AMOS - An Architecture for Active Mediators

by

Gustav Fahl
E-mail: gusfa@ida.liu.se

Tore Risch
E-mail: torri@ida.liu.se

Martin Sköld
E-mail: marsk@ida.liu.se

Abstract

AMOS (Active Mediators Object System) is an architecture to model, locate, search, combine, update, and monitor data in information systems with many work stations connected using fast communication networks. The approach is called *active mediators*, since it introduces an intermediate level of 'mediator' software between data sources and their use in applications and by users, and since it supports 'active' database facilities. A central part of AMOS is an Object-Oriented (OO) query language with OO abstractions and declarative queries. The language is extensible to allow for easy integration with other systems. This allows for knowledge, now hidden within application programs as local data structures, to be extracted and stored in AMOS modules. A distributed AMOS architecture is being developed where several AMOS servers communicate, and where queries in a multi-database language are allowed to refer to several AMOS databases or other data sources. An overview is made of the architecture and components of AMOS, with references to ongoing and planned work.

Also in the Proceedings of the International Workshop on Next Generation Information Technologies and Systems, NGITS '93, Haifa, Israel, June 1993.

1 Introduction

Future computer supported engineering, manufacturing, and telecom environments [Lob93, Imi92] will have large number of workstations connected with fast communication networks. Workstations will have their own powerful computation capacities which store, maintain, and do inferences over local engineering data- and knowledge-bases, or *information bases*. Each information base is maintained locally by some human operator and is autonomous from other information bases. Each information base will need a set of DBMS capabilities, e.g. data storage, a data model, a query and data modelling language, transactions, and external interfaces. The classical relational database languages are not powerful enough for the manipulations needed, e.g. to build advanced models to filter and extract interesting information. Facilities are also needed to support 'reactive' applications that sense changes in information, i.e. *active* database facilities [DE92].

The AMOS (Active Mediators Object System) architecture uses the *mediator* approach [Wie92] that introduces an intermediate level of software between databases and their use in applications and by users. We call our class of intermediate modules *active mediators*, since our mediators support active database facilities.

We have identified four classes of mediators needed in our architecture, which will be explained more in detail in the next sections:

1. *Integrators* that retrieve, translate, and combine data from data sources with different data representations.
2. *Monitor models* that notify mediators or application programs when interesting data updates occur.
3. *Domain models* that represent application oriented models and database operators.
4. *Locators* that locate mediators and data in a network of AMOS servers.

The AMOS architecture is built around a main memory based platform for intercommunicating information bases. Each AMOS server has DBMS facilities, such as a local database, a data dictionary, a query processor, transaction processing, and remote access to data sources. Central to the AMOS architecture is an OO query language, AMOSQL, that is a derivative of OSQL [Fis89]. AMOSQL supports OO abstractions and declarative queries. It is extensible to allow for easy integration with other systems. AMOS makes it possible to extract knowledge that currently is hidden within application programs as local data structures and represent it in AMOS modules. The query processing must be efficient enough to encourage the use of local embedded databases linked into applications without significant performance penalty. The query and modelling language must also be powerful enough to store complex knowledge models. Furthermore, queries should be allowed to access more than one autonomous AMOS server as well as other data sources. It should be possible to state queries using the same multi-database query language independent of

