Requirements for Agent-based Modeling of Federated Database Systems (Extended Abstract)*

Can Türker Gunter Saake Stefan Conrad

Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
E-mail: {tuerker|saake|conrad}@iti.cs.uni-magdeburg.de
Phone: ++49-391-67-{12994|18800|18066} Fax: ++49-391-67-12020

1 Motivation

Considering the needs of present and future information systems, interoperability of independent databases systems becomes more and more important for an enterprise in order to be able to act competitively in an increasing world-wide market [Bro92]. Today, nearly every enterprise has to face the situation that different database systems are used in its departments. Problems caused by this situation are redundant data, inter-database dependencies and data-exchange between different databases [HM93]. For each database system there often exists a large number of applications which have to be preserved. On the other hand, new multidatabase applications must be supported, for instance, for realizing workflow systems [BJ94]. Another very important aspect concerns the adaptability of information systems to changing requirements und highly dynamic environments (e.g. rapidly changing markets, increasing (world-wide) competition, and new trade agreements as well as (inter)national laws).

In the last decade, the term *federated database system* has evolved [HM85, SL90, LMR90]. A *federated database system* (FDBS) is a system which consists of a collection of cooperating *component database systems* (CDBSs) which are to a certain degree *autonomous* and possibly *heterogeneous*. Such kinds of systems aim at a virtual integration of preexisting, originally autonomous databases and database systems, respectively, and thus to provide a uniform access to data from different databases (see Figure 1). A central issue of FDBSs is that the independent CDBSs can continue performing their local operations while they are participating in a federation. In other words, CDBSs retain their interfaces so that existing applications can continue without any modifications.

*This research was partially supported by the CEC ESPRIT Basic Research Working Group No. 8319 ModelAge (A Common Formal **Model** of Cooperating Intelligent **Age**nts) and by the German Country Sachsen-Anhalt under FKZ: 1987A/0025 (Federating Heterogeneous Database Systems and Local Data Management Components for Global Integrity Maintenance).

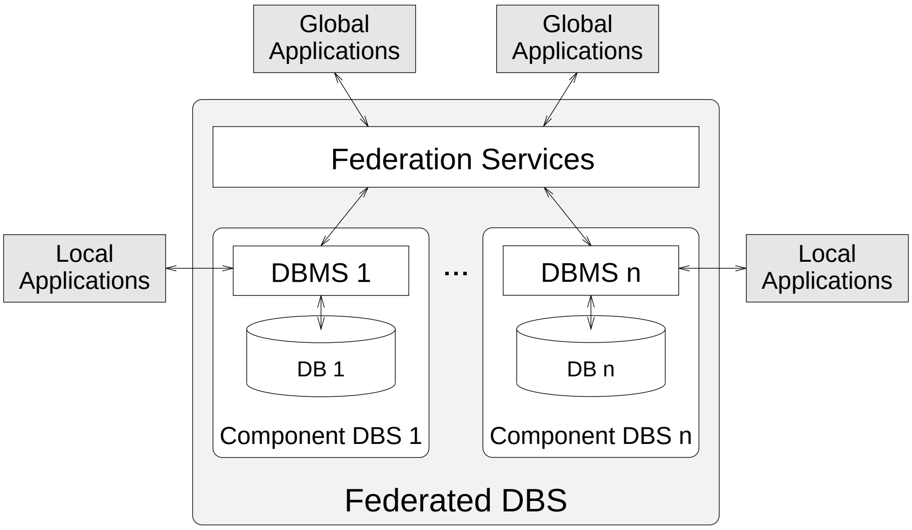

Figure 1: A simplified architecture for federated database systems

Federated database systems are characterized by three orthogonal dimensions: autonomy, heterogeneity, and distribution.
Autonomy: The component database systems are autonomous with respect to the

- execution of the local database operations,
- communication/cooperation with other component database systems
- design of the component database systems (free choice of the data model, transaction model, etc.)

Heterogeneity: The component database systems may be heterogeneous with respect to the

- data models (e.g. hierarchical, network, relational, object-oriented)
- query languages (SQL, QBE, ObjectSQL, etc.)
- transaction models (flat versus nested transactions, open versus closed nested transactions, and so on)
- concurrency control strategies (conservative versus optimistic scheduling, locking versus timestamp-based protocols, etc.)
- recovery strategies (different commitment protocols, e.g. two or three phase commit)
- semantic heterogeneity (different meaning or interpretation of shared data, synonyms and homonyms, meta conflicts, etc.)

