Agents and Databases: Friends or Foes?

Peter C. Lockemann, René Witte
Fakultät für Informatik, Universität Karlsruhe
lockeman|witte@ipd.uka.de

Abstract

On first glance agent technology seems more like a hostile intruder into the database world. On the other hand, the two could easily complement each other, since agents carry out information processes whereas databases supply information to processes. Nonetheless, to view agent technology from a database perspective seems to question some of the basic paradigms of database technology, particularly the premise of semantic consistency of a database. The paper argues that the ensuing uncertainty in distributed databases can be modelled by beliefs, and develops the basic concepts for adjusting peer-to-peer databases to the individual beliefs in single nodes and collective beliefs in the entire distributed database.

1. Introduction

Agents and databases seem to be worlds apart. Agent technology has for a long time been the domain of distributed artificial intelligence (DAI), with a focus that extends beyond mere technical characteristics towards "soft" properties such as social behavior. Database technology is an engineering discipline that centers around persistent storage of vast and ever growing repositories of information.

But agents and databases also seem like ideal complements: Agents carry out information processes whereas databases supply information to processes. What's more, agent communities seem a natural model for loosely coupled distributed systems, while database technology sees more and more of its mission in the support of highly distributed information systems.

Nonetheless, there has been little collaboration between the two worlds so far. Although agents must carry a view, however limited, of their environment and, hence, should include something akin to a small database, database technology does not seem to have a part in agent technology. And database technology seems to have little love left for a technology that deals with elements of unpredictable and intangible behavior.

Database technology is a mature technology. Agent technology seems to come of age. Consequently, there is every reason to study whether the combination of the two provides better value, how one can benefit from the other, or how one affects the other. This paper will concentrate on one direction in the question, the effect that agent technology has on database technology.[1] To study the effect the paper will be divided into two main parts: A first part giving a short outline of the pertinent properties of agents and agent communities, and a second part examining whether adjustments must be made to database technology and what these are.

Part I: Agents

2. Agent properties

The prevailing characterization of software agents comes from distributed artificial intelligence and seems well agreed upon. We take the characterization from the seminal book by Wooldridge ([32], for further details see also [10] and [30]). Wooldridge differentiates between minimal properties, i.e., properties that each agent ought to have, and those of intelligent agents.

2.1. Minimal properties

He lists as the minimal properties the following.

(1) *A software agent is a computer system that is situated in some environment.* If this sounds trivial because it applies to any piece of software, one has to read the fine print, so to speak, in Wooldridge's book where he adds that "like its real-world counterparts,

[1] Part of this work was supported by the German national research initiative on "Intelligent software agents and business application scenarios" (DFG SPP 1083).

agents do not dissolve once a task has been finished, rather the interaction is an ongoing, non-terminating one." Also, "some" is used in the sense of "special" and, hence, the property includes the characterization of *the agent restricting itself to a certain domain*. Figure 1 symbolizes the continuous lifecycle of an agent sensing the environment and acting on the environment.

(2) *A software agent offers a useful service. Its behavior can only be observed by its external actions, its internal processes remain encapsulated.* This is a property that does not differ significantly from the notion of software component, but at least it defines a bridge to a software engineering technology.

(3) *A software agent is capable of autonomous action in its environment in order to meet its design objectives, i.e., to provide its service.* Indeed, this property sets agents apart from other software: The agent is able to act even without the intervention of humans or other systems, and it has a certain degree of latitude in performing a task, e.g., the liberty to decline the task or pose conditions, or to break off its work, perhaps incurring some penalty, or to decide when to start or complete work. From an outsider's standpoint the property seems to confer on agents unpredictable, or non-deterministic (if not outright chaotic) behavior.

(4) *The autonomy of a software agent is determined by its own goals, with goal deliberation and means-end assessment as parts of the overall decision process of practical reasoning.* The property can be seen as a kind of counterbalance to property 3, insofar as it introduces some predictability into an agent's behavior: The agent's own goals influence its decision process which of the potential actions it should perform and when.

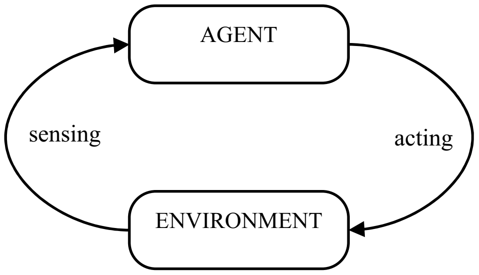

Figure 1 Agent Lifecycle

2.2. Intelligent agents

Non-deterministic behavior of software is not something that is particularly appreciated in software engineering. It may offer some strength, though, if the environment, due to its complexity, may itself appear non-deterministic to the agent. Hence, following Wooldridge we define:

An intelligent software agent is a software agent capable of operating in a non-deterministic environment.

Wooldridge states a number of consequences.

(5) *An intelligent software agent is reactive, that is, it continuously perceives its environment, and responds in a timely fashion to changes that occur.* The property reflects two obligations for the agent: To observe the environment for requests or other changes that require some reaction, and indeed to show a reaction where one is expected.

(6) *An intelligent software agent achieves an effective balance between goal-directed and reactive behavior.* The need for reactivity seems to contradict autonomy. In general, though, there may be several ways on how to respond, or the need to respond may even be in conflict with the goals. Consequently, the response should be consonant with the agent's goals.

