Alamo: An Architecture for Integrating Heterogeneous Data Sources*

Daniel P. Miranker
miranker@cs.utexas.edu

Vasilis Samoladas
vsam@cs.utexas.edu

Department of Computer Sciences
and
The Applied Research Laboratory
The University of Texas at Austin
Austin, TX 78712-1188

Abstract

We are developing an architecture, Alamo, that addresses both the semantic and physical aspects of data integration. The Alamo architecture permits the interoperability of both data sources and semantic components. As a collection, the supported semantic components capture most basic forms of knowledge representation. Since the semantic integration of heterogeneous data sources requires some representation of the semantic content of the data source, the Alamo architecture forms an infrastructure for the development and possible integration of different forms of semantic integration of heterogeneous data sources.

Central to the Alamo architecture is a CORBA compliant software bus called the Abstract Search Machine (ASM). The ASM augments a simple cursor class with methods that can be used to implement the marking, memoing and learning schemes exploited by clever search algorithms. The broad claim is that high performance implementations of semantic components can all be built using a common means for accessing data. Since the output of each semantic component can be accessed by virtue of the ASM definitions the components of the Alamo architecture can be composed to resolve higher level data integration problems. Ultimately, further compositions will embody complex knowledge-bases and be able to answer high-level queries.

1 Introduction

The widespread adoption of web related technologies has spawned the motivation to immediately solve the problem of integrating heterogeneous information sources. There is a large number of contributions to the problem of accessing heterogeneous information sources. These contributions typically address special subproblems of interest, like data warehousing, (loosely) federated databases, textual and semi-structured data access etc. We claim that in all of these environments, the problem of semantic integration is addressed, at least in part, by some means of knowledge representation. Further, at this juncture the forms of semantic modeling are at least as numerous as the number of application areas;

- SQL programming is popular for Decision Support,
- deductive databases are used for data mining,
- production rules apply well to expert systems and distributed systems of constraints,
- frame systems are widely popular in knowledge representation, and so on.

*This research is partially funded by DARPA, contract number: F30602-96-2-0226 and the Applied Research Laboratories Univ. of Texas, Internal Research Development Program.

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997

(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

A goal of the Alamo project is to provide foundations, formal and engineering, for integrating heterogeneous data sources, under multiple, different information and semantic representations. A critical aspect in the experimental part of this goal is to amortize the work required to integrate data sources into each experimental system. Toward this end we have defined what may most succinctly be described as a CORBA-compliant middleware interface, coined the Abstract Search Machine (ASM). The interface comprises an augmented cursor definition, such that data sources may be accessed through a "least-common denominator". Simultaneously, the interface definition includes direct support for sifting data and methods for associating markings with individual data elements. The behavior of of the ASM cursors change, transparently, per the markings. Marking techniques, (a.k.a. learning, memoing), become part of the data abstraction. Since these techniques are endemic to high performance implementation of knowledge-based systems both data processing methods and knowledge-base inference methods, or in our parlance *semantic modules*, may all be constructed using a common interface. Thus, the ASM will serve as the basis for interoperability among data sources and knowledge-base techniques.

Another starting point of the project is that we assume that there is no cooperation from the component databases beyond having made their data available on-line. Thus there is no requirement that the owners of the component databases provide information about the content and structure of their databases (meta-data). If an ontology or other aspect of distributed agent approaches is available for a data source, we anticipate that we will have a semantic component that may exploit that extra information directly. However, we do not view organizations that provide data as fundamentally altruistic or as having the extra people-related resources to build an agent infrastructure. Thus, we seek to empower a user, or at least his organization, to resolve semantic conflicts among data sources.

Most of the proposed system designs adopt a hierarchical structure, where individual layers of the hierarchy decompose, translate and optimize application requests. One of the main innovations in the Alamo architecture is the adoption of a recursive structure, rather than a hierarchical one. Since the data provided by a semantic module may itself be viewed as a data source, it follows that the output of semantic modules may be accessed through the ASM definition. Thus, we claim that Alamo further provides for interoperability among semantic modules. The advantages of interoperability include the ability to develop more complex applications, rapid prototyping, and use of existing development tools.

