

DBAA/ACL - A DATABASE AGENT ARCHITECTURE AND COMMUNICATION LANGUAGE

Markus Kirchberg

*Information Science Research Centre and Department of Information Systems
Massey University, Private Bag 11-222, Palmerston North 5301, New Zealand
M.Kirchberg@massey.ac.nz*

ABSTRACT

Reliable support for distributed data management is one of the key features of present and future database systems. In this paper, we propose an agent-based communication mechanism for distributed database systems. First, an agent architecture (DBAA) is introduced. Agents can function on their own or work cooperatively with a (virtual) collection, block or federation of agents. While working towards a certain goal, agents may exploit concurrency, parallelism and distribution. To do so, communication capabilities are required. Thus, a corresponding agent communication language, DBACL, is proposed. DBACL supports a variety of communication types (i.e. one-to-one, one-to-many and broadcast), message transfer modes (e.g. streams), and message types (e.g. signals, notifications, requests, and negotiations). Besides efficiency and interoperability, simplicity has been another main objective underlying our research.

KEYWORDS

Distributed Database System, Agent Communication.

1. INTRODUCTION

The distributed nature of nowadays businesses requires reliable and efficient support for their data management needs. On one hand, data should be kept close to the location where it is required most. On the other hand, data should also be available to everyone else within the scope of a particular business division, the entire business or even the whole world. This leads to distributed data management systems (DDBMSs) and their adequate support for the storage and retrieval of complex, real-world objects and relationships among them, and the realization of behavior associated with them. We consider such a support to be only adequate, if it is acceptable both on the application-level and the level of technical realization. Consequently, one of the most challenging problems in a DDBMS is its underlying data communication mechanism.

Object-oriented data communication approaches have come a long way. Microsoft DCOM and .Net, OMG CORBA, Sun Java/RMI and J2EE are used most commonly these days. While they are adequate for current needs of most application developers, more and more attention is drawn to agent communication languages (ACLs). An agent (or database agent or software agent) can be regarded as a piece of software (most commonly realized as a thread) that performs one or more relatively simple tasks to fulfill one or more given requests. Agents may work independently or cooperatively. According to (Labrou, Finin et al. 1999), ACLs stand a level above CORBA (and similar approaches). Among others, ACLs handle propositions, rules, and actions instead of simple objects with no semantics associated with them. (Singh 1998) also underlines this point. In addition, it is added that agents are able to perceive their environment and may reason and act both alone (i.e. autonomy) and with other agents (i.e. interoperability). ACLs define the type of messages that agents exchange. However, agents do not just exchange messages; they have (task-oriented) 'conversations'. For instance, in a database system (DBS) they could first negotiate how to execute a given request most efficiently (i.e. query cost estimation) and then cooperatively evaluate the chosen execution plan that implements the request most efficiently. ACLs are meant to equip agents with the ability to exchange more complex objects, such as shared plans and goals. Agents still (at the technical level) transport messages over the network using a lower-level protocol (e.g. TCP/IP, IIOP or even HTTP). However, such implementation-specific considerations are beyond the scope of this paper.

1.1 Objectives

In this paper, we propose an agent-based communication mechanism. We focus on both, the architecture of agents (DBAA) and an agent communication language (DBACL). The latter will enable agents and, thus, different components of a (distributed) system to interact in order to execute user requests more efficiently.

DBSs are our main target environment. In short, we aim at building a distributed object-oriented DBS as outlined in (Kirchberg, Schewe et al. 2006). Within this line of research an integrated distributed database programming and querying language, called DBPQL, is proposed. As with conventional DDBSs user requests (i.e. DBPQL programs) are compiled, fragmented, allocated, optimized, etc. Finally, a user request emerges in the form of an execution plan, which is then executed by a DBS component commonly known as the evaluation engine (EE). Each node that forms a part of the DDBS has one instance of such an EE. The collection of EEs can be regarded as a network of agents that cooperatively evaluate any given execution plan. However, such an evaluation also involves agents that belong to the transaction management system, agents that maintain metadata repositories, etc. In fact, a large number of agents of different DBS components exist that require communication in one or the other way. Originally, we have intended to rely on a mixture of inter-process and thread communication mechanisms and an extended remote object call mechanism as introduced in (Wang, Kirchberg et al. 2003). However, our experiences have shown that this level of communication support is not sufficient. For instance, during the optimization process an execution plan has to be selected. To do so, a number of potential plans are evaluated. Having negotiation and voting capabilities built-in the basic communication mechanism (as possible with ACLs), such a process can be realized much more efficiently in contrast to more conventional communication approaches.

