UNIVERSITY OF GOTHENBURG

Mediation systems

An approach to retrieve data homogeneously from multiple heterogeneous data sources

Jonas C. Ericsson <ericssoj@ituniv.se>

Bachelor of Applied Information Technology Thesis

Report No. 2009-051
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Göteborg, Sweden, May 2009

Abstract

Modern IS/IT systems tend to use several sources of data and many developers process the this data manually. A homogeneous system for processing the data will make their systems less error-prone and reduce the time to market. This article will present an architecture and a query processing technique for homogeneously retrieving data from multiple heterogeneous data sources. The approach presented has studied several similar systems, i.e. Garlic, TSIMMIS, SIMS and Starburst. These systems solves similar problems to ours in various ways. It turns out that a higher abstraction and a constructional processing technique in combination with a mediation architecture is a solid choice for homogeneously retrieving data from heterogeneous data sources.

1 Introduction

Have you ever created an object by using data from multiple data sources? Was your solution generic enough to create any other object by just adding a description of the data and point where it is stored? If you were to create a new type of objects you would probably have to develop similar code for this certain type. It is likely you would invent the wheel again to solve almost the same issue. Sure, you could refactor your code according to DRY[13] principle. Though, you will still create a lot of custom code for every object type you want to work with. This problem is likely to be more common in the future, service oriented architecture has been around for a while and cloud computing is on the rise.

Retrieving data from multiple sources can be both time in-efficient and error-prone. The three causes behind these issues are; *first*, the retrieval step where the developer writes different methods to retrieve data from each data source. *Second*, processing the data to filter out superfluous data. *Third*, merging the data from multiple sources into an integrated result set.

Developers are required to have detailed knowledge about their heterogeneous data sources. The capability sets, query languages, domain models or the procedure of processing data may be different for each source. This is not a problem with a homogeneous system since the integrated domain model shares a common set of capabilities and uses only one query language. One common situation is to have several different databases which you have to combine results from. If you are lucky you may have a homogeneous set of DBMS servers but you will still have different domain models in each database to query and merge results from. Another common situation is to combine data from databases and data from files. You either have a different storage model, capability set or query language that requires even more processing from the software and knowledge from the developers. The latter situation does also require matching between data types but there is no guarantee that the data types from different data sources will be compatible.

This article will investigate how to present multiple heterogeneous data sources in a homogeneous way. We will focus on the architecture and query processing, we will also investigate how to make use of the ODMG Object Query Language(henceforth OQL)[6] to retrieve data from a homogeneous database middle ware. These systems are commonly called *mediators*. The architecture will be integrated into an existing system developed by a third part. The architecture shall be able to fill the semantic gap between class models and storage models. Garlic is a similar system developed for DB2[1] in the early 90s, has influenced many of the choices during the development. Another system which has had great influence is Hibernate[2] which is a object relation mapping API. The disadvantage with Hibernate is the lack of support for multiple data sources. This is the problem our solution will solve.

The solution to this problem will make retrieval and merging of data from multiple heterogeneous data sources less error-prone. Developers will only have to keep one data model, one query language and one set of capabilities in mind compared to when doing it manually when the developer must have detailed knowledge about each data source. The productivity is likely to increase since the data retrieval tasks will now be implemented faster so one can work with other features that should be implemented.