Distribution: This aspect may concern to the distribution of

- data

- applications
- control
- knowledge

In the following section, we sketch the idea of providing an intelligent and flexible integration by using evolutionary agents which are able to act goal-driven without external stimuli and which are able to change their behavior dynamically during the system run-time [TCS96]. Here we use the term *agent* instead of (active) object because agents have additional properties which are useful when modeling federated systems. Basically, agents are interpreted as *dynamic* and *intelligent* objects (cf. [WJ95, BGKK95]), which are assumed to be

- *autonomous*: agents act without direct (user) intervention;
- *cooperative*: agents cooperate, coordinate and negotiate with other agents;
- *reactive*: agents respond flexible to dynamic and unpredictable situations that occur in their environments;
- *pro-active*: agents may also exhibit goal-driven behavior by taking the initiative.

2 An Agent-based Architecture for Database Federations

In this section, we discuss an agent-based architecture for the federation of autonomous heterogeneous database systems. A database federation consists of a federation agent and several mediator agents. For each component database system there is a mediator agent which is responsible for the access to this component system. Analogous, for each database federation there is a federation agent which is responsible for building and maintaining the federation (see Figure 2). Both, mediator agent as well as federation agent have to cooperate in order to fulfill the requirements of federated systems, e.g. to provide a uniform interface to distributed heterogeneous data sources.

The following subsections give an overview of the special tasks of the agents mentioned above.

2.1 Mediator Agents

The main task of a mediator agent is to build up the communication between the local applications and a component database system and between the federation agent and a component database system. Because the interface of a mediator corresponds to the original one of the corresponding component database system, the existing local applications may continue without any modifications. All local database calls are passed to the mediator which is then able to analyze the requests.

Special tasks of a mediator agent are:

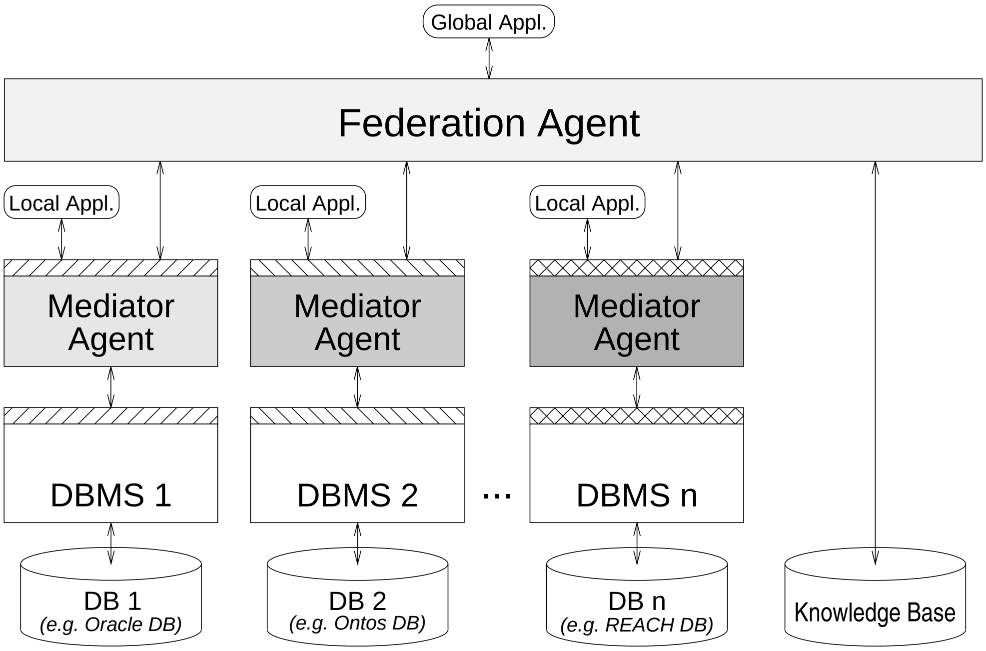

Figure 2: Agent-based federation of database systems

- *data model mapping*: a mediator has to transform data between different representations (data models).

- *query transformation*: queries has to be transformed between different query languages for different data models.

- *concurrency and recovery control*: in order to provide global transaction management which supports global serializability and global atomic commitment, a mediator agent has to cooperate with the corresponding federation agent (e.g. exchanging lock informations).

- *semantic integrity control*: local updates may violate the global integrity of a database federation. Therefore, a mediator has to control the local operations and to propagate the relevant updates to the corresponding federation agent.

- *access control*: a mediator agent must distinguish between local and global users as well as between the local database administrators and the global database administrators.

- *version control*: a mediator has also to deal with several upgrade version of component database systems.

- *system evolution control*: a main requirement for mediator agents is to control the evolution of the system behavior. A mediator must be adaptable

  – to changes in the local system environment e.g. new versions of component database systems,

– to changes of integrity constraints, and

– to changes of data semantics.