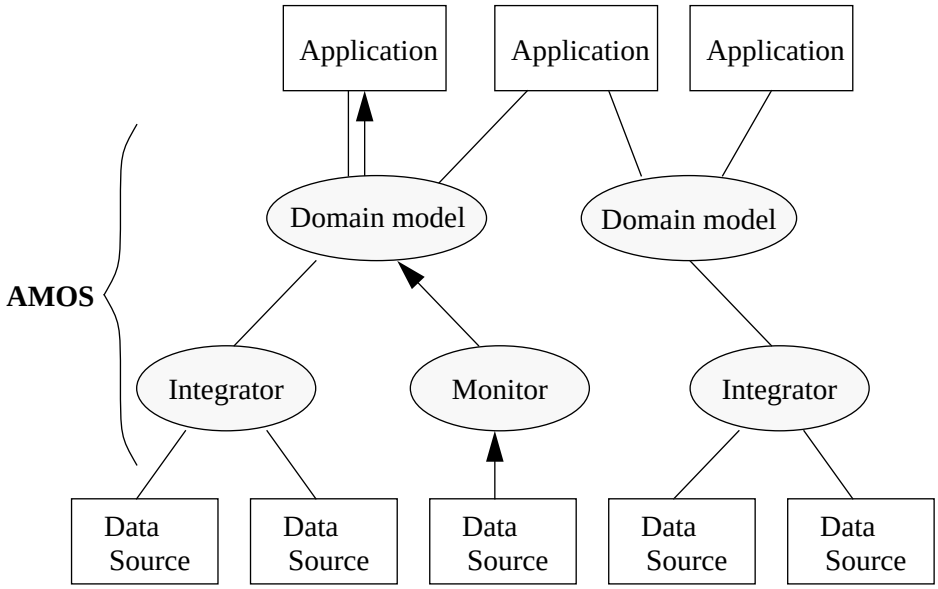

Figure 1 Active mediators of different classes mediating between data sources and users/ applications

where the queried data reside. AMOSQL also supports *active rules* [Ris92] that execute when certain more or less complex conditions change.

To support the initial work on AMOS, a main-memory OO DBMS engine, WS-IRIS [Lit92], is being modified. It provides an extended OSQL version and fast execution. WS-IRIS is open and easy to modify for our research. The system supports extensibility through *foreign functions* written in an external programming language (usually Lisp or C). A query optimizer translates OSQL queries and methods into optimized execution plans in an internal logical language, ObjectLog [Lit92]. The optimizer is extensible so that *cost hints* can be associated with arbitrary OSQL functions to guide the optimizer about alternative execution plans. We are developing new optimization strategies by new kinds of transformations on the ObjectLog query plans.

2 AMOS Components

Figure 1 illustrates how a set of application programs access a set of data sources through active mediators. An overview follows of the work we are doing on each kind of mediator.

2.1 Integrators

Data sources are likely to be heterogeneous. Data could be stored in different DBMSs, using different data models. And even if the same DBMS is used, data could still be semantically heterogeneous [She91].

Integrators are responsible for making this heterogeneity transparent to higher-level mediators and applications. Integrators retrieve and combine data

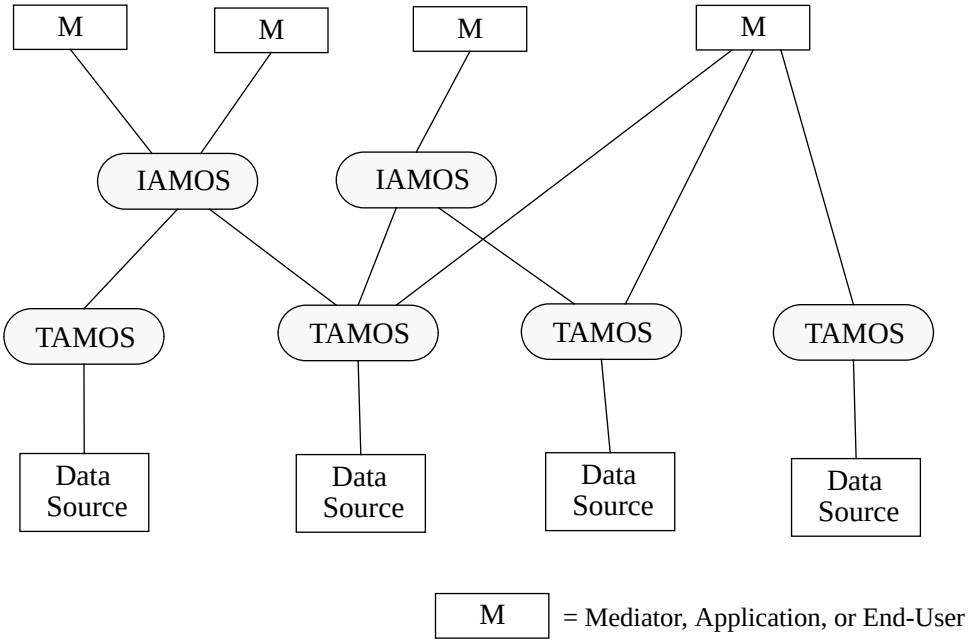

Figure 2 Translation AMOS (TAMOS) and Integration AMOS (IAMOS) - the servers implementing Integrators

from underlying data sources, giving applications and higher-level mediators an integrated view of data and decoupling them from the necessity to understand multiple data models.