While designing DBAA and DBACL, main objectives included simplicity, interoperability and cooperation with particular focus on:

- Flexibility in assembling a number of autonomous agents to fulfill one or more tasks.
- Flexibility in ways agents can communicate. This requires that the assembly of agents imposes no restrictions on possible communication patterns.
- Separation of Agent Communication and Agent Execution Languages. Thus, the communication architecture is not affected by the way agents execute tasks internally.
- Collaboration support between agents of different DBS components.
- Minimization of efforts required to create new agents, establish connections, exchange data, etc.

1.2 Outline

The remainder of this paper is organized as follows: Section 2 provides a brief overview of existing work. Section 3 addresses our target environment. Subsequently a suitable database agent architecture (Section 4) and database agent communication language (Section 5) are proposed. Finally, Section 6 concludes our work.

2. RELATED WORK

Significant research has been conducted in the area of software agents. Many communication mechanisms that apply to software agents have been proposed. However, not all of them are specifically designed for software agents. We are only interested in the ones designed for agent-based systems. (Genesereth and Ketchpel 1994; Nwana 1995; Singh 1998; Labrou, Finin et al. 1999; Chaib-draa and Dignum 2002) discuss the evolution of software agents, original objectives, trends that have emerged, and particular solutions that have gained a lot of attention.

(Finin, Fritzson et al. 1994) discusses KQML (Knowledge Query Manipulation Language) a language for knowledge exchange. KQML follows a similar approach as ours w.r.t. separating communication and content languages. During the past 25 years, a large number of KQML dialects have been proposed. However, only a few of them are still in use nowadays (Singh 1998). More recent approaches include the Open Agent Architecture OAA (Martin, Cheyer et al. 1999) a framework for building distributed software systems and standards proposed by the Foundation for Intelligent Physical Agents FIPA (<http://www.fipa.org/>).

Most of these systems are designed for general agent-based system development treating database access only as another feature. They mainly aim to support human beings to develop applications (e.g. on top of existing DBSs) more easily. We follow a slightly different path. We intend to build an agent-based communication mechanism that can support the efficient implementation of the core functionality of a distributed DBS. While related tasks correspond to a lower level of abstraction than those systems mentioned above, we can still benefit from an agent-based solution (e.g. in terms of task-oriented conversations, adapting a certain role based on the environment in which the agent is employed, etc.) that is richer than current non-agent-oriented solutions as mentioned in Section 1.

3. A DISTRIBUTED DATABASE ENVIRONMENT

In this section, we discuss the environment we had in mind when designing the communication mechanism.

The agent-based communication mechanism has been designed for a truly distributed DBS. In such a system, a high-level user is not aware of the distributed nature of the DBS. In fact, data independence, network transparency, replication transparency, and fragmentation transparency are key properties. Thus, a high-level user request does not contain any communication- or location-specific information. Only during the compilation, fragmentation, allocation, and optimization processes such information is added. This allows for an easy separation between the ACL and the Agents' Execution Language (AEL).

The choice of data model on the conceptual level is not of major importance. The communication mechanism is designed to be employed on a lower DBS level where requests are evaluated. In relational DBSs such a component is usually referred to as the Query Evaluation Engine (e.g. in (Ramakrishnan and Gehrke 2003)). However, more sophisticated DBSs (e.g. object-relational, object-oriented and XML DBSs) do not just evaluate queries. Thus, we will refer to such a DBS component as Evaluation Engine (EE). Details about types of objects, queries, methods, execution plans, etc. are not relevant to the proposed communication mechanism in general. Only the AEL needs to know about this. However, when data is transferred between two DBS nodes, data has to be transformed into a format suitable for transmission. Thus, there has to be a means of flattening (and unflattening on the receiving end) an object, a connected set of objects, a query, a method, or even an entire execution plan. This is a difficult issue for most systems since explicit flattening (and unflattening) functions have to be provided. However, emerging DBSs (such as the ones this communication mechanism is designed for) are desired to support (refer (Atkinson, Bancilhon et al. 1989; Atkinson and Morrison 1995; Leontiev, Özsu et al. 2002))

- Persistence independence;
- Orthogonality of type and persistence, which implies 1) a uniform treatment of volatile and persistent data; and 2) that data of all types (e.g. objects and behavior) can be persistent as well as transient.