[1]http://www.ibm.com/db2/
[2]http://www.hibernate.org

2 Research method

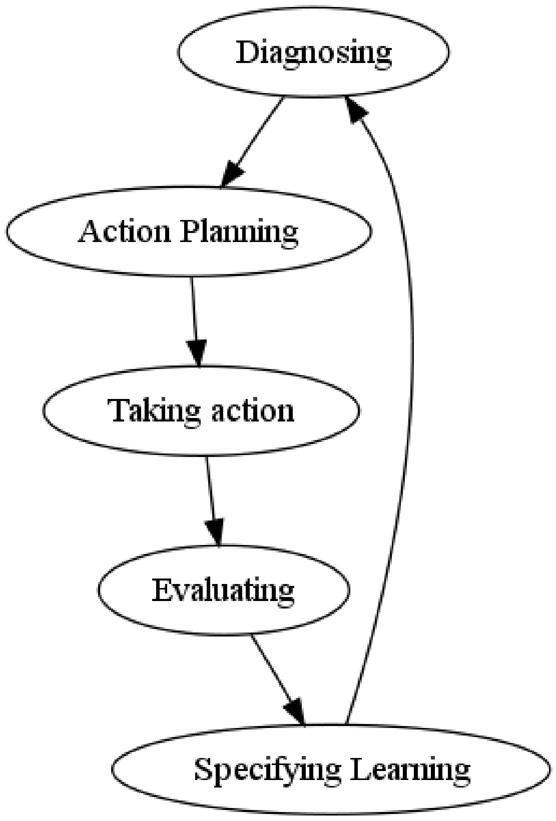

Figure 1: Action research model

Action research was used throughout this project because it uses an iterative approach[17][3]. Its flow is showed in figure 1. An iterative approach was preferable so the problem could formally be split up in phases, much like a agile software process which are proven to work well for software development[16]. Action research is however not bound to software project and can be applied to all kinds of research[2]. Some argue that action research is an academic way of consulting due to the close collaboration with the industry but Baskerville has distinguished the differences for us[3]. For instance, Baskerville states that action researchers contribute to both research and his client in contrast to a consult who is only interested in his client. This project did not have a lot of similarities with consulting since the stakeholders only provided input on requirements and assisted a few times with possible solutions. The action research was applied more towards a design driven action research approach. There are three more benefits with action research. *First*; action research focus on change[2], reflection[3] and experiments[17]. The possibility to look back and reflect on the outcome from an iteration instead of doing a big bang approach, will probably avoid bad decisions for the upcoming iteration. *Second*, action research focus on applied theories to validate if it is applicable for the concerned case. Third, merely an extension of the previous arguments about iterations; iterations allows decisions to be postponed until later iterations so no premature decisions are made.

This research project followed a pragmatic philosophy. Pragmatism enables mixed method research and allows both qualitative and quantitative inquiries[8]. The pragmatic view was primarily chosen since it is problem-centered and real-world practice centered which makes it suitable with action research which is more or less the corresponding research method. An alternative to pragmatism would be postpositivism since the observations of a theory could be a good way of problem solving[8]. It does however appear like a postpositivism philosophy is less inviting to change than pragmatism which tends to be useful in changing environments and software requirements changes all the time[13]. Using postpositivism might have lead to the development of an theory that is proven false instead of adapting while doing the research do ensure success.

3 Related work

The research in retrieving data from multiple heterogeneous data sources started in the late 80's. Many of the ideas used today was presented by Gio Wiederholds paper published in 1992[21]. He stated the term mediator as a system which integrates data from various data sources which were easy to extend. Such mediators were later developed, the ones which has influenced this project are; Garlic, TSIMMIS, STRUDEL, SIMS and Starburst. But first we must present a few concepts behind query processing so we can properly present and compare the different mediators.

Lohman research contributed much in the area with his algorithm called STrategy Alternative Rules(henceforth STAR)[15]. His query processing algorithm has been widely used for mediator systems. STAR is a constructive algorithm and constructive does in this case mean to build a new query tree from the retrieved input. Traditionally you would use a transformational processes to alter your query tree rather than constructing a new one, the latter has however been successfully used in Garlic[19] and TSIMMIS[7] which are two systems with a lot in common to ours. The algorithm is cost- and rule-based solution that constructs a set of alternative rules and picks the one with lowest cost to execute. Three different types of components are used to build these rule sets. *First*, Constants, a constant presents an attribute that are to be retrieved, it can also be values in predicates. For instance, in the query *SELECT address FROM Patient p, X-Rays x WHERE p.birthnumber = p.patient* consists of two constants, the attribute birthnumber in Patient and the attribute patient in X-rays. *Second*, Terminals are in fact an extension of relational algebra. Terminals have been extended with non-relation algebra operators, for example SORT for sorting information. *Third*, Non-Terminals are a higher abstraction of Terminals which has been grouped together. They are connected to one another in a tree.. Some terminals require its input data to undergo certain preprocessing steps before it can be handled properly, such preprocessing steps are called *Glue rules*[15][12]. Consider once again the query *SELECT address FROM Patient p, X-rays x WHERE p.birthnumber = x.patient*, where you have to use JOINs to process the data. Joins can be processed in different ways and in this example we use Merge-Join which requires the input to be ordered by the join attributes. A SORT rule would be added to the rule set to fulfil this requirement and this rule would be called a *glue rule*.