The ability of Alamo to host divergent semantic models and integrate a wide range of data sources, does not compromise clean system design and modularity. An Alamo implementation remains a tightly integrated system. Concessions made in our current effort in order to promptly achieve design goals, include

1. Updates to the component databases are not supported.
2. There is no concept of "global transactions", which implies that on-line processing is not straightforwardly supported.

2 Architectural Overview

The structure of the Alamo architecture is shown in Fig.1. The circles represent processes we call data pumps. Not illustrated is that behind the ASM is a cache manager that replaces the buffer and storage managers of a conventional database system and a directory service manager which includes both broker and mediator services.

2.1 Data Model

The Alamo data model defines two types of entities: objects and collections.

Alamo objects are complex objects, i.e. contain set-valued attributes and references to other objects (not through identity, but through physical pointers). Objects *do not have identity*, but they do have type. There is a class hierarchy, encapsulating object properties and behavior. Each object is an instance of a class. In other words, Alamo classes are also ADTs, and the data model is "object-relational." Dynamically created classes are, by necessity, allowed in order to represent the results of ad-hoc queries.

Collections are logical sets (or multi-sets) of objects. Collections can be extensional, i.e. defined by their contents, or intentional, i.e. defined by a specification. This specification, which can be a query, a datalog program, or any other semantic specification is expressed in terms of semantic module programs.

2.2 The Abstract Search Machine

The Abstract Search Machine (ASM) is an augemented cursor interface. The primary cursor operations, i.e. Open(), Next(), EOF(), provide a least common denominator into which virtually all potential data sources may be cast. Once data is loaded in the system a richer abstraction is exploited. The additional power falls into two categories, access and marking.

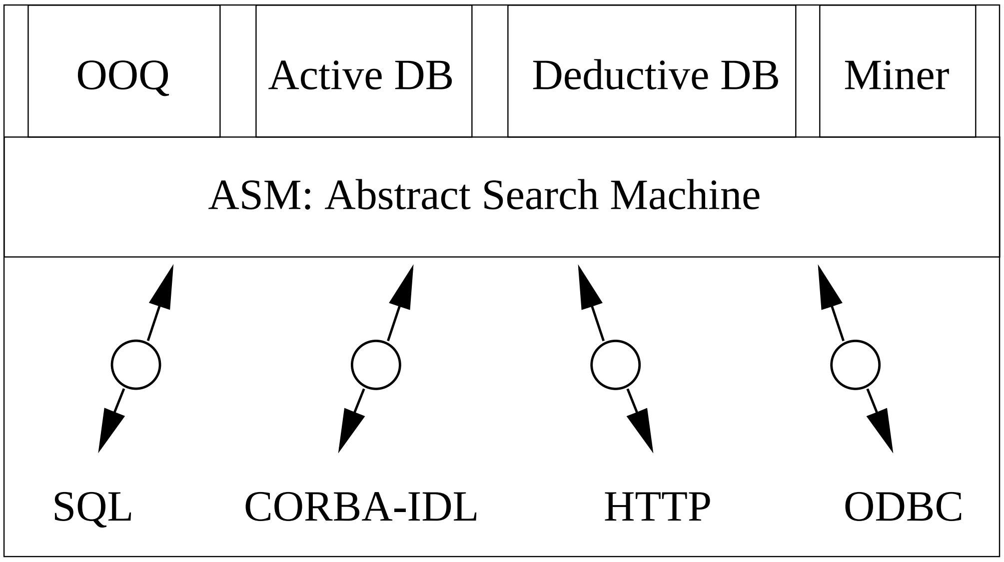

Figure 1: The Alamo architecture. Boxes above the ASM represent semantic modules. Circles represent data pumps.

The added access facilities comprise the following. When a cursor is opened it may be optionally qualified by a selection predicate and/or an ordering qualification. Similarly the next() method may optionally be qualified by a select predicate. These abstractions have been borrowed from the Genesis extensible DBMS, a component based DBMS. The same abstractions have been further exploited in the definition of several follow-on systems and proven apt in several applications including the implementation of forward-chaining rule-systems[He93].