2.2 Federation Agents

The main task of a federation agent is to mediate between mediator agents. Thus a federation agent can be seen as a global mediator with similar tasks as mediator agents. Besides, a federation agent is responsible for the configuration management of the federated system. It must be able to integrate new component systems (database systems plus mediator agents) as well as to activate and deactivate components. Consequently, a federation agent has to manage changes of the global schema.

Furthermore, a federation agent must deal with vertical and horizontal schema evolution. It has to adapt the global schema to the set of currently available component schemas. Moreover, it must reflect changes in the component schemas as well as changes of global integrity constraints. In the letter case, it must be able to react on changes of integrity constraints of local schemas. In conclusion, the agent's behavior must be adaptable to evolving schemas.

Another very important aspect concerns the global query processing. A federation agent has to decompose a global query by using the metadata (about the database federation) into local queries which can be executed in the component systems. The results of the local queries must then be merged and returned to the user.

A federated database system also requires global transaction management facilities. In order to realize a certain kind of correctness criterion for the execution of global transactions, coordination of local transactions is necessary. Please notice that the ACID (atomicity, consistency, isolation, durability) requirements for global transactions are very difficult to maintain — sometimes it is impossible. Therefore, advanced transaction models with compensating and alternative (contingency) transactions must be supported at the federation level.

3 Towards Agent Specification

In this section, we sketch the basic frame of an agent-oriented specification und show how federation and mediator agent may be specified. We use a two-level specification framework which is based on the idea "states as theories". By this approach we are able to express the agent's state (knowledge) as logical rules. Thus, state changes can be seen as theory revisions. Consequently, the behavior of an agent can be changed by adding and removing axioms from the current state of the agent.

In our specification framework, the fixed behavior of an agent, which must not be changed during the lifetime of this agent, is represented by usual attributes and events as in traditional object specification languages like TROLL [JSHS96]. The part of the behavior specification which must not be changed is specified in the **rigid axioms** section. For the modeling of dynamically changeable behavior we have introduced additional non-first-order attributes (**axiom attributes**), which contain axioms as values, and special events (**mutators**) for manipulating these attributes (see [SCT95, CS96, TCS96]).

An Agent Class `Mediator`. The following example sketches a part of a possible specification of a mediator agent. Mediator agents are identified by their names and locations. Furthermore, they are participating in several federations. The information about the corresponding federation agents is stored in the attribute `Federator`. A mediator agent may have some fixed events, e.g. an event `Update` which always initiates an `Update` on the corresponding `Federator`.

```
agent class Mediator
  identification   ID: (Name, Location);
  attributes
    Name:          string constant;
    Location:      string;
    Federator:     |Federator|;
    ...
  events
    Update(s:string);
    Modify(s:string);
    ...
  rigid axioms
    Update(s)        calling   Federator.Update(s);
    Modify(s)        changing ...;
    ...
  axiom attributes
    Axioms initialized {};
    ...
  mutators
    ResetAxioms;
    AddAxioms(f:Formula);
    RemoveAxioms(f:Formula);
    ...
  dynamic specification
    ResetAxioms       changing Axioms := {};
    AddAxioms(f)      changing Axioms := Axioms ∪ f;
    RemoveAxioms(f)   changing Axioms := Axioms - f;
    ...
end agent class Mediator;
```

An Agent Class `Federator`. The following lines show a part of a possible specification of a federation agent. A federation agent may be identified by a name. Furthermore, it has an attribute which contains the information about the participating mediator agents. By using the event `Add` a new mediator agent is integrated into the federation. The event `Remove` removes a mediator agent from the federation. Moreover, there may be a lot of other events, for instance, for the global query processing or transaction management.

In this example we specified an axiom attribute `ICs` which contains a set of dynamically changeable integrity constraints. These attribute may be modified by using the corresponding events `ResetICs`, `AddICs`, and `RemoveICs`.

```
agent class Federator
  identification   ID: (Name);
  attributes
    Name:          string constant;
```

```
    Mediators:   set(|Mediator|);
    ...
  events
    Add(m:|Mediator|);
    Remove(m:|Mediator|);
    GetQuery(s:string);
    SendQuery(s:string);
    Update(s:string);
    ...
  rigid axioms
    Add(m)        changing Mediators := Mediators ∪ m;
    Remove(m)     changing Mediators := Mediators - m;
    ...
  axiom attributes
    ICs initialized {};
    ...
  mutators
    ResetICs;
    AddICs(f:Formula);
    RemoveICs(f:Formula);
    ...
  dynamic specification
    ResetICs      changing ICs := {};
    AddICs(f)     changing ICs := ICs ∪ f;
    RemoveICs(f)  changing ICs := ICs - f;
    ...
end agent class Federator;
```

The following lines list some parameters for the `AddIC` event of **agent class** `Federator`. These parameters represent a certain kind of integrity constraints. By adding and removing integrity constraints we are able to change the behavior of a federator agent. In the case below, the federator agent may initiate a `Modify` event on a mediator agent `Mediator1`, or on a `Mediator2`, or on both of them.

```
{ Federator.Update(x) calling  Mediator1.Modify(x); }

{ Federator.Update(x) calling  Mediator2.Modify(x); }

{ Federator.Update(x) calling  Mediator1.Modify(x);
                               Mediator2.Modify(x); }
```

4 Open Questions

In this extended abstract, we presented an application area for the usage of agent technology. In the context of federated database systems, we sketched some fundamental properties of agents.