Garlic is a fully-fledged mediator that uses STARs in its query processor[10][5][19]. The Garlic project was started about 1990 in IBM Almedan Research Center. The goal of the project was to build an extensible middle ware system for homogeneously retrieving data from various data sources. The four different concepts of garlic are; *first*, A layered architecture with separate objects for the integrated schema, query processor and the different wrappers are each connected to a data source. *Second*, Wrappers. Wrappers are objects whose task is to convert queries from the mediator to its data source native query language and convert the results received into a desired class model. *Third*, Extending[12] Lohmans STARs with a generic solution which works well for multiple data sources. *Fourth*, usage of the ODMG standards. ODMG has developed standards for an object oriented query language(OQL) and a corresponding data model. Garlic uses subsets of both these standards for its query language and integrated schema.

TSIMMIS is another complete mediator[14][7] but with a different approach than Garlic. Interestingly enough, both Garlic and TSIMMIS were developed at the same research center during the same and they have assumingly affected one another because they share several concepts. Differences do however exist, TSIMMIS uses plain STARs without extensions in comparison to Garlic. The query languages does also differ, hence TSIMMIS uses a different set from the ODMG and SQL standards.

STRUDEL is a mediator extended with content management for web pages[9]. The architecture of STRUDEL is similar to TSIMMIS but the content management extension has added additional modules. It is however one of very few mediators that presents an integrated solution. This solution does also use a domain specific language extended with features to easily create web pages from the retrieved content. Although, it is not known which query processing technique they have used in STRUDEL or how their wrappers work in detail.

SIMS is a mediator with a domain specific query language[1] and, in contrast to the other mediators, uses a pipeline architecture. The domain specific query language uses an hierarchy between objects to see what different data each object can store[1]. It also enables casting between types, thus B inherit its properties from A, thereby you can cast object of type B to an object of type A if it creates a more efficient query. The query processes using a different technique than the previously described STARs. Instead, it works directly on the query by using templates of rewrite rules; a template can for instance show when it is more efficient to cast an object type in a query. In addition it does not use a layered mediator architecture, instead it uses a pipeline mediator architecture. The main difference between the architectures is that the pipeline architecture rigid flow of data, it strictly follows $sourceselection \rightarrow accessplanning \rightarrow rewriting \rightarrow execution$ and for each step gathers some additional meta data from the sources.

Starburst is just query processor and not a full mediator. Despite the fact it is not a full mediator it is still interesting. Starbursts query processing technique is similar to Garlics technique. Both uses a rule-based system but Starburst does in contrast to Garlic use a transformational approach and is not cost-based[18]. The rules are built up in a similar way to SIMS rules, Starbursts rules are however better explained and expressed with pseudo code. Startbursts and SIMS architectures are alike, Starburst is however more thorough with steps that neither of the previously mentioned projects has brought up. It has included steps for lexing, semantic checking and query evaluation before a query is executed. These steps are used to ensure the correctness of a query throughout the pipeline. Starbursts success has lead to a working prototype and an integrated implementation in IBM dB2.

We will in the next section see how we have used, extended or combined these techniques. The overall architecture is borrowed from Garlic and TSIMMIS but some extensions were made to decouple the design from domain specific details. STARs will be used for query processing but here we combined Lohmans solution and Garlics solution and put some metadata in the integrated schema instead of having it in the different LOLEPOPS.