We use the term marking as it is the term coined by Stonebraker as a category in a taxonomy of approaches to the implementation of rule systems in databases[Sto92]. Consider that to solve many problems multiple passes may have to be made over the same data set and instance specific relations between pairs of data sets may be discovered. For example, a record in one table may join with a particular record in another table, or the opposite may be true. A record in one table may not join with any records in the second table. In the latter case, we may conclude that that record need never be examined again. Thus, the record may be marked and for the duration of that task it need not be returned by the iterator. In the former case, the record in the first table may be marked with pointers to all the joining tuples in the second table. The search literature refers to these markings as "no-goods" and "goods" respectively. A collection of "goods" is known as a good list and can be viewed as a kind of denormalized join-index.

It happens that these marking primitives have been exploited for the development of good, if not optimal algorithms, in each of relational query possessing, object-oriented query processing, forward-chaining rule systems, logic programming and truth maintenance systems[De90, BaMi94, SaMi96, He93, KuLi88]. Depending on the domain the primitives may be called, marking, learning, memoing or justification.

2.3 Data Pumps

The purpose of the data pumps is to overcome the impedance mismatch between clever main-memory implementations of knowledge-based components which deal with data one element at a time and remote data sources whose access suffers from long latency and is best optimized around large block transfers. The idea is that a data pump is an active element, in a separate process, that prefetches data ahead of its use. Discussions of middleware commonly refer to establishing a "pipe" as a connection to a component database. Since pipes are passive elements we prefer the term "data pump".

Since data pumps do little more than pull data streams into the cache manager they can easily accommodate data from a variety of external data sources. The structure and functionality of an individual data pump depends on the type of data source it accesses. Should a data source be a conventional database server, the selection and sorting qualifiers specified at the level of the ASM may be pushed to the component database. We expect such decisions to be made by the Directory Service Manager in a manner similar to access path selection performed by a query optimizer.

Notice that joins may not, dynamically, be pushed down through a data-pump into a component database. If system optimization calls for and allows joins to be executed by component databases, a view may be defined on the component database and treated as another information source. Disallowing adhoc execution of joins on component databases simplifies the system considerably and defers to the priority of the transaction processing load in a live OLAP environment.

We are investigating if the data pumps should have a role at a low-level of semantic integration. It seems to make little sense for a stream of data to be loaded into memory in an incorrect physical format. Until a pump is implemented, it will be unknown if the process communication and buffering primitives integrate in a way for us to avoid that step.

2.4 The Cache Manager

Given that the intended behavior of the data pumps is to mask latency through extensive prefetching, substantial buffer space is required including managing the overflow of buffer space to secondary storage. Potentially the storage capacity of an Alamo platform may have to match that of a conceptually equivalent centralized data warehouse.

Depending on the precise use and nature of a data stream different indexing methods may be appropriate. Thus, the cache manager serves as much more than a simple buffer manager. Sophisticated implementations of the cache manager may even dynamically restructure the data. For example an ASM cursor may be opened on a collection such that it provides no qualification on the data. Thus, it is most likely a simple sequential scan of the data will ensue and the data is stored in a sequential structure. A second cursor on the same collection might then be opened such that it specifies an ordered-by attribute. At that juncture an ordered index might be built. But there are several many different circumstances, each one demanding a different structure. To consider just a few:

- The first cursor may already be closed, the collection completely present in the cache and the collection fits into main memory. Thus, a main-memory sort is most effective means of organizing the data.
- The data pump instantiated for the first cursor happened to exploit an access-path that used a sorted index on the attribute. Nothing need be done and the second cursor may start iterating immediately.
- The collection is not yet fully materialized in the cache and data is not sorted conveniently. A sorted index may be constructed, and optimized in anticipation of future inserts. The second cursor is blocked until the collection is fully materialized.

2.5 The Directory Service Manager

The Directory Service Manager (DSM) contains a system wide catalog of the collections, both intensional and extensional. The DSM acts as a broker, marrying data sources to data consumers independent of whether those sources are external component databases or internal semantic elements. In addition the DSM tracks and manages the state of the Cache Manager.