Further, we exemplarily showed how parts of such kinds of agents can be specified in an extended specification framework which allows to model fixed as well as changing behavior of agents.

The following items list some open questions which have to be discussed in future work:

- *agents versus traditional objects*: should we distinguish between agents and objects (as agents with fixed behavior and thus without any evolution capabilities)?
- *modeling an agent as a composition of "smaller" agents*: agents are complex "units" which have to be divided into smaller ones in order to be able to structure the agent's capabilities. Here, we have to identify basic properties of agents which are common to all kinds of agents. Further we have to find out a set of "aggregation/decomposition rules" for composing/decomposing agents from/into other agents which are responsible for certain tasks.
- *controlled behavior evolution*: which kinds of behavior evolution is necessary and which one has to be forbidden?
- *restricted autonomy of agents*: who may change the behavior of an agent? Does it make sense to have "self-adapting" agents? How can we express autonomy aspects in a specification language?
- *real-time aspects*: how can we integrate real-time into a specification framework in order to be able to express common requirements of agents, e.g. a certain request must be fulfilled in a given time interval?

In conclusion, on the one hand the agent paradigm opens a new perspective for modeling complex information systems, but on the other hand it lefts a lot of open questions which are "very" far from being solved.

Acknowledgements: We are grateful to the participants of the joint ModelAge SG2/4 meeting in Birmingham (July 1996) for the inspiring discussion.

References

[BGKK95] J. Bailey, M. Georgeff, D. B. Kemp, and D. Kinny. Active Databases and Agent Systems — A Comparison. In T. Sellis, editor, *Rules in Database Systems, Proc. of the 2nd Int. Workshop, RIDS'95, Athens, Greece, September 1995*, pages 342–356. LNCS 985, Springer-Verlag, Berlin, 1995.

[BJ94] C. Bussler and S. Jablonski. Implementing Agent Coordination for Workflow Management Using Active Database Systems. In J. Widom and S. Chakravarthy, editors, *Proc. of the 4th Int. Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE-ADS'94), Houston, Texas, USA, February 1994*, pages 53–59. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[Bro92] M. L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Systems. In P. M. Gray and R. J. Lucas, editors, *Advanced Database Systems, Proc. of the 10th British National Conf. on Databases, BNCOD 10, Aberdeen, Scotland, July 1992*, pages 1–28. LNCS 618, Springer-Verlag, Berlin, 1992.

[CS96] S. Conrad and G. Saake. Specifying Evolving Temporal Behaviour. In J. Fiadeiro and P.-Y. Schobbens, editors, *Proc. of the 2nd Workshop of the ModelAge Project (ModelAge'96), Sesimbra, Portugal*, pages 51–65. Departamento de Informatica, Universitade de Lisboa, January 1996.

[HM85] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[HM93] S. Hammer and D. McLeod. An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous Database Systems. *Int. Journal of Intelligent and Cooperating Information Systems*, 1(2):51–83, 1993.

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 14(2):175–211, April 1996.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

[SCT95] G. Saake, S. Conrad, and C. Türker. From Object Specification towards Agent Design. In M. Papazoglou, editor, *OOER'95: Object-Oriented and Entity-Relationship Modeling, Proc. of the 14th Int. Conf., Gold Coast, Australia*, pages 329–340. LNCS 1021, Springer-Verlag, Berlin, December 1995.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[TCS96] C. Türker, S. Conrad, and G. Saake. Dynamically Changing Behavior: An Agent-Oriented View to Modeling Intelligent Information Systems. In Z. W. Raś and M. Michalewicz, editors, *Foundations of Intelligent Systems, Proc. of the 9th Int. Symposium on Methodologies for Intelligent Systems, ISMIS'96, Zakopane, Poland*, pages 572–581. LNAI 1079, Springer-Verlag, Berlin, June 1996.

[WJ95] M. J. Wooldridge and N. R. Jennings. Agents Theories, Architectures, and Languages: A Survey. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents, Proc. of the ECAI'94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 1994*, pages 1–39. LNAI 890, Springer-Verlag, Berlin, 1995.