4 Architecture

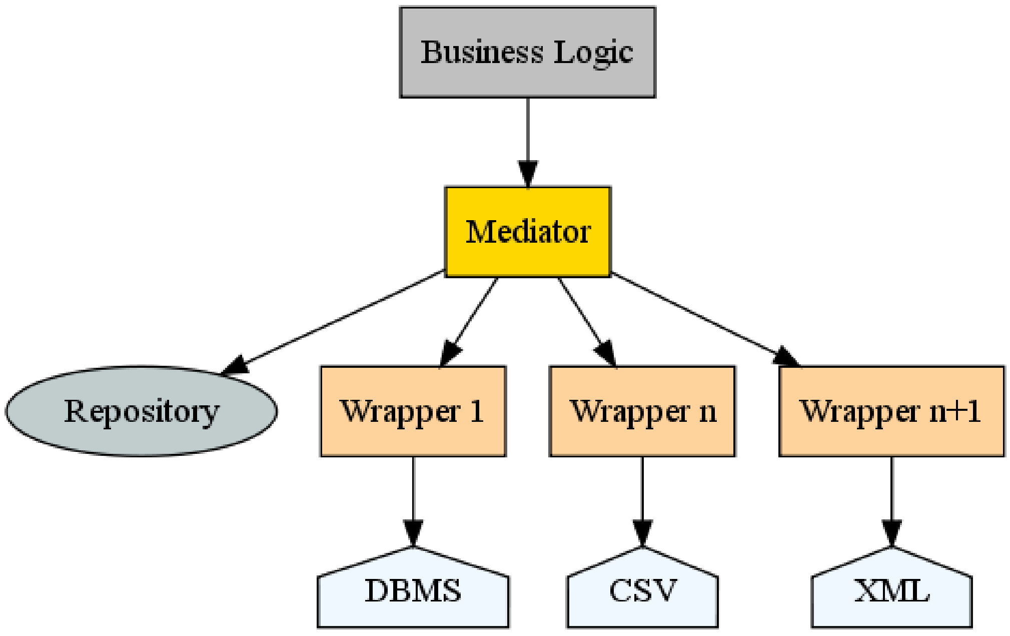

Figure 2: System architecture

Our mediator system uses a layered architecture with wrappers, as shown in figure 2, to increase extensibility. A layered mediator with wrappers is more extensible than a pipeline mediator, wrappers can be easily added to extend the functionality in a layered mediator in comparison to a pipelined where one must also modify the query processor. Thus developers must deal with at least two subsystems in a pipeline architecture, the wrapper and the query execution module. This requires more knowledge from a developer compared to if the developer were to implement a wrapper for a layered mediation architecture. It is however not likely that any significant differences in performance exists between a pipeline mediator and a layered mediator architecture. Performance has tighter coupling to the query processing than the architecture which merely presents the structure of a system.

The different modules of our solution are described below.

Business logic The business logic is generally an encapsulated user application for solving organizational problems or supporting organizational processes.. It uses the mediator system to retrieve data.

Mediator The mediator is responsible for processing queries, merging results and keeping track of available data sources. The mediator consists of a query processor for constructing query trees and merging data, it is further described in section 5.

Repository The Repository stores an integrated data model gathered from the available data sources. The repository is used to determine relationships between different objects. The objects may be located in different data sources. Thus must each object also contain information about the data source it belongs to.

Wrappers The wrappers are a homogeneous interface for working with heterogeneous data sources. It provides functionality to extract the schema from a data source in a usable format for the repository. A wrapper does also provide a parser used to execute queries towards its data source. The mediator will use one wrapper for each data source and its schema will be integrated in the repository.

Data sources A data source is where data is stored. It can be a DBMS or as shown in figure 2 a CSV or XML file. Nothing limits the data sources to be of other types as long there is a wrapper that can handle the format of the data source.