Integrators are implemented with two kinds of AMOS servers; *Translation AMOS* (TAMOS) servers and *Integration AMOS* (IAMOS) servers (see figure 2). We will initially concentrate on *access* to heterogeneous data sources, not updates.

There is a lot of current research on heterogeneous database systems [She90]. The usual way to deal with data model heterogeneity is to map the schemas of the data sources to schemas in a common data model (CDM). Most previous research use a relational CDM. This is inadequate if there are data sources with a data model that is semantically richer than the relational model. In these cases, it will not be possible to capture all of the semantics of the data sources in the CDM. Ideally, the expressiveness of the CDM should be greater than, or equal to, the expressiveness of all the data models of the data sources. We use the functional and object-oriented data model from IRIS [Fis89] as our CDM.

Related work of particular interest are the Multibase [Lan82] and Pegasus [Ahm91] projects.

Multibase has a similar architecture and uses a functional data model as their CDM and DAPLEX [Shi81] as the Data Manipulation Language (DML). AMOSQL is a DAPLEX derivative, but an important difference is that AMOSQL is object-oriented. Queries in AMOSQL can return OIDs. Another

difference is the role of the translation component. This will be discussed in section 2.1.1.

The Pegasus project also uses the IRIS data model as their CDM and an extension to OSQL as the DML. The main difference to AMOS is architectural. A Pegasus server performs both translation and integration, whereas in AMOS this is separated in two modules. There is one type of TAMOS server for each type of data source. Each TAMOS server only needs to know the data model of one data source and how to map this to the CDM. IAMOS servers only need to understand the CDM. The Pegasus server must understand all underlying data models and must have language constructs for mapping each of these data sources to the CDM.

2.1.1 TAMOS

Translation AMOS servers map the schemas of the data sources to schemas in the CDM. There is one TAMOS server for each kind of data source. An AMOSQL query sent to a TAMOS server is translated to calls to the underlying data source. The results of these calls are then processed to form answers to the AMOSQL query.

A TAMOS server can be used by one or more IAMOS servers or directly by applications and other mediators. We are initially developing TAMOS servers for a relational database (SYBASE), and for a conventional file data source.

A central problem is how to get OO access to a non-OO data source. In the method chosen, each TAMOS server will contain descriptions of how to map values in the underlying data source to object identifiers (OIDs). OIDs are dynamically generated when necessary and are thereafter maintained by the TAMOS server.

When the data model of the data source provides less semantic modelling constructs than the CDM, mapping a data source schema into a schema in the CDM involves a semantic enrichment process [Cas93]. We want TAMOS to capture as much of the semantics of the data source as possible. This is different from, e.g., Multibase, where the translated schema is the simplest possible and where the semantic enrichment is performed in the integration module. We want to avoid this approach, which leads to increased communication between the translation and integration modules. Our approach makes query optimization in TAMOS more difficult, since query processing involves both calls to the data source and local TAMOS computations. The optimizer must find the most effective combination of these.

TAMOS query plans are represented by an extended version of ObjectLog. Some TAMOS types will have instances corresponding to atomic values in the data source. OIDs must be generated when a query returns objects of such a type. Similarly, OIDs must be converted back to atomic values when they are used in queries to the data source. Thus, TAMOS query plans often contain statements which map between OIDs and atomic values. However, when a query is a used as a subquery of a larger query and thus query plans are combined, the OID mappings on intermediate results are not needed. The optimiz-

er recognizes these cases and removes such unnecessary OID mappings from the execution plan. This makes larger portions of TAMOS execution plans translatable to, e.g., relational queries to the data source, which minimizes communication between TAMOS and its data source.

2.1.2 IAMOS

An Integration AMOS server combines data from other AMOS servers (TAMOS or IAMOS) and presents an integrated view of the data. A query sent to a IAMOS server is transformed into several queries against the underlying AMOS servers. The results of these queries are then processed to form an answer to the initial query. Special optimization techniques are needed compared to conventional distributed DBSs due to the heterogeneity and autonomy of the data sources [Lu93].