We stated that the ASM forms a software bus. However, a bus is a broadcast mechanism and in the context of software components, "bus" is a misnomer. The fact is to compute each intentional collection the computation must be connected directly to each collection that serves as an argument. If an allusion to computer hardware is to be made, then it is more correct to think of a collection of data-flow processing elements connected using a multiprocessor interconnection network. In this analogy the DSM serves as controller for the routing mechanism.

When a cursor is created on a collection, the following actions are taken:

1. The collection is looked up in the DSM.
2. If it is an extensional collection,
   (a) an appropriate physical cursor is created in the CM.
   (b) If an appropriate data pump is not in place, an access path for the component database is selected and a data pump instantiated.
3. If the collection is an intensional collection,
   (a) its specification is retrieved from the DSM.
   (b) This specification is merged with any cursor qualifications (selection predicate, ordering qualification), and the result is sent to the appropriate semantic module, for processing.
   (c) The semantic module is responsible for translating, optimizing and executing this specification, and returning the resulting data to the ASM.
   (d) The ASM buffers the data in the cache manager and returns it through an ASM cursor.

2.6 Semantic Modules

Standard semantic modules in Alamo will include (see Fig.1):

- an object-relational query engine (OOQ)
- a production rule engine (Active DB)
- a deductive database engine (Deductive DB)

We have chosen these three modules for a first implementation, because of their familiarity and wide applicability.

Processing performed by the semantic modules, is done through the ASM facilities, at every level. This statement implies a slightly different design of well-known data processing algorithms (relational join algorithms, semi-naive evaluation etc.). Our claim is that such algorithms can be expressed in a cursor-based language, together with a library of data structures. Such a library of data structures, both memory-based and disk-based ones, is implemented in the Cache Manager inside the ASM, and is accessible through ASM cursors. This somewhat unorthodox design was adopted because it is more extensible, permits the detection of race conditions and allows more clever resource allocation.

3 Data Integration in Alamo

The Alamo schema consists of the set of classes and collections defined in the system. We adopt the terminology of federated databases, and define *local schemas* and a *federated schema*.

3.1 Local Schemas

Data pumps do not perform any semantic integration processing (beyond binary translation of data). Alamo objects constructed at the data pumps resemble the structure of external data very closely, e.g. if relational tuples are received, there will be one object per tuple, having exactly the same attributes as that tuple. Still, these objects have a class. Also, objects returned by a query to an external data source are viewed as belonging to a specific collection. Thus, a (partial) schema is associated with each data pump. We adopt the term *local schema*, and use the symbol L to describe it.

3.2 Federated Schema

Data integration is achieved when the application views the data organized under a single, consistent, intuitive schema. We adopt the term *federated schema*, and use the symbol F to describe it. Notice that F subsumes any local schemas.

3.3 Mapping Languages

Each of the classes and collections in F is defined by a specification, residing in the DSM. These specifications are the results of a transformation from the local schemas $L_1, \ldots, L_n$, to the federated schema F. Such a transformation is usually expressed as a program in a *mapping language*.

The recursive nature of Alamo allows for multiple mapping languages, each used to define some part of F. A mapping language program is not simply a collection of queries defining collections, but rather a process of translating domain knowledge into metadata, that is then used by the system to perform data processing tasks. This translation of domain knowledge into metadata is called *knowledge compilation*.

4 Interoperable Semantic Processing

We expand on the notions of the previous section with a discussion of a number of semantic integration approaches, and their possible implementation in the context of Alamo. First, we give a brief example of integrating relational databases through knowledge compilation. Then, we discuss two specific systems for data integration proposed in the literature: the SIMS project, and the Information Manifold. We show that both of these systems can be integrated into Alamo.