A wrapper has, as in Garlic[19] and TSIMMIS[7], two responsibilities. *First*, it is responsible for retrieving and parsing the schema from a data source. This may be done in various ways, i.e. a database management system has a schema for each database which makes it easy to retrieve. Once retrieved it is parsed into a usable format for the integrated schema in the repository. Working with XML-files are fairly similar, you have a schema but you have to do a couple of extra steps to extract it. You must first store the data in the memory and thereafter analyze its structure to extract the schema. It is thereafter stored in the integrated schema. A CSV-file does not have an explicit schema in its content so one must either hard code the schema into the wrapper, or use an external file containing the schema.

An external file containing the format of a schema is called *template*. We will present what a template is, what they are good for and how they are solved in this solution. A template is a file for presenting how the schema of a certain data type is formated. For instance, a CSV files template shows in what order values should be bound to different attributes of an object. "Put Abstractions in Code, Details in Metadata" as proposed in The pragmatic Programmer[13], decouples your details from your design and makes it more adaptable. A metadata-driven solution makes it easier to extend a wrapper with a richer set of configurations. The wrapper does not have to be recompiled if there is a defect in the template since it has been decoupled from the binary. A wrapper is ready for use once it has been configured with a template and conversions from an external schema to the integrated schema can be performed. Templates are not bound to CSV-files specifically, they can be used for any types as long there is a wrapper for processing the templates, schemas and data.

The repository provides the integrated schema to the mediator. It contains models of objects, relationships between the different object models and bindings to where the models can be retrieved. Our repository has chosen to follow the same approach as Garlic and TSIMMIS and uses a subset of the ODMG Data Model to model objects in the repository. This standard is used as it

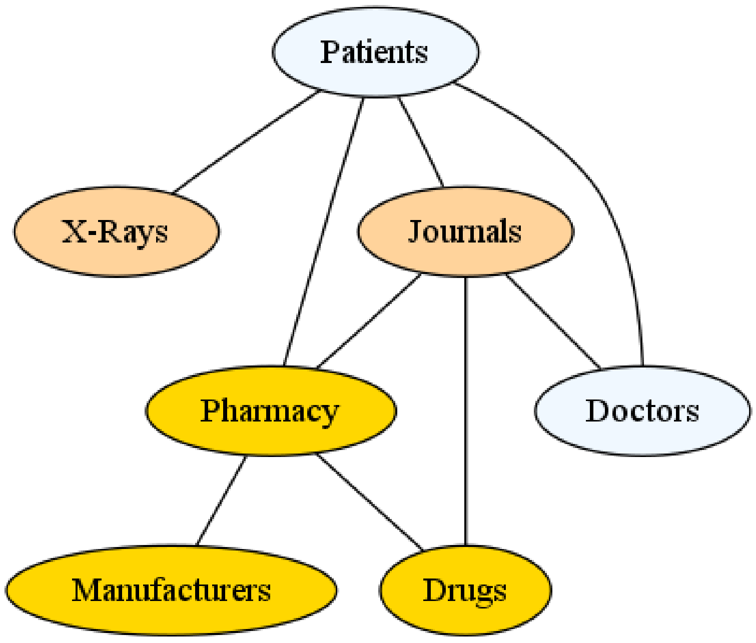

Figure 3: Example data model

also has the OQL as another part in the same standard and they are tightly connected. OQL is the preferred query language by the stake holder of this project. Although, some extensions have been made since the ODMG Data Model does not store where the modelled objects actually are stored. These bindings are stored by storing an attribute assigned an unique identifier assigned to each wrapper. This identifier is later used to validate if two models exists in the same data source or not. An example data model is shown in figure 3, this is just a fictitious model for demonstration. Nodes with identical colors are stored in the same data source.