(7) *An intelligent software agent may be proactive, that is, take the initiative in pursuance of its goals.* The agent may also take its own initiative to effect changes in the environment in order to further its own goals.

(8) *An intelligent software agent may have to possess social ability, that is, should be capable of interacting with other agents to provide its service.* Agents may become clients of other agents, for example when they cannot solve a given problem on their own but must delegate part of the problem solution to other agents.

One may argue that goal deliberation and means-end assessment, e.g., plan selection is fairly meaningless unless the agent is able to learn from past experiences in order to improve on future solutions. Nonetheless, Wooldridge treats

(9) *An intelligent software agent may be able to learn.*

as an optional property.

3. Multiagent systems

The idea behind multiagent systems – entire collections of agents – is that the whole is more than the sum of its parts [7, 32]. Consequently, there must be a purpose behind the system. In a *multiagent system* the purpose is the pursuit of an overall goal. Under the assumption that the individual agent is self-interested and pursues its own, low-level goals, Weiss ([30], p.3) lists four major characteristics of multiagent systems:

1. Each agent has just incomplete information and is restricted in its capabilities.
2. System control is distributed.
3. Data is decentralized.
4. Computation is asynchronous.

Item 1 simply states that a single agent's contribution to the overall goal is limited. Item 2 suggests that a multiagent system follows a peer-to-peer organization. Item 4 implies that agents are only loosely coupled, that is, interact by message exchange, and item 3 that much of the information exchange is to bring the agents' data up-to-date.

The presence of a shared (high-level) goal requires some sort of organization to overcome the individualism of the agents. Message exchange is the primary means for the agents to cooperatively pursue the high-level goals. Since there is no central authority that enforces the high-level goals, the agents must agree among themselves what the rules of engagement are. These rules define their *interaction protocol*.

Classical interaction protocols in multiagent systems are the blackboard system and the contract net protocol. The former requires – at least in principle – a centralized blackboard and control component, and problem solving is purely sequential. Much better suited to self-organization is the contract net protocol because there is no central authority, and agents may act as manager at one time and as contractor at another, or even simultaneously as when a contractor for a specific task acts as a manager by soliciting the help of other agents in solving parts of the task.

Although there is no natural central authority in a multiagent system one can easily introduce one. For example, why should agents not decide to entertain a central database that could establish a common view of the environment? The way to do this in a multiagent system is to choose one of the peers to manage one, and let them enter into agreements among themselves so that they will supply the necessary data to the chosen peer. Likewise, agents could agree on the use of a blackboard system and would have to install a peer agent for controlling the blackboard. We refer to such agents with central authority as *resource agents*. However, the more responsibilities we shift to resource agents the lesser autonomy is left to the individual agents.

Part II: Databases

4. Agents and databases

Agents must carry a view, however limited, of their environment as the basis for decision making. Due to the long-lasting agent lifecycle the view must be maintained over time and even across disturbances, i.e., the view should be durable. Consequently, there are in the individual agent all the ingredients of a local database no matter how small, and if we take an entire community of agents that form the complete system what we have is a highly distributed database.

Now, as we saw before an agent certainly is much more than a database. However, for the purpose of studying the effects of agents on database technology we abstract agents to just that, local databases. The first question is how the abstraction from the agent properties is reflected in their databases, i.e., whether these are ordinary databases or databases with special qualities, and hence whether a multiagent system presents us with a distributed database that has distinctive characteristics. From what was said in Part I we submit as characteristics:

- Multiagent systems form a huge, highly distributed database where each constituent database is small with an extremely limited view of the mini world.
- The database is an open system where constituent databases may come and go.
- The constituent databases are fully autonomous, there is no central control.
- Due to interaction the constituent databases may overlap in content.
- The activities of agents, and their cooperation, is centered around their self-interest in the form of individual low-level goals and collective high-level goals.
- The constituent databases may show nondeterministic behavior by responding with delay or not at all to input and with answers that are only predictable within limits, and by taking unpredictable initiatives on their own to interact with their environment.

On first glance these characteristics seem to conform to those of peer-to-peer databases. These have the properties [1]:

- no central database,
- no peer has a global view of the system,
- peers are autonomous,
- no central coordination,
- global behavior emerges from local interactions,
- all existing data and services should be accessible,
- peers and connections are unreliable.

And indeed, agent and peer-to-peer organizations agree in several fundamental respects: Autonomy of the peers, lack of a global view, decentralized control, data distribution, global behavior as a result of individual interactions, and variable connectivity. But there is more to the agent world: Non-deterministic and at best qualitatively predictable behavior of the peers with a concomitant effect on the interactions and the resulting behavior of the entire system. Furthermore, peer-to-peer information systems and agent systems pursue complementary objectives: For the former the predominant objective is discovering information [1, 2], for the latter information is just the means to act towards an objective. Therefore, we do need to take a fresh look at the database world.

5. Consistency, uncertainty and beliefs

5.1. Consistency and uncertainty

The central paradigm of database technology is *semantic consistency*: Databases should always give answers that reflect true states of the real world. Database systems guarantee semantic consistency through transactions that ensure that any new state is a true and current image of the outside world.

