Modeling Database Federations in Terms of Evolving Agents*

Can Türker Gunter Saake Stefan Conrad

Otto-von-Guericke-Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {tuerker|saake|conrad}@iti.cs.uni-magdeburg.de

Abstract

For advanced information systems, the interoperation between heterogeneous database systems which autonomously manage their existing, independently developed databases is a fundamental issue. Thus, organization's data is virtually represented at the global level in a uniform way and transparent access to distributed and heterogeneous data sources becomes possible.

In this paper, we present a formal agent-based approach to modeling federated database systems. We identify some fundamental properties of agents which act as mediators between different, possibly heterogeneous and autonomous database systems. Finally, we briefly sketch an agent specification framework which may provide the basis for specifying highly dynamic and scalable information systems based on database federations.

Keywords: federated database systems, agent specification technology, mediator agents, federation agents.

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 on an increasing world-wide market [4]. 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 [9]. 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 [5]. Another very important aspect concerns the adaptability of information systems to changing requirements und highly dynamic

*This research was partially supported by the CEC ESPRIT Basic Research Working Groups ModelAge (No. 8319), ASPIRE (No. 22704) and FIREworks (No. 23531) and by the German Country Sachsen-Anhalt under FKZ: 1987A/0025 (Federating Heterogeneous Database Systems and Local Data Management Components for Global Integrity Maintenance).

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 been evolved [10, 15, 12]. 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 providing 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.

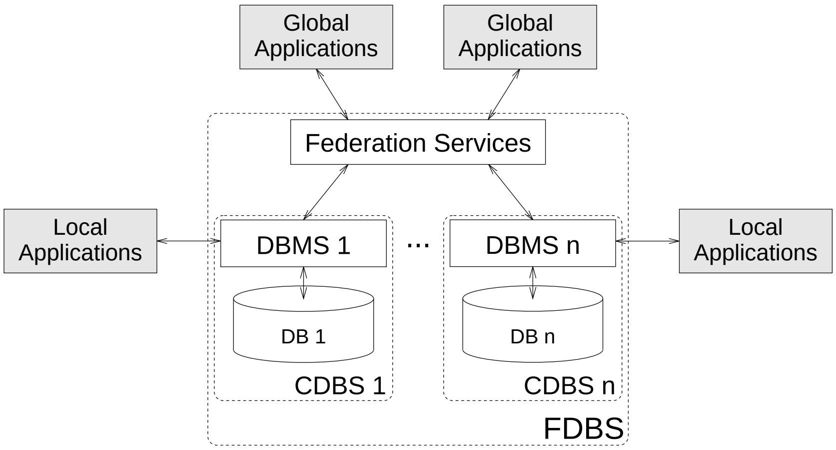

Figure 1: A simplified architecture for federated database systems

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, and 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 [17]. 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* objects (cf. [18, 1]), 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

There are several possible architectural approaches to federated database systems. In this section, we discuss two different agent-based architectures for the federation of autonomous heterogeneous database systems: a centralized and a decentralized approach.

2.1 Tight Database Federations using Federation Agents

As a first approach, we consider a database federation consisting 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. Analogously, 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 a federated database system, 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.

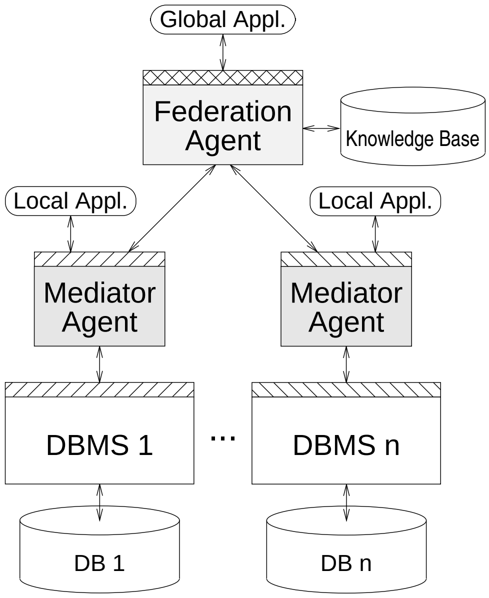

Figure 2: Agent-based federation of database systems

2.2.1 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:

- *Data model mapping*: A mediator has to transform data between different representations (data models). In fact, a mediator is responsible for the mapping between the (common) data model of the federation layer and the data models of the corresponding component database systems.
- *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. Please notice that the local applications should not interact directly with the original component database systems because of global concurrency and atomic commitment problems [3].
- *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. The maintenance of global semantic integrity is, thus, another reason why local applications should not directly interact with the original component database systems [16].