Data sources can be positioned at different physical or logical locations. Data positioned at different logical locations are trivial, different physical locations can on the contrary be troublesome. Interruptions, corrupt data and latency are the potential problems for us, we will however only look in depth at the interruption issue. Interruptions of power or network connectivity can make a data source unavailable to our mediator. Imagine figure 3 as an unsynchronized data model where the wrapper representing X-Rays and Journals has lost connectivity to its data source. The query *SELECT address FROM Patient p, X-Rays x WHERE p.birthnmbr = x.patient* can thereby not be processed properly until connectivity has been restored. We could just let our connection to the data source time out, this is however not very efficient or proactive. A proactive approach would let us know even before we execute a query if the data source is up or not. Such a solution would faster let a user know when something is wrong, sometimes even a partial result is good enough! Our solution to this issue is to use either Heartbeats[4] or a Echo[4] to get the status of your source. The heartbeat solution lets the data source send a message to the mediator regularly to verify it is online. The mediator tries to reestablish the connection once a heartbeat is not received and the data source is assumed to have lost connectivity. Heartbeats can be used by active sources, for example a DBMS or services. Echoes does like Heartbeats send a message but now the mediator sends the message instead of receiving. Echo can be used for passive data sources for instance a file or web server. It is not known why previous research has neglected this issue, perhaps it was not very common to store data at different physical locations when their research took place. This is however just speculations of what the reason could have been.

It turns out that there is no tight coupling between the architecture and query language. The coupling between the query language and query processing technique is also quite weak as long the implementation is generic enough. One example is how Garlic made STARs more generic to work with multiple sources and their query language, the processing technique although the same as

Lohmans STARs. TSIMMIS and Garlic both uses an object oriented dialect of SQL but different query processing techniques as well as some differences in their architecture. Starburst[11], a query processor, uses STARs like Garlic but does not use a object oriented language. STARs is also used in DB2 which does not have a mediator system nor an object oriented query language.

The presented solution used a layered mediator architecture and integrated the ODMG Data model 3.0 for modeling objects. Interruptions are dealt with either by Echoes or Heartbeats, depending on if the data source is active or passive.

5 Query Processing

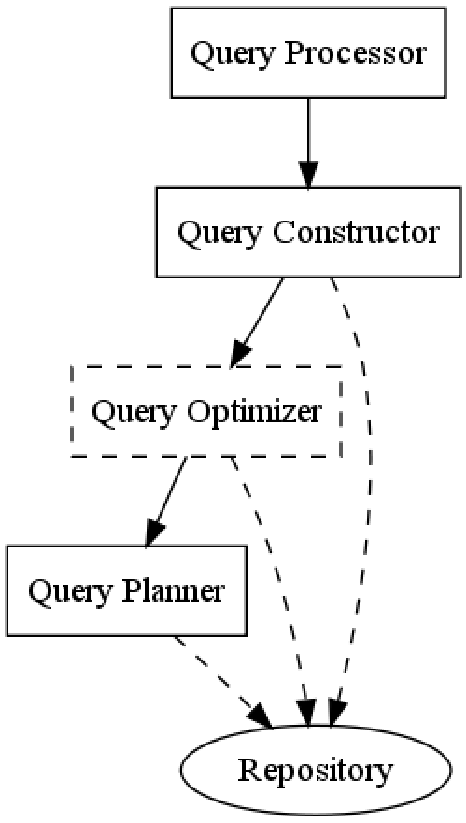

Figure 4: The query processor

Figure 4 shows the internals of the mediator module. This section will present how the query constructor and query planner works and how they handle single source and multi source queries. Query optimizations are left out at this stage of the project and it is assumed the data sources will optimize the queries passed to them. The query processor has three different parts; The query constructor that creates a new STAR from the input query. The query planner splits up a STAR into smaller parts by iterating through the STAR tree and decide to what wrapper it should be sent to.

There are several different query planning and query processing techniques. You can either use plain relational algebra, rewrite the query directly in the string or as in this project, use STrategy Alternative Rules. We use a combination of Lohmans STARs and Garlic STARs[15][12], out STARs are very similar to Lohmans STARs but it has borrowed a couple of operators from Garlic. The second difference is that the operators do not contain as much metadata as in Garlic. Having information about the data sources in an operator is superfluous since this information is already stored in our data model.