But is semantic consistency a realistic proposition in a large-scale peer-to-peer organization? After all, no agent is ever supposed to observe the entire environment, and even collectively the agents will never observe the environment at the same instance in time. Hence, there is neither individually nor collectively a consistent image of the environment. This reminds one of the uncertainty principle in physics which states that predictions can be made only within certain probabilities (see [11]). Consequently, what such systems need to incorporate is what Pearl calls *uncertainty management* [23].

On first glance, uncertainty management seems to reflect the needs of peer-to-peer organizations in general. But that is not the common view of peer-to-peer systems. The literature never treats peer-to-peer systems as a collective if distributed database. Rather the individual members are considered completely independent and the emphasis is on finding the desired information. In multiagent systems, on the other hand, the members cede some of their autonomy to the common good and, hence, need to base their collaboration on some common understanding of the real world.

5.2. Uncertainty and beliefs

Taken as a whole, a distributed peer-to-peer database is inherently uncertain. Or in terms of agents as the peers, we cannot expect each individual agent to share a common, unanimous view of the world. On the other hand, the common interests of agents will motivate them to pursue as much certainty as needed in view of their self-interest. In recognition of this dilemma, agent-oriented programming relies on mentalistic notions such as belief, desire, and intention. According to Bratman [6], “practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires (values) and what the agent believes.” Beliefs are what the agent knows or assumes to know about the outside world. Intentions describe some future state of affairs the agent will attempt to achieve. Desires reflect the agent’s goals. The three notions give rise to the belief-desire-intention (BDI) architecture which is at the heart of many agent architectures (see Figure 2 for an example).

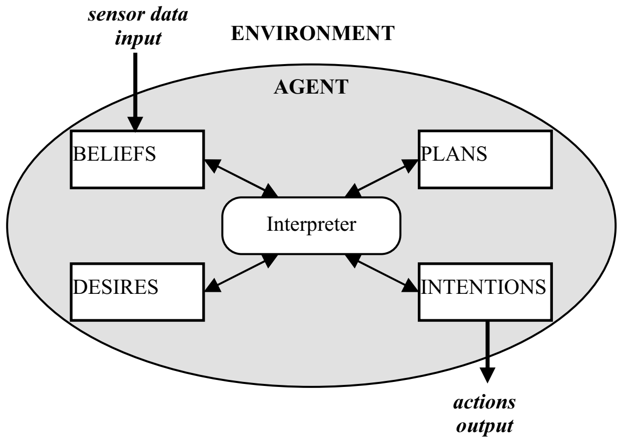

Figure 2 BDI architecture in the Procedural Reasoning System PRS (from [32])

Consequently, we introduce beliefs as our model of uncertainty. The database local to the individual software agent reflects its beliefs about its environment rather than a true and current image of the environment. Then, how could one interpret the collective, distributed database of a multiagent system? After all, beliefs may contradict one another, particularly if some parts of the environment overlap in the various agents.

In the remainder we separate the two issues. We first study how the individual agent will deal with its beliefs. With that knowledge we proceed to study how the multiagent system arrives at a collective belief. As a working definition we refer to a distributed database represented by a multiagent system as a *belief database* if *each software agent carries beliefs about its environment, and the multiagent system has a collective belief that tolerates contradictions, but only to a degree that does not inhibit the pursuit of the common goal*.

5.3. Querying beliefs

Beliefs are due to incompleteness of information. In incomplete databases the closed-world assumption can no longer hold. Hence, in querying an incomplete database one must apply a process referred to as belief reasoning [17] (or plausible reasoning [23]). In the distributed world of a multiagent system the process includes theorizing about the beliefs of other agents, i.e., forming an opinion about the data visible to those agents. Opinions may correspond to, e.g., optimistic, firm, or cautious views. Another example of querying incomplete databases is speculative computation [26]. Nonetheless, as the agents interact they may gain explicit knowledge of the beliefs held by other agents, and should treat these as new observations that cause them adjust their own local database.

6. Individual beliefs

6.1. Structuring beliefs

An agent must be able to store and manipulate his own beliefs, but this alone does not suffice. In order to collaborate on a task with other agents he needs to incorporate their beliefs as well. However, these beliefs may be in conflict with his own beliefs due to their owners' focus, status, or task.

Consequently, we need to allow an agent to hold his own, unique view of the world, while at the same time enabling it to incorporate beliefs from other agents in order to collaborate on a task. In addition, an agent may want to distinguish beliefs stemming from trusted sources (like its master) and untrustworthy sources (like a foreign e-commerce agent), assigning them different certainty or credibility values, which influence subsequent reasoning. How can we achieve this?

A model for this scenario has been proposed by Ballim and Wilks [3]: Beliefs are structured into *viewpoints* for representing a particular agent's point of view and *topics* for collecting beliefs that are relevant to a given subject. The two can be *nested* within each other. For example, viewpoints can contain other viewpoints such as the first agent's view of another agent's viewpoint, or topics such as the agent's viewpoint on a particular subject. Likewise, a topic can contain beliefs pertaining to its subject, but also other viewpoints, e.g., when the subject is another agent or a class of agents.