Thus, the DBS already contains mechanisms for (un-)flattening all types of data since this functionality is required to ensure that any data item the DBS is dealing with can be made persistent. Whether this functionality is part of an agent's AEL or a Persistent Object Store component depends on the particular DBS. Alternatively, agents have to provide the message content (in AEL) in a serial form -- for instance, KQML (Finin, Fritzson et al. 1994) makes such an assumption. Note: Neither of the two options affects the DBS performance significantly more than the other. The main drawback of the latter solution would be that agents have to be aware of whether or not a request is directed to a local or remote agent. The former approach allows agents not to worry about this and the ACL deals with this issue.

Each system that forms a part of the DDBS employs a database management system (DBMS), which consists of a number of DBS components. Common components are: an EE, a Transaction Manager, a Recovery Manager, a Fragmentation and Allocation Facility, Metadata Repositories, etc. Refer to your favorite textbook (e.g. (Ramakrishnan and Gehrke 2003)) for a more details.

Implementing the functionality of such DBS components can be done in an agent-oriented way. (Kirchberg, Schewe et al. 2006) can be considered as sample architecture of a modern DDBS architecture that follows this path. For instance, the evaluation engine is regarded as a network of agents that cooperatively evaluate database requests.

4. THE DATABASE AGENT ARCHITECTURE

In this section, we present an overview of the database agent architecture. While the exact implementation of a particular agent may differ from DBS component to DBS component, the agent architecture and the agent communication language (described in Section 5) are common properties of all DBS agents.

Basically, there are two types of agents. These are:

1. **Master Agents (MAs).** There is at least one MA per DBS component. It provides a means of first point of contact to other DBS components or external 'users'. Each DBS component that intends to provide services to others will have to register at least one MA before its services can be accessed. MAs maintain a pool of idle execution agents that can be assigned dynamically to answer a particular request that has been forwarded. Thus, MAs delegate incoming requests. Once a request has been delegated, the MA will not be involved in this or any subsequent requests originating from the same requester. The assigned execution agent will take over all communication. A collection of MAs can be used when the number of incoming requests becomes large enough so that an individual MA would be a bottleneck. To do so, each MA has the capability to replicate itself (refer to the `cloneMA` command of the DBACL introduced in Section 5).
2. **Execution Agents (EAs).** Execution agents are basic building blocks and have an associated (internal) AEL. They perform a number of relatively simple tasks to fulfill one or more requests.

An execution agent can be part of one or more of the four common building blocks, which are as follows:

- **Agent Collection.** A collection of agents is formed as soon as an EA delegates a number of sub-tasks to another EA on the local DBS node. The new EA is obtained from the pool of idle agents. These agents of the same collection work cooperatively towards a common goal. Collections may contain sub-collections. The collection concept can be used to model (local) transactions. This is done by designating a particular sub-collection as sub-transaction. The outermost collection always corresponds to the top-level transaction (e.g. in case a multi-level transaction model is used as discussed in (Kirchberg and Schewe 2001)).
- **Agent Block.** A block of agents can be understood as a special type of a collection that becomes important in the DBS environment. For instance, during the evaluation of DBS requests intermediate results are created. Whenever possible results are pipelined between agents implementing subsequent tasks as specified in an execution plan. However, at times, it is necessary to materialize some of these intermediate results. Blocks can be seen as 'materialization' boundaries. Pipelining is used for agent communication inside a block. Whenever the block of agents has fulfilled one or more tasks, results are materialized before being returned to the caller. Thus, blocks are used to make local pipelining more efficient. Agents that belong to the same block can share memory and use special synchronization commands (implemented as signals, which are part of DBACL) to govern access to these shared memory areas. Note: concurrent and parallel processing also benefit from using blocks. For instance, accessing data and merging results can be implemented more efficiently.
- **Agent Federation.** While agents in collections and blocks add new agents to help with the completion of tasks, agents that join a federation only share their already obtained results. Agents in a federation might have different goals but share some common interest, sub-tasks, sub-goals, etc.
- **Virtual Agent Collection.** Virtual collections can be used for different purposes. They can be used in the same way as collections but may span multiple DBS nodes; or they can be maintained explicitly to group a number of agents that work towards the same goal. For instance, distributed transactions and replica management can be governed using the notion of virtual collections.