To define the mapping between the integrated IAMOS schema and the underlying TAMOS/IAMOS schemas, an OO multi-database query language is needed. This language is used to define object views [Abi91] in terms of combinations of data from other AMOS servers and from local data and views.

To access the data sources it is not necessary to use a IAMOS server. Queries can be put directly against TAMOS servers using the multi-database language. Using the terminology from [She90], our architecture can be seen as a combination of a Loosely Coupled Federated DBS and a Tightly Coupled Federated DBS (with multiple federations).

Thus, the same OO multi-database query language is used for local queries, multi-database queries, and for defining multi-database object views. One proposal being studied for such a language is in [Cho92].

2.2 Monitor Models

Some applications require a mechanism to handle the problem of dynamically changing data. Mediators are provided that continuously monitor these data changes and notify applications when changes of interest for some application occur. These *monitor models* allow application programs to cooperate via AMOS. Of particular interest is to provide means to build monitor models that filter change in data sources, so that irrelevant changes are ignored.

To support monitor models AMOS provides active database capabilities by *active rules* using AMOSQL queries [Ris92]. The active database capabilities of AMOS are used also for other purposes than monitor models, such as for consistency checking. AMOSQL permits functional overloading on types, and types and functions are first-class objects. By implementing rules on top of AMOSQL, overloaded and generic rules are possible, i.e. rules that are parameterized and that can be instantiated for different types. We also utilize the optimizations performed by the AMOSQL compiler.

The HiPac [Day89] project introduced active *ECA rules* (Event-Condition-Action). The event specifies when a rule should be triggered. The condition is a query that is evaluated when the event occurs. The action is executed when the event occurs and the condition is satisfied.

In our active rules the event is made optional by defining each rule as a pair <Condition,Action>, where the condition is a declarative OO query and where the action is an OO database procedure body, i.e. a *CA rule*. An action is executed (i.e. the rule is triggered) when the condition becomes true. We believe that CA rules are more suitable for integration in a query language, since they are more declarative. CA rules make physical events implicit, just as a query language makes database navigation implicit. OPS5 [Bro85] and Ariel [Han92] have similar rule semantics. Unlike those systems, the condition in an AMOS rule can refer to derived AMOSQL functions (which correspond to views). Data can be passed from the condition to the action of each rule by using shared query variables. By quantifying query variables set-oriented action execution is possible [Wid90]. Rules are furthermore parameterized and type overloaded, so that they can be instantiated for objects of different types.

An interface is also developed between active rules and application programs where the programmer can specify *trackers* [Ris89], which are procedures or processes of the application or other AMOS servers that are invoked or called by AMOS when a rule action is triggered. AMOS thus needs a callback mechanism that is invoked from active rules. Such a mechanism will be part of the application programming language interface to AMOS. We have developed such an interface between AMOS and the functional concurrent programming language Erlang [Arm93] for real-time applications. The callback mechanism is also a part of the communication protocol between AMOS servers.

Possible tasks for the trackers include:

- Notifying the end user that data have changed.
- Refreshing data browsers
- Modifying values in mediators.
- Changing processing heuristics in mediators.
- Changing stored abstractions in mediators.
- Informing applications that data views which the application depends on have changed.

By using active rules the monitor model can filter insignificant data source changes before notifying the application. This decreases the frequency of notification for intensively updated data. Notification filtering is required, for example, by real-time monitoring AI systems where the tracker initiates time consuming reasoning activities [Was89].

2.3 Domain Models

Domain knowledge and data now hidden in application programs should be extracted from the applications and stored in mediators with domain specific models and operators, called *domain models*. The benefits of using domain models include easier access through a query language, better data description (as schemas), transaction capabilities, and other benefits currently provided only by advanced DBMSs. Examples of domain models are models for structural analysis of mechanical designs, models to obtain a preferred part

for a product, or models to describe properties of a user interface. The query processing of AMOSQL must be about as efficient as customized main-memory data structure representations. This would encourage the use of local embedded AMOS databases linked into applications. Domain models often need to be able to represent specialized data structures for the intended class of applications.