The model is particularly elegant because an agent can explicitly distinguish between its own beliefs on certain topics and, at the same time, hold (possibly conflicting) beliefs about what *he believes other agents believe* about the same topic, which are contained within different, nested viewpoints. Moreover, the agent may hold one viewpoint that is logically consistent and a second with conflicting information about the same topic. This enables an agent to collect and maintain as much information as possible, attributing them to their sources, without having to decide on a "correct" set of beliefs, thereby losing information prematurely.

6.2. Adjusting the belief structure

In the nested model a group of agents can dynamically reconcile their beliefs about a given number of topics in order to collaborate, temporarily giving up some of their own beliefs within a nested viewpoint in order to reach a consensus, but still holding on to these beliefs in their primary view. This, however, requires mechanisms for *constructing* as well as *destructing* viewpoints.

An agent can create viewpoints about *other* agents' views on a given topic through the process of *belief ascription.* The default ascription rule is to assume that another agent's view is the same as one's own. If there is explicit evidence to the contrary, this knowledge overrides the default rule, which may prompt the agents to further reconcile their views. Ascription also allows an agent to incorporate beliefs from another agent.

Belief percolation deconstructs nested beliefs. For example, a belief about another agent's belief may be acquired as one's own belief by percolating it from a lower level all the way up to the top-level viewpoint of the agent.

The model does not prescribe a single approach for the representation of beliefs: All formal language can be adapted to model beliefs on certain topics. However, the processes of ascription and percolation will need to be adapted for each representation formalism.

6.3. Belief revision

Updates in the classical database world focus on one scenario, changes in the real world that impose corresponding changes in the miniworld. However, as Katsuno and Mendelzon [18] point out, this is not the only situation where a change in an agent's beliefs is warranted. When an agent makes a new observation, or receives beliefs from other agents, the world didn't necessarily change, he just obtained more information about it. In this case he does not want to simply replace his old set of beliefs with the new ones (what a simple database update would do), but rather *revise* his set of beliefs. This reflects what Harman states as "when changing beliefs in response to new evidence, you should continue to believe as many of the old beliefs as possible" (quoted [31], section 9.1.2).

The field of *belief revision* distinguishes three operators for such changes. Suppose an agent makes a new observation. Then one action could be to simply add the observation as a new belief no matter whether it contradicts existing beliefs or not. The corresponding operator is referred to as

- Expansion. Simply adds a new proposition. As a consequence, the database may now contain incompatible information.

Note that the nested model of Section 6.1 allows for conflicting views, hence expansion is indeed a useful operator. More often, though, an agent will not tolerate local contradictions but rather resolve them during ascription or percolation. Two operators follow a resolution principle:

- Revision. Adds a new proposition. If the information is incompatible with the previous state then older contradictory information is removed. Revision is non-deterministic because there may be more than one way to restore compatibility, and it may be unknown which is chosen.
- Contraction. Removes a proposition that until now was considered valid. Contraction may trigger further removals until no incompatible proposition can further be derived. Hence, contraction is also non-deterministic.

Revision and Contraction are not only non-deterministic but also non-monotonic, but they do remove contradictions. Expansion preserves contradictions, and this certainly makes sense if we wish to defer resolving them, e.g., until the agent had a chance to communicate with others. In general, though, no matter whether contradictions have been resolved or not, the multiagent system as a whole may contain contradictions. This raises the question of whether and when a multiagent system should resolve global contradictions to arrive at something like common beliefs.

6.4. Formalizing beliefs

What are suitable data models that allow to inquire contradictory or incomplete information and that can give specific semantics as well as a formal basis for the update operators?

Classical databases are logically consistent: They do not tolerate contradictions. They enforce logical consistency by overwriting the old information, and to the extent semantic consistency can be expressed in terms of the database schema and consistency constraints the new information is checked against these. For semantic consistency in general they rely on transactions. Consequently, theories in classical databases will offer little in support of beliefs.

Which requirements should a formalism for belief databases meet? Our working definition in Section 5.2. limits logical contradictions and semantic inconsistencies to those that do not inhibit the pursuit of the common goal. The formalism, then, should cover both, logical and semantics inconsistencies. Further, an agent's actions may have real-world effects. In this case internal and external states must either not diverge at all or only over limited time, like in debit/credit transactions. Consequently, belief databases should allow for a continuum from vague beliefs all the way to rigid semantic consistency.

Wooldridge suggests modal logic as a formalism to characterize, and reason about, the mental states of agents as they act and interact [32]. He discusses two models for interpreting the logic, epistemic logic and possible-world semantics. Both exhibit certain shortcomings, particularly if – as a first step towards the aforementioned continuum – one distinguishes between knowledge which must be non-contradictory, and beliefs which may hold contradictions.

Another, more operational approach is due to Witte [31]. In his representation formalism, individual beliefs are modeled as fuzzy propositions over a discrete set of (not necessarily numerical) values. A set of beliefs places an elastic restriction on the possible interpretations of a variable. The fuzziness allows to explicitly capture uncertainty, which allows to move from a notion of strict consistency to a gradual *consistency degree*. An agent can dynamically adapt its consistency degree over time: arriving fresh onto a scene, it might want to absorb many, possibly contradicting beliefs, while being more strict for contexts established over a longer period of time.