4.1 Knowledge Compilation

A fundamental semantic module is an object-relational query engine. It provides basic access to the data in a very efficient way, which makes it the tool of choice for massive data applications, like Decision Support. There are also numerous extensions to the basic relational calculus and algebra. One of these is presented in [KLK91], where it is argued that second-order relational queries (i.e. queries over the schema, as well as over the data) are often required for data integration. We will present a short example with respect to a primary application of Alamo, the World-Wide Herbarium, in order to introduce the concept of *knowledge compilation* in Alamo.

A herbarium is a library of plant specimens. Similar to book libraries, herbaria are moving their card-catalogs on-line. In contrast to book libraries, the field notes on the "cards" indexing the plant specimens form a rich source of information. The integration of these notes from many collections promises to be a great botanical resource.

Suppose there are two herbaria in the Austin[1] area, and each of them maintains a database of the number of wild flower specimens collected in the Austin area.

[1] the capital of Texas, USA. The Austin city area includes the Travis and Williamson counties

We will denote the local schemas of these databases by L_1 and L_2. Sample databases for these schemas appear in Fig.2.

We would like to access all the data through a single table, with the schema of Fig.3 Observe that both

Figure 3: Federated schema of the herbarium example.

Federated Schema F

AllFlowers:

Species	County	Specimens
lily	Travis	5
⋮	⋮	⋮

of these local schemas have semantic information (the county) embedded in the schema. This is domain knowledge, known to the user. This knowledge can be used to derive an SQL query, that will serve as the specification of collection `AllFlowers` in the federated schema. In general however, the user should be assisted by a high-level tool, that will allow him to express this knowledge succinctly. For example, the user could write a high-order SQL query:

```
counties = { Travis, Williamson };

SELECT *
FROM t IN
  (SELECT t.Species, name(c), t.Specimens
   FROM c in COLLECTIONS counties,
        t IN c)
  UNION
  (SELECT t.Species, name(a), t.a
   FROM  t IN Flowers,
         a IN ATTRIBUTES counties OF Flowers)
```

The above program fragment would be a program in a *mapping language*. The program would be compiled, and appropriate SQL code would be produced, for the specification of collection `AllFlowers`. This specification would be stored in the DSM, as metadata.

4.2 The SIMS project

The SIMS project ([AKS96]) approaches the data integration problem by building a domain model for the application which integrates all information sources by describing the semantic interdependencies between them. Then, for each query submitted to the system, a query plan is computed using AI planning techniques such that the user query is rewritten into subqueries targetting the appropriate component databases. The domain model in SIMS is built using the LOOM knowledge representation language. The query language of SIMS is an object-oriented query language.

All of the SIMS components have to be mapped to Alamo. First, the domain model, expressed in LOOM, has to be compiled. Information sources in the domain model would simply be Alamo collections. These of course, would not necessarily be external database relations, but could themselves be computed inside Alamo, by other semantic modules. The generation of the domain model is a knowledge compilation step, thus a suitable LOOM compiler has to be built.

A semantic module would also have to be built for SIMS. This module would implement the runtime functionality of the SIMS system, and in particular query reformulation, optimization and execution. This semantic module would then be used for two purposes: answering ad-hoc queries, and defining new (intensional) collections through queries, that would be available to other semantic modules in the system.

4.3 The Information Manifold

The Information Manifold (IM) ([LRO96]) is a highly scalable system for the integration of WWW databases. A major issue resolved by IM, is the fact that most WWW databases have very limited capabilities in terms of the queries they will accept. This factor makes it difficult to integrate such data sources into systems that assume full query capabilities for external data sources. IM is a hierarchical system, and as such it can be integrated into Alamo, and interoperate with other types of systems, e.g. a loose federation of relational databases.

The implementation of IM in the context of Alamo has to define the mapping between IM queries and ASM collections. This is indeed straightforward, since IM queries are expressed by deductive rules. The IM has its own concept of a federated schema (called the *world view*), which consists of a number of IM classes, and a number of IM relations. Each of these classes and relations naturally defines an ASM collection. This definition is a knowledge compilation. Thus, IM's world view would be part of Alamo's federated schema.