Important research problems in developing domain models are to investigate:

1. How is the domain modelled using an OO query and modelling language?
2. Which domain oriented data structures are required, and how should they be represented?
3. What domain oriented operators need to be defined?
4. How are queries accessing domain oriented data structures optimized?

2.4 Locators

In large dynamic information bases, it is not trivial to know which data sources contain requested data. For this, a class of mediators is needed, which given descriptions of the data to retrieve, locates the matching data sources. AMOS mediators, called *locators*, will be developed as servers that know properties of other mediators and where they are located. In a simple environment the application will know exactly where the data sources are located, e.g., by knowing the exact locations of database tables. In a broadly distributed environment one may not have such direct 'handles' to the data sources, but rather query the locators given descriptions of what to look for. Locators provide a query language for connecting data to application programs. The effect is to increase flexibility when information sources are changing.

The need for locator facilities has been acknowledged in the research area called *mobile databases* [Imi92], which combine future telecommunication and database capabilities. In a global and very fast network of information servers, databases are accessible via radio links. When people travel long distances the system will eventually move data (or create data) to new locations. In such an environment non-trivial locator facilities become very important. There are some connections between locators and traditional name servers, where IP addresses are looked up via a set of distributed servers. However, since AMOS servers are relatively lightweight, it will be feasible to make each of our locators a complete AMOS server. This will make it possible to provide many new locator services through locator querying. Locators also have connections to traditional DBMS data dictionaries. However, data dictionaries are centralized, i.e. a single data dictionary knows where all data is located, which is what is required to support conventional distributed databases. In contrast, our locators are distributed, autonomous, and loosely coupled.

2.5 Distributed AMOS Systems

The architecture requires facilities to state OO queries and to build OO models that span many AMOS servers. Therefore the system needs to contain means for intercommunication between AMOS servers as well as between AMOS and applications.

We have developed a transactional remote procedure call mechanism that handles low level message interfaces between AMOS servers. A query layer will be built on top of this mechanism. Transactional behaviour ensures that each database can remain consistent after communication or software failures.

We also plan to generalize monitor models so that active rules can be specified that access more than one AMOS server.

3 A Scenario

With AMOS it will be possible to build domain models that combine data from several outside data sources with local data, and which contain rules that assists the user in making decisions.

As an example, consider a computer supported quotation task, where the suggested design and price depends upon prices from subcontractors, e.g. a HV-transformer design depends on copper and oil prices, or a turn-key dairy process equipment depends on stainless steel tubing prices. Integrators allow the information to be stored in and accessible from different vendor databases.

Product data are represented differently by different suppliers. Integrator models allow conversions between semantically different data representations.

Domain models allow customization of parts of the product selection model by local data and rules. For example, the user might specify preferred price ranges, quality requirements, and constraints on the means for transportation from the supplier. Different domain models will be used by different users and have different customizations.

All data used in the product selection may not be directly available for each considered product and locators must then be used to find the appropriate database. For example, access to each supplier's database is needed in order to estimate the cost of obtaining a product.

Assuming that the main contractor is not in direct hurry to buy the product, s/he may postpone the purchase until the right market conditions occur which can be provided by monitor models. For example, if supplier A does not have the desired product in stock, the main contractor may want a signal if and when it can be delivered also from supplier A. Similarly, s/he may want a signal if the price for the product drops below some threshold in case a sale is expected. These kind of monitoring conditions are expressible in the rule language. We also plan to add timeliness specifications in monitoring models,

e.g. to specify a deadline after which it is absolutely necessary to have an order put on the needed product, even if the choices are not the best.

4 Summary

The AMOS architecture was described where AMOS information bases mediate between application programs and data sources. AMOS provides facilities to extract data, and to manipulate and model the extracted data using a powerful query and modelling language. The system provides integrator servers that combine data from many different data sources, and provide OO views for all types of data sources. The modelling language has active rules facilities that detect when the state of a data source gets updated in some 'critical' way. Critical data source updates can then initiate reasoning in application programs or just notify the user. The modelling language is based on extensions to OSQL [Fis89].

An example scenario was given of the use of AMOS to help main contractors get timely and needed information for the quotation task. One may construct similar scenarios for other domains, e.g., computer network service planning systems, and project planning and tracking systems.