Fuzzified versions of the operators introduced above allow to modify belief sets. A *fuzzy revision* will admit new beliefs even when they are inconsistent with the existing beliefs, if the remaining consistency reaches at least the prescribed degree. A degree of one implies complete consistency (as in a non-fuzzy framework), a degree of zero admits even completely inconsistent beliefs. Practical systems will deploy intermediate consistency degrees, where a revision removes existing beliefs if they are too incompatible. With these fuzzy operators, an agent will be able to tolerate gradual inconsistencies while maintaining beliefs within the processes described above. As with the dynamic construction of viewpoints, the decision on just how much inconsistency can be tolerated within a given situation is left to the agent while it interacts with the environment.

Grounded in the tradition of base revision, the model foregoes an inference operator in favor of highly efficient algorithms that are suitable for large-scale belief sets and frequent updates. This follows the idea that modifications to a belief set within a database will be primarily based on external processes rather than internal inference engines.

Other approaches that merit closer inspection are probabilistic databases, and Bayesian belief networks.

7. Robustness, atomicity and correction

7.1. Failure models

Now that we have an idea how the individual agent deals with its beliefs we can proceed to study how the multiagent system arrives at a collective belief. We demonstrate first that the issue is affected by another central property of database systems, robustness in the presence of disruptions, failures, incursions, or interferences.

Robustness must always be founded on a failure model. The failure model states what it considers the correct outcome of a service request and how the service provider achieves correctness in the case of failure, perhaps with the help of the client.

In classical databases correctness is defined as semantic consistency of the database. This leaves little leeway to the failure model: We expect it to maintain consistency under any circumstances. Since transactions embody consistency, the failure model is translated into the property of transaction atomicity. As a first step we return the database to the last state to be known as consistent. If the transaction has a real-world effect the recovered database may now diverge from the state to which the external world progressed in the meantime. The failure model must then be augmented by compensation (undo real-world effect) or correction (advance database to semantic consistency) (see, e.g., ConTracts [15]).

But what is a meaningful failure model if beliefs replace semantic consistency? We now lack a stringent correctness criterion. What we gain instead is considerable leeway as to which state to reach.

We will discuss below how to use the leeway without deviating too far from classical database techniques. In doing so we follow Pleisch and Schiper who in a survey paper on (albeit mobile) agents make a clear distinction between infrastructure failures and unfavorable outcomes [24]. They point out that only transactions – more specifically their property of atomicity – protect against both, the former by crash recovery, the latter by transaction recovery. Less benignly speaking, classical transactions fail to grasp the difference and apply one and the same failure model to both.

7.2. Infrastructure failure

Take first the individual agent. It seems incumbent to return the agent to a well-defined state. The proven way to do this is classical rollback. Consequently, an agent should be transactional with regard to infrastructure failures. On the other hand, the agent started from some belief rather than consistency. Hence, correction seems unnecessary because the failure model should tolerate uncertainty as long as it remains within the confines of consistency as exemplified in Section 6.4. . The agent may simply hold beliefs about the environment that are perhaps wrong in ways different from before.

But what if failure occurred while several agents were collaborating? In distributed transactions global commit and global abort are enforced by some two-phase commit protocol (2PC) variant. But 2PC protocols run counter to the autonomy of its participants because they temporarily tie the participants' actions together, sometimes even blocking them altogether. Nor does the approach by Pleisch and Schiper seem helpful because they guard against infrastructure failures in a system of mobile agents participating in a distributed action by intruding into the agents' autonomy: They replicate subactions on agents, with the various commit protocols somehow enforcing exactly-once execution.

This seems to leave as the only solution transactional atomicity of the individual agents. But now there is a real danger that beliefs start to disagree beyond what seems tolerable. Consequently, infrastructure failures should somehow be made known to the agents in a collaboration, and leave it to them how to cooperate in adjusting their local databases. A canonical extension of the model described above is to allow agents to hold and manipulate beliefs about the infrastructure as just another topic of their discourse world.

7.3. Unfavorable outcome

An outcome can only be called unfavorable if some system component takes an action whose result has not entirely been expected by its client, i.e., if the servicing agent did not reach its objective. Hence the notion of unfavorable outcome seems only to make sense if the "failing" agent's action is part of a distributed action. Even classical transactions have doubts whether global abort following, e.g., 2PC is the right answer, and resort to, e.g., nested transactions. One may as well argue that the client simply held the wrong beliefs, and all it has to do is to revise them. Consequently, there is no need for any rollback on the part of the client but a need for immediate revision.

7.4. Concurrency

Concurrency control in the form of isolation of transactions is about control of unwanted interaction, and surely we would like to minimize unwanted interaction in a multiagent system as well, or avoid it altogether. But at what price?

Let us start with the individual agent. If it acts on one request a time then there clearly is no problem. But nothing in the definition of agent keeps it from servicing several requests at a time – after all it is autonomous and may thus decide to accept several requests. It is also free to decide when and in which order to act on them, or whether to run them concurrently or not.

Intuitively, one would like to avoid any interference among concurrent requests, because if we did not then the already existing indeterminism would grow to unmanageable proportions. Consequently, for the individual agent serializability of requests seems to remain a good strategy in general. In particular, though, the serial order, and, hence, the scheduling strategy may have to reflect the agent's goals. The agent may also deem some kind of local interaction permissible – take relative serializability as an example [29].