In addition to the definition of the world view, there is another piece of metadata produced by knowledge compilation, called *source descriptions*. This is a set of specifications for each of the information sources, used in query plan generation, that allows IM to access data sources of limited query capabilities. At this point, we should mention that Alamo does not address this issue explicitly. In other words, there are no assumptions in the architecture, or in the data model, about the ability to open arbitrary cursors (e.g. with arbitrary selection predicates) to a collection.

In IM, interaction with external sources is done through *interface programs*. In Alamo, this role is

Figure 2: The local schemas of the herbarium example

Schema L_1

Flowers:

Species	Travis	Williamson
lily	5	11
daisy	12	9

Schema L_2

Travis:

Species	Specimens
rose	9
tulip	14

Williamson:

Species	Specimens
rose	9
tulip	14

played by data pumps. The two components are very similar: they both have the ability to formulate queries in the source's native language using query templates, and they both have the ability to cast received information into the system's binary format.

The main functionality of IM would be implemented by a semantic module, which would perform plan generation and execution, as described in [LRO96].

5 Conclusion and Status

Our primary observation is that despite different terminology there is a common collection of techniques that are often used in the construction of knowledge-based systems and that these techniques are expressible in terms of a simple extension to standard cursor definitions. We are exploiting this observation toward the development of a multi-purpose platform with the intention of supporting multiple mechanisms in a single integrated system. Existing systems usually concentrate in one area of the space of possible applications. For example, a system may perform very sophisticated data integration, but only allow browsing of the resulting data. Another system may support sophisticated processing of the data (e.g. statistical analysis), but only operate with high-quality and/or high-speed data sources. Our hope is that Alamo will serve as a basis for establishing much larger collaborative efforts.

We are underway in the construction of simplified data pumps and the integration of the ASM interface with a query engine and the Venus rule language compiler [MiOb96]. Inspection of the Coral deductive database system reveals a promising integration root with the ASM. Since both Venus and Coral are reasonably mature systems we anticipate a much more modest implementation effort than may first come across.

References

[AKS96] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2/3):99–130, 1996.

[Baetal94] D. Batory, V. Singhal, J. Thomas, S. Dasari, B. Geraci, and Marty Sirkin. The GenVoca Model of Software-System Generators. *IEEE Software*, September 1994

[BaMi94] R. J. Bayardo-Jr. and D. P. Miranker, An Optimal Backtrack Algorithm for Tree-Structured Constraint Satisfaction Problems, *Journal of Artificial Intelligence*, 71(1), 159-181, 1994.

[De90] R. Dechter, "Enhancement Schemes for Constraint Processing", *Journal of Artificial Intelligence*, 41:273-312, 1990.

[He93] T. Hetherington, "On Intelligent Backtracking, Shallow Learning and Produciton Systems", M.S Thesis. Department of Computer Sciences, University of Texas at Austin 1993.

[KLK91] R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. *SIGMOD Record*, 20(2):40–49, 1991.

[KuLi88] V. Kumar and Y-J. Lin, "A Data-Dependency-Based Intelligent Backtracking Scheme for Prolog", *The Journal of Logic Programming*, 5:165-181 1988.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source specifiers. In *Proc. of the 22nd VLDB Conf.*, 1996.

[MiOb96] Daniel P. Miranker and Lance Obermeyer. An Overview of the VenusDB Active Multidatabase System. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, Kyoto, Japan, December 1996.

[SAD+94] Ming-Chien Shan, Rafi Ahmed, Jim Davis, Weimin Du, and William Kent.

Pegasus: A heterogeneous information management system. In W. Kim, editor, *Modern Database Management - Object-Oriented and Multidatabase Technologies*, pages 664–682. Addison-Wesley/ACM Press, 1994.

[SaMi96] V. Samoladas, D.P. Miranker, Loop optimizations for acyclic object-oriented queries, Technical Report 96-10, Dept. of Computer Science, University of Texas at Austin.

[SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *Computing Surveys*, 22(3):183–236, Sept. 1990.

[Sto92] M. Stonebraker. The Integration of Rule Systems and Database Systems. *IEEE Trans. on Knowledge and Data Engineering*, 4-5:415–423, Oct. 1992.

[Ull88] J. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, Inc., 1988.