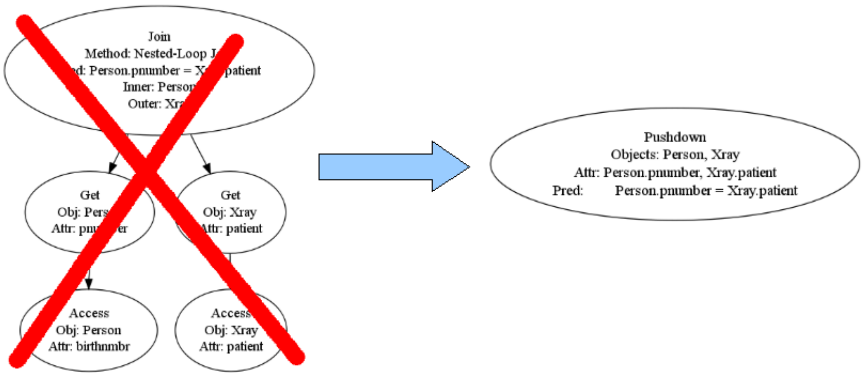

Figure 5: A single source query using PUSHDOWN.

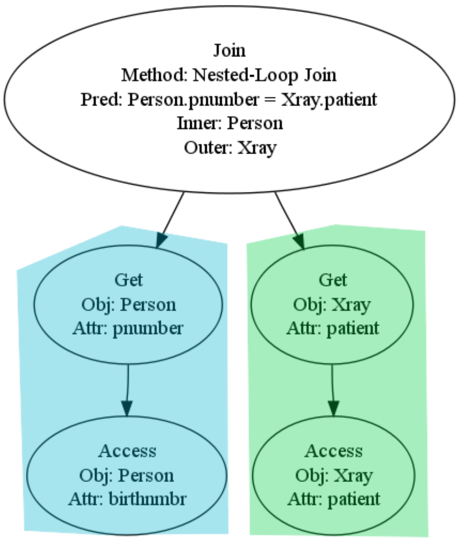

Figure 6: A multi source STAR.

A query can be split up in two different types, single source and multi source. The following examples constructs a STAR from the query *SELECT address FROM Patient p, X-Rays x WHERE p.birthnmbr = x.patient*, the data model used is presented in figure 3 but the objects have been extended with additional attributes. A single source query will however be passed directly to a wrapper after its STAR has been constructed and analyzed. Figure 5 shows how a STAR is more efficiently written as a PUSHDOWN instead of creating a complex STAR. Multi source queries requires several more steps. Figure 6 shows a query that retrieves data from two different sources and is later joined in the Mediator using a Nested Loop Join method. The performance could have been increased by skipping the STAR construction for single queries but it would also require a wrapper to have two parsers; one parser for STARs to the native query language and parser for OQL to the native query language.

Queries using multiple sources must be split up in several subqueries to be able to return a correct result. The subqueries must in some cases be processed in a specific order because of dependencies between non-terminals and siblings in the query tree. A STAR is processed bottom-up and cuts off non-terminals which uses a single source and passes it to a wrapper for retrieval of data. Another issue to deal with is lack of capabilities in some wrappers. Wrappers may have different sets of capabilities which the mediator must compensate for. A wrapper will compare the capabilities required to process the STAR with the data source set of capabilities. It should be rewritten to match the data source's capabilities if capabilities does not match The STAR is then parsed to an executable format and sent to the wrappers corresponding data source. The mediator will then perform joins on the data retrieved from the different wrappers until all STARs have been processed and it has reached the root STAR.

6 Status & Conclusion

Our solution as presented, uses a layered mediator architecture[7][19] with four modules. The mediator, repository, wrappers and data sources. The repository uses the ODMG Data Model 3.0[6] to model objects and relations but it has been extended with an identifier to bind the model to its data source. Wrappers make use of templates to decouple the schema layouts for each type, wrappers must however first be developed to support different data sources. An issue was discovered during the design, interruptions of power and network connectivity can lead an unsynchronized schema. This makes it impossible for some queries to execute properly. We use a proactive solution to verify that a data source is up instead of just waiting for time-outs. The suggested solution is to use either Heartbeats or Echoes. Previous research has not brought up this problem for unknown reasons, possibly they assumed all data sources were located locally.