But what about the multiagent system as a whole? Several agents interact while pursuing a shared objective. If we wish to make sure that other agents do not interfere, we would have to encapsulate the interaction within a single global, distributed transaction. But then, strict isolation requires that the participants temporarily relinquish their independence. Since the participants rarely agree to do so, one would have to resort to weaker forms of global serializability known for heterogeneous federations [29]. But even these require some form of common control, and because agents behave somewhat unpredictably, such control could easily block the system altogether.

Consequently, it seems more realistic to run the risk of some unwanted interaction and to accept that the beliefs in the various agents may grow further apart. For example, we may entirely drop control from within the global transaction, and commit is considered just a signal to readjust the beliefs across the participating agents.

8. Collective beliefs

8.1. Belief reconciliation

To summarize, a distributed database based on beliefs leaves a lot of slack to flexibly react to a changing environment – precisely what motivated the use of multiagent systems in the first place. We used the slack to forego the strict control of classical transactions and to allow beliefs to depart from the real world within the single agent, and to diverge across a community of agents.

To add some controlled resilience to the individual agent it is made transactional. Equally important, to ensure that the state of the entire distributed belief database limits logical contradictions and semantic inconsistencies to those that do not inhibit the pursuit of the common goal we have to adjust the individual databases. We refer to the adjustment process as *belief reconciliation*. In a nutshell, reconciliation enforces on a global scale what update operations already enforce locally, some weaker notions of consistency.

8.2. Reconciliation process

When should belief reconciliation be initiated, and by whom? As noted above, global abort or global commit may be such an event, and any participating database may start the reconciliation. Of course, in a long-running transaction one may not wish to wait until the end, and conversely one may tolerate a larger divergence and readjust the beliefs only at longer intervals. Or a database receiving an unfavorable outcome may inquire with other databases to readjust its beliefs, and it may also do so if it has to violate its degree of consistency. If none of those events occur over a long time beliefs may diverge more and more across agents so that at least periodically we need a signal to close the gaps between the beliefs across the distributed database.

In summary, belief reconciliation must be started by a single agent, either one that updated its beliefs or that has every reason to suspect that it needs an update. Take an agent that wishes to "push" its update. Can the agent limit the communication of its changes to those agents to which they are *acquainted*? How can it be sure that only these are affected? Or should it simply flood the multiagent system and leave it to the individual agents to decide whether to revise or contract their beliefs? Is there a quick way to bypass those agents that are not interested, something we may call a *context* that describes the current preferences of an agent? Suppose that a recipient agent adjusts its beliefs. Does the change have a new effect on the other agents? Do we have a kind of snowball effect, with each agent being stimulated into action? How, then, does the process come to an end? And since agents are autonomous, how can one force individual agents to take timely adjustment action?

Take instead an agent that needs to "pull" beliefs from others. These others may in turn have to inquire with further agents and in the process revise or contract their beliefs. Again the process may spread, with many questions similar to the ones above.

The process may also come to an abrupt end. Suppose an agent tries to convince other agents to see the world the same way they do, i.e., to share their beliefs. For example, among the FIPA communicative acts are some where an agent sends a proposition to another agent in the belief that the other agent either does not know or is uncertain of [12]. However, instead the other agent may reject it and convince the original agent otherwise, or the agents enter into a dialog before settling on a particular effect of the reconciliation process.

8.3. Protocols and models

The challenge, then, is to develop protocols that reflect the spread and interaction of beliefs with all the variations discussed before. But where to look for them?

Even in classical databases where all components act entirely deterministically, uncertainty is known to arise in conjunction with cache management. Generally speaking, the objective is exact replication of data, however currency of data may be relaxed to periodic refreshing [16]. As a concrete example of a centralized approach, take cache coherence protocols that follow a client-server paradigm [28] or global cache manager [5]. The protocols are supposed to supply cache users with data that are as fresh (are exact replicas of some master copy) as possible. Clients may poll the server to ensure freshness, or servers may send invalidation messages to the clients. If clients are prepared to accept some stale data, polling or invalidation may be delayed (soft coherence). Closer to a peer-to-peer situation come the data replication protocols of parallel database systems. Refreshing the replicas is always initiated by the node where the change took place, and is communicated to the other nodes with delays that vary according to the protocol used ([19], chapter 14). These protocols, however, assume a small number of nodes.

Closest to our needs comes a recent approach by Bernstein and others for data management for peer-to-peer computing [4]. They still insist on local consistency but accept inconsistency across the network. Each peer maintains a – perhaps dynamic – list of acquaintances together with coordination formulas that explain how the data in one peer may relate to data in an acquaintance. These formulas are used to either query acquaintances or propagate updates to acquaintances.

Propagation models have been studied in disciplines outside informatics as well and may be useful for our purpose. One interesting area are models for the spread of infectious diseases, so-called epidemiological models. The models assume that some fraction of the population is susceptible to the disease, and that transmission between individuals also takes place with less than 100% efficiency. The various – mostly analytical – models differ in the strength of the bonds between the individuals (see, e.g., [14, 21]). They agree in their simplistic characterization of the individuals: if they are affected they die. Also common to all of them is their bimodal behaviour: There is a sharp threshold in the dissemination parameters' values for which a reliable delivery is ensured with a high probability.