Note: Whereas collections and blocks are maintained by the system automatically (through the creation of new agents, a corresponding hierarchy is formed), federations and virtual collections have to be maintained explicitly (i.e. using corresponding ACL calls such as `addToFederation`, `newVirtualColl`, etc.).

Based on these types of agents and building blocks, we require the following three DDBS components: A system-wide registry, one or more Master Agents per DBS component and an agent pool per DBS component. All communication needs are provided by the DBACL discussed in Section 5.

5. THE DATABASE AGENT COMMUNICATION LANGUAGE

DBACL is similar to the KQML language (Finin, Fritzon et al. 1994) in terms of separating communication aspects from, what we call, the Agent Execution Language. Thus, agents exchange messages (in an AEL of their choice) wrapped in a DBACL message. However, KQML and DBACL differ in the way these wrapped AEL messages are transformed in a serial form. Recall the corresponding comments from Section 3.

DBACL mainly aims at creating or locating agents suitable to achieve a certain goal; packaging messages in a way that general intentions (e.g. communication types) are easy to identify; and then engaging these agents in conversations to cooperatively meet that goal. DBACL supports communication types as follows:

- **Agent-to-Agent (AtA)**. One agent communicates with another agent.
- **Agent-to-AgentList (AtAL)**. One agent communicates with a list, (virtual) collection, (virtual) sub-collection, block, or federation of agents.
- **AgentBroadcast (AB)**. A broadcast message is sent to a list of interested agents, to a (virtual) collection, (virtual) sub-collection, block, or federation of agents.

Once the required communication type is known, the type of message transmission is to be decided. DBACL supports the following types of data transmission:

- **All-At-Once (AAO)**. Messages are only transmitted when assembled completely.
- **Stream**. Messages are transmitted without knowing all arguments and / or results. Thus, multiple messages are exchanged. We further distinguish between UpStream and DownStream as follows:
 - **UpStream (UpS)**. The task specification is provided in multiple messages. For instance, some arguments will be forwarded as they become available. The first message contains a description of how the remaining data is provided. Triggers are used here. The agent is then given a reference to a message queue that will be used to provide additional messages.
 - **DownStream (DwnS)**. Results are provided piece-by-piece or can be requested on demand. Again, the agent is given a reference to a message queue that will be used to provide results. When using stream transmission modes, triggers have to be specified. A trigger tells an agent when to push or pull the next message containing still-to-be-delivered data. We distinguish between push and pull triggers. Push triggers push data from the source to the destination. Push-constraints include: *AsAvailable*, *SizeOf(bytes)*, and *TimeElapsed(ms)*. Pull triggers result in a signal (from the destination to the source) requesting the next message. Pull-constraints include: *TimeElapsed(ms)*. Trigger conditions can be overwritten by using the *pushASAP* or the *pullASAP* signal. (Software) pipelining is one of the main applications using Stream-based communication.

Transmission types can be specified both ways, upstream and downstream. For instance, an agent can forward a task without providing any of the operational data. Operational data may then be pulled from the target-agent using triggers. This type of request is very useful for DBSs, e.g. for join operations. An agent can be told to prepare to execute a join operation while two other agents retrieve the collections to be joined. The join-executing agent then pulls data from the two collection-retrieving agents as required.

Table 1: Overview of Common Message Formats.

Message Type	Communication Type	Transmission Type			Triggers	Pre	AEL	Post	Cmpl
		AAO	UpS	DwnS					
Signal	AtA, AtAL	Yes	No	No	No	No	Yes	No	No
Notification	AtA, AtAL, AB	Yes	No	No	No	No	Yes	No	No
Broadcast	AB	Yes	No	No	No	No	Yes	No	No
Request	AtA, AtAL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Vote	AtA, AtAL	Yes	No	No	No	Yes	Yes	Yes	Yes
Negotiation	AtA, AtAL	Yes	No	No	No	Yes	Yes	Yes	Yes

Pre (pre-condition), AEL (message content), Post (post-condition), and Cmpl (completion-condition) fields are specified in an AEL of the agents' choice.