The query processor uses a dialect of Lohmans[15] and Garlics STARs[12]. It was developed during this project to fit better into the extended subset of the ODMG data model. It was probably a rather successful choice since the query algorithm can now solely focus on the query and not bother about where the data is stored until the planning stage. Then we already have the data ready in our model. The model can also be used forwarded to the business logic if it for some reason would like to know where certain objects are stored. The query planning step was the toughest step to find research about. None of the existing mediators mention how they split their queries over many data sources. The bottom-up approach was found by coincidence while trying to figure out how to split the query. Just the thought "lets try it backwards" solved the problem after been prototyping a top-down approach. It has just been tested in theory and is not yet implemented. This system and the research behind it will soon be implemented and integrated into our stakeholders framework.

Many solutions propose to work with operator trees and relational algebra on a lower level than a STAR[20]. These solutions has however mainly worked with relational databases and not tried to integrate data from various types of sources. Neither has any performance measurements been found of the two so it was decided to go for an algorithm that was proven to work for similar projects, such as Garlic.

This area is likely to grow in the future, cloud computing, SOA and other technologies are becoming more popular for each day. These technologies do also often use data from diverse data sources in some way, few has a solution for mediating data homogeneously from their sources.

References

[1] Yigal Arens, Chin Y. Chee, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2:127–158, 1993.

[2] David Avison, Francis Lau, Michael Myers, and Peter Axel Nielsen. Action research. *Communications of ACM*, 42, 1998.

[3] Richard L. Baskerville. Investigating information systems with action research. *Communications of AIS*, 19, 1999.

[4] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison-Wesley Professional, 2003.

[5] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen W. Luniewski, Wayne Niblack, Dragutin Petkovic, John Thomas, John H. Williams, and Edward L. Wimmers. Towards heterogeneous multimedia information

systems: The garlic approach. *Proceedings of the Fifth International Workshop on Research Issues in Data Engineering(RIDE): Distributed Object Management.*, 1995.

[6] R.G.G Cattell, Douglas Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russel, Olaf Shadow, Torsten Stanienda, and Fernando Velez. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann Publishers, 2000.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, Y. Papakonstantinou K. Ireland, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. *Proceedings of IPSJ Conference*, 1994.

[8] John W. Creswell. *Research design, Qualitative, Quantitative, and Mixed Methods Approaches. Third edition.* Sage publications, 2009.

[9] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26:4–11, 1997.

[10] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork, Roth P. M. Schwarz, and E. L. Wimmers. Transforming heterogeneous data with database middleware: Beyond integration. *IEEE Data Engineering Bulletin*, 22:31–36, 1999.

[11] Laura M. Haas, J.C. Freytag, G.M Lohman, and H. Pirahesh. Extensible query processing in starburst. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data.*, 1989.

[12] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *In Proc. of VLDB*, pages 276–285, 1997.

[13] Andrew Hunt and David Thomas. *The Pragmatic Programmer.* Addison-Wesley, 2000.

[14] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-molina, Yannis Papakonstantinou, Jeffrey Ullman, and Murty Valiveti. Capability based mediation in tsimmis. In *In Proc. of ACM SIGMOD*, pages 564–566, 1998.

[15] Guy M. Lohman. Grammar-like function rules for representing query optimization alternatives. *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, 1998.

[16] Steve McConnel. *Rapid Development, Taming Wild Software Schedules.* Microsoft Press, 1996.

[17] Rory O'Brien. An overview of the methodological approach of action research. 1998.

[18] Hamid Pirahesh, T.Y. CliffLeung, and Waqar Hasan. A rule engine for query transformation in starburst and ibm db2 c/s dbms. *Proceedings of the Thirteenth International Conference on Data Engineering*, 1997.

[19] Mary Tork Roth and Peter Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[20] Dave D. Straube, Dave D. Straube, L. Y. Yuan, and W. Joerg. Queries and query processing in object-oriented database systems. *ACM Transactions on Information Systems*, 8:387–430, 1990.

[21] Gio Wiederhold. Mediators in the architecture of future information systems. *The IEEE Computer Magazine*, 1992.