- *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.

2.2.2 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 schemata. Moreover, it must reflect changes in the component schemata as well as changes of global integrity constraints. In the latter case, it must be able to react on changes of integrity constraints of local schemata. In conclusion, the agent's behavior must be adaptable to evolving schemata.

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. For that, a query transformation may be required. Thereafter, these local queries are sent to the corresponding mediator agents which analyze these queries (for global concurrency and recovery control purposes) and propagate them to the original component systems. The results of the local queries must then be merged and returned to the user. For that, a mapping of the results into the common data model is required. The corresponding mapping tables as well as other metadata are managed by the federation agent in its knowledge base.

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 [8]. Therefore, advanced transaction models [7] with compensating and alternative (contingency) transactions must be supported at the federation level.

2.2 A Decentralized Approach to Database Federations

In some application scenarios, a decentralized approach to database federations is more appropriate. Especially, for global applications which consist of read-only transactions, e.g. for statistical computations, there is no need for a global central control in form of a federation agent. The problem of global atomic commitment and global semantic integrity is considered in such scenarios as not so harmful.

Figure 3 depicts a decentralized, agent-based approach to building database federations. In contrast to the architecture presented before, in this approach the mediator agents directly communicate with each other. There is no federation agent. The functionality of this federation agent is replicated on the mediator agents. Every mediator agents has to know about the other agents in the federation in order to fulfill requests of global applications which require data from or want to execute operations on different component database systems. Thus, each mediator agent acts to a certain degree as a federation agent.

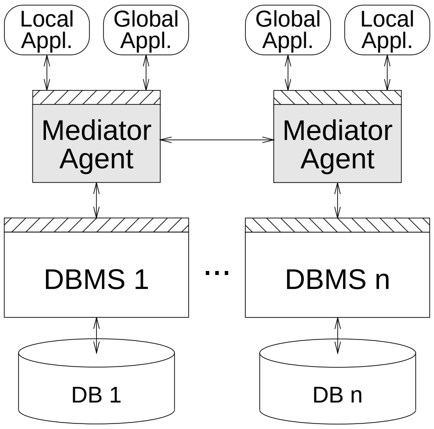

Figure 3: A decentralized, agent-based federation of database systems

Both approaches has several advantages and disadvantages. The centralized one with federation agents has the advantage that traditional database functionality like transaction, integrity and access control can be fully supported. The price which has to pay for that is the complexity of the federation agent. Furthermore, the federation agent can become the bottleneck of the system which may have also effects on the local applications. The favorite application area for the centralized approach are tight federations within an organization. In these scenarios, there is a high demand on the quality of data which has to be globally consistent.

The decentralized approach avoids the disadvantages of the centralized approach by skipping some of the functionality. On the other hand, the mediator agents are more complex compared to the other approach. Here, mediator agents are equipped with some of the functionality of a federation agent. A possible application area of this second, more loosely-coupled approach is the cooperation of several autonomous organizations which only allow the partners to read a certain part of their databases.

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. Here, we only consider the first database federation approach presented in the previous section.

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 to (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 [11]. 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 [13, 6, 17]. The logical foundations for the semantics of the language used here is given in [14].

3.1 An Agent Class Mediator

The following example sketches parts 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`. Further fixed events are `ReceiveQuery` and `SendQuery` which are used for communicating with the federation agents and the component database systems, respectively, in order to fulfill query requests.

```
agent class Mediator
  identification  ID: (Name, Location);
  attributes
    Name:         string constant;
    Location:     string;
    Federators:   set(|Federator|);
    ...
  events
    Update(s:string);
    Modify(s:string);
    ReceiveQuery(s:string);
    SendQuery(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;
```