References

[Abi91] Abiteboul S., Bonner A., 'Objects and Views', *Proc. ACM SIGMOD*, 1991.

[Ahm91] Ahmed R., DeSmedt P., Du W., Kent W., Ketabchi M.A., Litwin W., Rafii A., Shan M-C. 'The Pegasus Heterogeneous Multidatabase System', *IEEE Computer*, Vol. 24, No. 12, Dec. 1991.

[Arm93] Armstrong J., Williams M., Virding R., *Concurrent Programming in Erlang*, Prentice-Hall, 1993. ISBN 13-285792-8.

[Bro85] Brownston L., Farell R., Kant E., Martin A., *Programming Expert Systems in OPS5*, Addison-Wesley, Reading Mass., 1986.

[Cas93] Castellanos M., 'Semantic Enrichment of Interoperable Databases', *Proc. RIDE-IMS (Interoperability in Multidatabase Systems) Workshop*, Vienna 1993.

[Cho92] Chomicki J., Litwin W., 'Declarative Definition of Object-Oriented Multidatabase Mappings', in Özsu M.T., Dayal U., Vadduriez P. (eds.): *Distributed Object Management*, Morgan Kaufmann Publishers, 1993 (to appear).

[Day89] Dayal U., McCarthy D., 'The architecture of an Active Database Management System', *Proc. ACM SIGMOD*, 1989, pp. 215-224.

[DE92] *IEEE Data Engineering* bulletin, Vol. 15, No. 1-4, Dec. 1992.

[Fis89] Fishman D. et al., 'Overview of the Iris DBMS', *Object-Oriented Con-*

cepts, Databases, and Applications, ACM press, Addison-Wesley Publ. Comp., 1989.

[Han92] Hanson E. N., 'Rule Condition Testing and Action Execution in Ariel', *Proc. ACM SIGMOD*, 1992, pp. 49-58.

[Imi92] Imielinski T., Badrinath B.R., 'Querying in highly mobile distributed environments', *Proc. VLDB '92*, pp. 41-52.

[Lan82] Landers T., Rosenberg R., 'An overview of Multibase', in Schneider H-J. (ed.): *Distributed Databases*, North-Holland, 1982, pp. 153-184.

[Lit92] Litwin W., Risch T., 'Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates', *IEEE Transactions on Knowledge and Data Engineering,* Vol. 4, No. 6, December 1992.

[Lob93] Loborg P., Risch T., Sköld M., Törne A., 'Active Object-Oriented Databases in Control Applications', *19th Euromicro Conference*, Barcelona 1993 (to appear).

[Lu93] Lu H., Ooi B-C., Goh C-H., 'Multidatabase Query Optimization: Issues and Solutions', *Proc. RIDE-IMS (Interoperability in Multidatabase Systems) Workshop*, Vienna 1993.

[Ris89] Risch T., 'Monitoring Database Objects',*Proc. VLDB '89*, Amsterdam 1989.

[Ris92] Risch T., Sköld M., 'Active Rules based on Object-Oriented Queries', *IEEE Data Engineering* bulletin, Vol. 15, No. 1-4, Dec. 1992, pp. 27-30.

[She90] Sheth, A., Larson, J., 'Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases', *ACM Computing Surveys*, Vol. 22, No. 3, September 1990.

[She91] Sheth, A., 'Semantic Issues in Multidatabase Systems', Preface by the special issue editor, *SIGMOD RECORD*, Vol. 20, No. 4, December 1991.

[Shi81] Shipman, D.W., 'The Functional Data Model and the Data Language DAPLEX', *ACM Transactions on Database Systems*, Vol. 6, No. 1, March 1981.

[Was89] Washington R., Hayes-Roth B., 'Input Data Management in Real-Time AI Systems', *11th Intl. Joint Conf. on Artificial Intelligence*, 1989, pp. 250-255.

[Wid90] Widom J., Finkelstein S.J., 'Set-oriented production rules in relational database system', *Proc. ACM SIGMOD,* Atlantic City, New Jersey, 1990, pp. 259-270.

[Wie92] Wiederhold G., 'Mediators in the Architecture of Future Information Systems', *IEEE Computer*, March 1992.