Epidemic protocols seem to hold a particular fascination to researchers in distributed computing as long as simple models for the nodes suffice. For example, Eugster et al. claim in a survey article [9] that epidemic algorithms have been recognized as robust and scalable means to disseminate information in large-scale peer-to-peer settings. They expect that so-called *epidemic information dissemination* scales to large systems provided one can meet the challenges of scalable peer-to-peer application level multicast protocols. A concrete example is given by Rabinovich et al. [25]. User operations are performed on a single replica. Asynchronously, i.e., at a convenient time, a separate activity termed anti-entropy compares version information of different copies of data items and propagates updates to older replicas. Since in principle anti-entropy must periodically compare each pair of data items, the foremost challenge is to find solutions that do indeed scale well. For example, propagation may be scheduled in such a way that every node eventually performs the propagation transitively from every other node. The protocols include a so-called out-of-bound update propagation that can be demanded by a single node to refresh its copy.

Belief propagation poses further challenges. A simple model for the nodes won't do, while on the other hand the susceptibility of the nodes to the dissemination is low, and exists only if the new information is relevant. Relevance is modelled by the so-called *context* of the node. Coutaz and others point out the importance of contexts in an ever-changing environment [8]. A fascinating article by Schmidt and Gellersen draws first operational conclusions [27]. They refer to context as an abstract description of the local situation of a node. There are context producers and consumers. Context must be propagated throughout a distributed system, with a propagation model where the importance of context diminishes across time and space. All these properties are reflected in an abstract model, the fuzzy (state) space. Fuzzy sets also underlie the recent paper by Gal et al. on semantic reconciliation [13].

Interestingly, although it is well accepted that agents may be endowed with disparate long-term knowledge, and that coordination of agents requires some sort of conflict resolution, hardly any approaches on reconciliation have appeared so far. In [22] (pp. 59-61 and 133) conflicts are only resolved among a limited number of agents, either when they explicitly cooperate, often using a centralized interaction protocol such as a blackboard, or when an agent receives messages and compares them to the model it keeps of its acquaintances.

9. Reliable message exchange

All the approaches in Section 6. through 8. must exercise enough control to guarantee that logical contradictions and semantic inconsistencies do not inhibit the pursuit of the common goal. To be sure of their own guarantees they need some lower-level guarantees by the underlying technical infrastructure. Since interaction protocols are at the heart of pursuing the common goals, message exchange – and perhaps its higher form of publish/subscribe – must definitely be reliable. To guard against message loss the proper answer seem queued transactions [29]. They consist of three transactions, one at each participating agent that includes queuing and dequeuing, and one within the queue manager (persistent, recoverable message queues). Consequently, a persistent queue manager seems an absolute necessity for a multiagent system. And as a consequence, each of the agents participating in a message exchange should run in a transaction – exactly what they already are recommended to do to cope with infrastructure failures.

Part III: Friends

10. Conclusions

On first glance agents seem more like foes to databases, hostile intruders into the database world. To view agent technology from a database perspective seems to question some of the basic paradigms of database technology, particularly the premise of semantic consistency of a database. And as a result, transactions seem to lose some of their lustre, and entirely new update propagation protocols should be developed for a highly distributed world.

On second thought, though, agents and databases seem more like friends, like complementary technologies. Uncertainty should be seen as a phenomenon that covers a continuum from entirely certain to some limit on acceptable uncertainty. Semantic consistency is by nature difficult to maintain in distributed systems, and database research seems to accept that the assumption of consistency should be relaxed. Replacing semantically consistent databases by the notion of belief appears to offer a uniform model for covering the entire range between applications that work only under strict consistency and those where there is not the slightest chance nor need to maintain more than some weaker form of formal consistency.

Nor is the issue purely academic. The authors believe that the flexibility of agents, and the slack in belief databases, is of critical importance for the practical world because both carry all the potential for systems that are much more resilient to technical failures as well as catastrophic events or malicious attacks in the real world. Lately, some researchers start to wonder whether it is not an inherent property of nature that allows living organisms to adapt to continuous or abrupt changes (evolution). Taking the cues from natural organizations they argue for a new paradigm of self-organizing software.

Agent technology is an expensive technology, and has its problems to become accepted in software engineering. And so would be belief databases. An apparent reason seems to be the price one would have to pay for the non-determinism. There is empirical evidence, though, that multiagent systems are superior to more traditional software when both the problem space and the solution space become too large for practical enumeration [20], something we might call a *complex application situation.*

Complex situations are characterized by the failure to write down a complete decision table or case statement. Indeed, this sounds familiar to the database community. Their original motivation for performance optimization was that the system implementation is way too complex and offers too many influential factors as for the user to determine the best execution plan for a given request. Instead the user is asked to submit a descriptive request that merely states the properties desired of the result while the optimizer generates the best plan from it. The request, one may argue, is lifted to a more qualitative level.

