunifying all of the local data models to keep pace with local internal changes. TSIMMIS buys this flexibility with the need for a high degree of human interaction and effort when defining the Mediators.

Due to this requirement it seems questionable if such a system would scale well with the number of users and be up-to-date in the sense of the number of Mediators needed. The ability of the system to handle these issues is highly dependant on its ability to automatically generate Mediators based on loose definitions. This has been stated to be a design goal, but remains as yet unsolved. In the scenario decribed above, an organisation using the Garlic system to manage complex object types would have complete knowledge of the internal representations of each data provider and it would be no problem to fully describe the objects used, thus negating the advantages of TSIMMIS in this case.

The ability to define Mediators for a certain problem has some commonality to the idea of a function-provider in ObjectGlobe. Someone could write a Mediator for a specific problem area and allow others to use it. It could also be combined in a hierarchy to attack more complicated problems. However the Mediator really only deals with information queries and is therefore a much simpler and less-extendable concept than the function-provider. There is also no provision for the distribution of such Mediators, a prerequisite for shared use among many different people or organisations.

TSIMMIS has demonstrated the ability to browse through the available Mediators using a web-based interface [10], but since it lacks a global directory it seems questionable whether a relevant Mediator could be quickly located in order to deal with specific data types. Garlic and ObjectGlobe avoid this problem by maintaining global Metadata information and (in the case of ObjectGlobe) a directory service for locating global resources.

ObjectGlobe has a much broader vision than the other two systems. With its model of information, function, and cycle providers ObjectGlobe provides not only theoretical information processing advantages, but also several new business models for service providers across the Internet. It also enables both local and distributed access to information and the ability to use ObjectGlobe in parallel to any exsisting systems due to the careful inclusion of relevant security measures. Neither of the other systems considers security at all.

ObjectGlobe has by far the most extensive provisions for query optimization over all sources in the network. Garlic also includes query optimization (albeit not completely implemented) while TSIMMIS does not address this area at all. The query optimizations together with ObjectGlobe's ability to transport not only data, but also functions and computation time provide it with the most flexible (and probably the fastest, if that could be reliably tested) architecture of the three systems. That these provisions can make a large difference in the performance of the such a data integration system was demonstrated in [1].

In summary, Garlic appears most viable for the management of complex data objects within a business or organisation, TSIMMIS seems most applicable when the type of specific service is known, i.e. "We need a Mediator to provide us with current affairs information," particularly if the data source is unstructured or very dynamic, and ObjectGlobe seems to provide the most comprehensive collection of tools to enable realistic distributed information processing at the current state of the art.

References

[1] Reinhard Braumandl, Markus Keidl, Alfons Kemper, Alexander Kreutz, Stefan Pröls, Stefan Seltzsam, and Konrad Stocker. *ObjectGlobe: Ubiquitous Query Processing on the Internet*, Internal Technical Report, November 1999.

[2] R. Braumandl, A. Kemper, D. Kossmann, A. Kreutz, S. Pröls and K. Stocker. *ObjectGlobe: Integrating Data Processing Services on the Internet*, Workshop, October 1999.

[3] M.J. Carey, et. al. *Towards Heterogeneous Multimedia Information Systems: The Garlic Approach*, RIDE-DOM 1995.

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. *The TSIMMIS Project: Integration of Heterogeneous Information Sources*, In Proceedings of IPSJ Conference, pp. 7-18, Tokyo, Japan, October 1994.

[5] W.F. Cody, et. al. *Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project*, Visual Database Systems 1995

[6] L.M. Haas, et. al. *Transforming Heterogeneous Data with Database Middleware: Beyond Integration*, Data Engineering Bulletin 1999.

[7] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. *Extracting Semistructured Information from the Web*, In Proceedings of the Workshop on Management of Semistructured Data. Tucson, Arizona, May 1997.

[8] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. *Integrating and Accessing Heterogeneous Information Sources in TSIMMIS*, In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford, California, March 1995.

[9] The Garlic Homepage. *http://www.almaden.ibm.com/cs/garlic/homepage.html*

[10] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. *Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System*, In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995.

[11] The ObjectGlobe Homepage. *http://www.db.fmi.uni-passau.de:8000/projects/OG/*

[12] The TSIMMIS Homepage. *http://www-db.stanford.edu/tsimmis/*