There are a number of pre-defined message types. An overview is given below. In addition, Table 1 shows which message fields are associated with each message type. For instance, *Request*, *Vote* and

Negotiation types allow the specification of pre-condition, post-condition, and completion-condition in addition to the message content. While triggers result in agents taking actions when some set of communication-related conditions are met, pre-conditions, post-conditions, and completion-conditions (formulated in AEL) correspond to conditions that are to be forwarded to the inner-agent logics. Thus, two levels of condition-based actioning are supported. Types of messages include:

- **Signals; Notifications; Broadcasts; Votes; Negotiations;** etc.
- **Requests**, which include the following sub-types:
 - Registration Requests (`register`, `unregister`, `lookup` etc.);
 - Agent Administration Requests (`cloneMA`, `delegateToAgent`, `obtainAgent`, `releaseAgent`, `obtainAgentBlock`, `releaseAgentBlock`, `newFederation`, `addToFederation`, `joinFederation`, `removeFromFederation`, etc.);
 - EvaluationRequest (formulated in some AEL).

Unfortunately, a complete syntax, detailed semantics and implementation aspects of DBACL are beyond the scope (and page limitations) of this paper. Our prototype implementation operates on top of TCP/IP connection pools that exploit physical and virtual connections to cut down on connection set-up times.

6. CONCLUSIONS

In this paper, we have proposed a database agent architecture and a corresponding database agent communication language. Our motivation is to implement the functionality of modern distributed DBSs using software agents. The proposed agent-based communication mechanism allows for an easy and straightforward interoperability of DBS components. A common communication language is used for both, local and remote communication demands. In addition, non-DBS components (e.g. low-level access for Database Administrators) can be added easily.

REFERENCES

- Atkinson, M., F. Bancillon, et al. (1989). The Object-Oriented Database System Manifesto. First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan.
- Atkinson, M. and R. Morrison (1995). "Orthogonally Persistent Object Systems." The VLDB Journal **4**(3): 319-402.
- Chaib-draa, B. and F. Dignum (2002). "Trends in Agent Communication Language." Computational Intelligence **18**(2): 89-101.
- Finin, T., R. Fritzson, et al. (1994). KQML as an Agent Communication Language. Third International Conference on Information and Knowledge Management (CIKM), Gaithersburg, Maryland, United States, ACM Press.
- Genesereth, M. R. and S. P. Ketchpel (1994). "Software Agents." Communications of the ACM **37**(7): 48-??
- Kirchberg, M. and K.-D. Schewe (2001). A Comparison of Multi-Level Concurrency Control Protocols. Twelfth Australasian Database Conference (ADC), Gold Coast, Australia, IEEE Computer Society Press.
- Kirchberg, M., K.-D. Schewe, et al. (2006). "A Multi-Level Architecture for Distributed Object Bases." Data & Knowledge Engineering.
- Labrou, Y., T. Finin, et al. (1999). "Agent Communication Languages: The Current Landscape." IEEE Intelligent Systems **14**(2): 45-52.
- Leontiev, Y., M. T. Özsu, et al. (2002). "On Type Systems for Object-Oriented Database Programming Languages." ACM Computing Surveys (CSUR) **34**(4): 409-449.
- Martin, D., A. Cheyer, et al. (1999). "The Open Agent Architecture: A Framework for Building Distributed Software Systems." Applied Artificial Intelligence **13**(1/2): 91-128.
- Nwana, H. S. (1995). "Software Agents: An Overview." Knowledge Engineering Review **11**(2): 205-244.
- Ramakrishnan, R. and J. Gehrke (2003). Database Management Systems, McGraw-Hill Higher Education.
- Singh, M. P. (1998). "Agent Communication Languages: Rethinking the Principles." Computer **31**(12): 40-47.
- Wang, R. B., M. Kirchberg, et al. (2003). OORPC: A Communication Mechanism for Distributed Object Bases. Third International Conference on Electronic Commerce Engineering (ICeCE), Hangzhou, China, International Academic Publisher/World Publishing Corporation.