Now isn't that exactly what we are doing when we employ agents? We constrain the problem spaces and solution spaces, but we give the agent – and likewise the multiagent system – rules that construct for a given problem some solution within the constrained solution space. There may be many solutions, and the challenge for the agent or multiagent system is to find a good if not the best solution among them. In other words, we give the multiagent system a chance for *self-organization as the capability of the agents to optimize their collaboration with regard to a given qualitative task.*

11. References

[1] Aberer, K., Hauswirth, M.: *Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems*. Tutorial ICDE 2002. URL: http://lsirpeople.epfl.ch/hauswirth/papers/ICDE2002-Tutorial.pdf

[2] Androutsellis-Theotokis, S., Spinellis, D.: *A Survey of Peer-to-Peer Content Distribution Technologies*. ACM Comp. Surv. 36:4. 2004, 335-371

[3] Ballim, A., Wilks, Y.: *Artificial Believers: The Ascription of Belief*. Lawrence Erlbaum Ass., 1991

[4] Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: *Data Management for Peer-to-Peer Computing: A Vision*. Proc. Workshop. on the Web and Databases (WebDB'02) 2002, 89-94

[5] Bodorik, P., Jutla, D.: QoS Architecture for Caching in Middleware. SSGRR'2003

[6] Bratman, M.E.: *What is Intention?*. In P.R.Cohen, J.L.Morgan, M.E.Pollack (eds.): Intentions in Communications. MIT Press 1990, 15-32

[7] Bussmann, S., Jennings, N.R., Wooldridge, M.: Multiagent Systems for Manufacturing Control – A Design Methodology. Springer. 2004

[8] Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D.: *Context is Key*. Comm. ACM 48:3. 2005, 49-53

[9] Eugster, P.T., Guerraoui, R., Kermarrec, A.-M., Massoulié, L.: *From Epidemics to Distributed Computing*. IEEE Computing 37:5 (2004), 60-67

[10] Ferber, J.: Multi-agent Systems – An Introduction to Distributed Artificial Intelligence. Addison-Wesley. 1999

[11] Feynman, R.P., Leighton, R.B., Sands, M.: *The Feynman Lectures on Physics*. Addison-Wesley. 1965

[12] Foundation for Intelligent Physical Agents: SC00037J – FIPA Communicative Act Library Specification.

[13] Gal, A., Anaby-Tavor, A., Trombetta, A., Montesi, D.: *A framework or modelling and evaluating automatic semantic reconciliation*. The VLDB Journal 14:1 (2005), 50-67

[14] Gonzalez, M.C., Herrmann, H.J.: *Scaling of the propagation of epidemics in a system of mobile agents*. Physica A, 340 (2004), 741-748

[15] Gray, J., Reuter, A.: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann. 1993

[16] Guo, H., Larson, P.-A., Ramakrishnan, R., Goldstein, J.: *Relaxed Currency and Consistency: How to Say "Good Enough" in SQL*. Proc. ACM SIGMOD Int. Conf. 2004, 815-826

[17] Jamil, H.M.: *Belief Reasoning in MLS Deductive Databases*. Proc. ACM SIGMOD Int. Conf. 1999, 109-120

[18] Katsuno, H., Mendelzon, A.O.: On the Difference between Updating a Knowledge Base and Revising It. KR 1991, 387-394

[19] Lockemann, P.C., Dittrich, K.R.: *Architektur von Datenbanksystemen*. dpunkt.verlag. 2004

[20] Lockemann, P.C., Nimis, J.: *Flexibility Through Multiagent Systems: Solution or Illusion?* In P.v.Emde Boas, J.Pokorny, M.Bielikova, J.Stuller (eds.): SOFSEM 2004: Theory and Practice of Computer Science. Lect. Notes in Comp. Science 2392. Springer 2004, 41-56

[21] Moore, C., Newman, M.E.J.: *Epidemics and percolation in small-world networks*. Physical Review E 61:5 (2000), 5678-5682

[22] Ossowski, S.: *Co-ordination in Artificial Agent Societies*. Lect. Notes in Artif. Intelligence 1535. Springer. 1999

[23] Pearl, J.: *Belief Networks Revisited.* In: Artificial Intelligence in Perspective. MIT Press 1994, 49-52

[24] Pleisch, S., Schiper, A.: Approaches to Fault-Tolerant and Transactional Mobile Agent Execution – An Algorithmic View. ACM Computing Surveys 36:3 (2004), 219-262

[25] Rabinovich, M., Gehani, N., Kononov, A.: *Scalable Update Propagation in Epidemic Replicated Databases*. Proc. 5th Int. Conf. on Extending Database Technology 1996, 207-222

[26] Satoh, K., Yamamoto, K.: *Speculative Computation with Multi-Agent Belief Revision.* Proceedings 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems: Part 2. 2002, 897-904

[27] Schmidt, A., Gellersen, H.-W.: *Modell, Architektur und Plattform für Informationssysteme mit Kontextbezug*. Informatik Forsch. Entw. 16 (2001). 213-224

[28] Wang, J.: *A Survey of Web Caching Schemes for the Internet*. ACM Computer Communication Review 29:5 (1999), 36-46

[29] Weikum, G., Vossen, G.: *Transactional Information Systems*. Morgan Kaufmann. 2002

[30] Weiss, G. (ed.): Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence. The MIT Press. 1999

[31] Witte, R.: *Architektur von Fuzzy-Informationssystemen*. Dissertation Universität Karlsruhe. 2002. http://rene-witte.net

[32] Wooldridge, M.: *An Introduction to Multiagent Systems*. John Wiley & Sons. 2002