3.2 An Agent Class Federator

The following lines show parts 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|);
    ReceiveQuery(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;
```

3.3 Executing Global Queries

A global query execution is initiated on a federation agent by calling the event `ReceiveQuery` with an parameter of type `string`. The implementation of this event first analyzes the query-string whether it is syntactical correct. Then the decomposition of the query takes place. Depending on the component databases concerned by the query, the query is decomposed in several parts and then sent to the corresponding mediator agents. By using this approach, we are able to easily change the way queries are executed. In database federations where several replicas of one real-world object are managed in multiple component databases, the federation agent can choose different strategies to fulfill global requests. For instance, depending on the work load of the mediator agents or the network communication costs, a query can be sent to exactly one, to multiple or to all relevant mediators. The current strategy can be easily changed by dynamically adding (or removing) formulas to (from) the specification of the federation agent. Thereby, the effects of the event `ReceiveQuery` are changed.

3.4 Behavior Evolution by Changing Integrity Constraints

The following lines list some possible 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 federation agent.

{ **count**(`Mediators`) $\leq$ `42`; }

{ **occurs**(`Remove(m)`) $\Rightarrow$ **not sometime**(**occurs**(`Add(m)`)); }

The first axiom defines a static integrity constraint which says that a federation agent may be connected to at most 42 mediator agents. The second axiom corresponds to a temporal integrity constraint. This constraint ensures that a mediator agent cannot be added to a federation agent from which it was removed before.

In this specification framework, integrity constraints as well as access rules can be easily formulated in a rule-based manner, for instance, in an implicative normal form. The implicative normal form has the advantage that ECA-rules or triggers can be derived in a very straightforward way for prototyping parts of a specification. The negation of such a formula in an implicative normal form corresponds directly to the event/condition which violates an integrity constraint.

3.5 Behavior Evolution by Changing Communication Strategies

Similar to integrity constraints, the communication strategies of a federation agent can be changed by adding (and removing) formulas to (from) the specification of a federation agent. In the case below, the federation 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); }
```

Changing communication strategies may also be required if there are agents in a network with different reliability degrees. This issue concerns the implementation of different read/write protocols for transaction management in distributed system environments. For example, a federation agent may propagate an update to all mediator agents which manage a replica of the objects which should be updated. This strategy corresponds to the *read once write all* protocol [2]. However, it may also be useful to use another strategy such as having a master copy of each real-world object for which a special, very reliable mediator agent is responsible. In this case an update is only propagated to the corresponding mediator agent.

4 Open Questions

In this paper, 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 advantage of an agent-based approach to modeling federated database system compared to traditional approaches without agents lies in the more appropriate capturing of dynamic aspects. Especially, the description of evolving behavior becomes possible.

In the following, we 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 leaves a lot of open questions which are, at least for the moment, "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

[1] 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. 2nd Int. Workshop, RIDS'95*, pages 342–356, LNCS 985, Springer, Berlin, 1995.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[3] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, October 1992.

[4] 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. 10th British National Conf. on Databases, BNCOD 10, Aberdeen, Scotland, July 1992*, pages 1–28, LNCS 618, Springer, Berlin, 1992.

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

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

[7] A. K. Elmagarmid, editor. *Database Transaction Models For Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[8] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On Serializability of Multidatabase Transactions through Forced Local Conflicts. In N. Cercone and M. Tsuchiya, editors, *Proc. 7th IEEE Int. Conf. on Data Engineering (ICDE'91), Kobe, Japan*, pages 314–323, IEEE Computer Society Press, April 1991.

[9] 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.

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

[11] 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.

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

[13] 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. 14th Int. Conf., Gold Coast, Australia, December 1995*, pages 329–340, LNCS, Vol. 1021, Springer, Berlin, 1995.

[14] G. Saake, A. Sernadas, and C. Sernadas. Evolving Object Specifications. In R. Wieringa and R. Feenstra, editors, *Information Systems — Correctness and Reusability. Selected Papers from the IS-CORE Workshop*, pages 84–99, World Scientific Publishing, Singapore, 1995.

[15] 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.

[16] C. Türker. An Approach to Supporting Semantic Integrity of Federated Databases. In *Heterogeneous Information Management, Prague, Czech Republic, 4–5 November 1996, Proc. 10th ERCIM Database Research Group Workshop*, pages 51–60, ERCIM-96-W003, European Research Consortium for Informatics and Mathematics, 1996.

[17] 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. 9th Int. Symposium on Methodologies for Intelligent Systems, ISMIS'96, Zakopane, Poland*, pages 572–581, LNAI 1079, Springer, Berlin, June 1996.

[18] 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. ECAI'94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 1994*, pages 1–39, LNAI 890, Springer, Berlin, 1995.