Enriching Active Databases with Agent Technology

Johan van den Akker
Arno Siebes

CWI
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
e-mail: {vdakker,arno}@cwi.nl

Abstract. Intelligent agents are software components with a largely autonomous behaviour, that are fitted out with a considerable degree of artificial intelligence. They are a promising paradigm to serve as a foundation for future computing environments in general, and information systems in particular. At the same time database research has seen the rise of active databases, database systems that add autonomous behaviour to a database. In this paper, we investigate the addition of notions from intelligent agents to an active database. We explain why active databases already implement weak agency, and look into the benefits stronger agency can bring to an active database. It turns out that these are mainly found in the increased flexibility facilitated by the reasoning abilities strong agency implies. For example, an agent can have multiple strategies to maintain a constraint instead of a one fixed strategy defined by triggers.

1 Introduction

The ongoing miniaturisation of computers is leading us to a world, where computers are omnipresent. This phenomenon has become widely known under the name *ubiquitous computing*. Although this development is still in its early stages, there is already some consensus on the software architecture for ubiquitous computing. This consensus considers *intelligent agents* the most promising paradigm for future information systems. In this paradigm, software consists of a number of entities collaborating towards a common goal, functioning autonomously with little intervention. Hence it is reasonable to expect that, in future computing environments, information systems will be based on a large number of cooperating agents.

At the same time, research in databases, the traditional foundation of an information system, has addressed the inclusion of additional modelling notions in databases. This has lead, among other things, to active databases, i.e. databases that include production rules. This allows databases to react autonomously to certain situations in the database.

Since databases are the current foundation and agents a future foundation of information systems, the question rises how information systems might evolve

from databases to agents. Since active databases are the first step in this evolution, we examine the role agent technology can play in an active database. The possibility of integrating agents in active databases was earlier mentioned in [4], which compared active databases and agent systems. Its main focus, however, was on the similarity of concepts in both areas, whereas we look into the opportunities agent technology offers active databases.

In this paper, we first identify the "level of agency" in a state-of-the-art active database model. Then, we present our view on the benefits the addition of further features of agent technology can bring to active databases.

2 Active databases

An active database [12] is a database that exhibits autonomous behaviour. That is, it executes actions without explicit intervention from a user. This behaviour is modelled by production rules, usually specified by an event-condition-action triple. If the event occurs and the condition is satisfied, the action of a rule is executed.

Originally, rules were introduced in databases in order to deal with constraints more flexibly, but rules have found much wider use. In fact, large parts of an application can often be specified by rules. The use of active databases has been classified by Kappel and Schrefl in [7]. They categorise applications of rules as follows:

1. maintaining static integrity constraints
2. maintaining derived data and materialised views
3. maintaining dynamic integrity constraints
4. database access authorisation
5. work step ordering
6. representing permissions to act
7. representing obligations to act

Of these seven, the first four are implementations of DBMS functionality. The fifth is relatively specific to certain applications, esp. workflow management applications. It can be regarded as a special case of the third application. The last two applications are forms of *business rules*. Business rules are a specification of company policy, or a description of the behaviour of a company. Simple examples are rules in an inventory application, that reorder an item if the stock falls below a certain level. More advanced business rules describes the competence of persons in the organisation. These business rules are the rules used to specify application functionality.

For the discussion in the remainder of the paper, we now introduce a state-of-the-art active database model developed at CWI, named DEGAS[1]. Here, we will

[1] DEGAS stands for Dynamic Entities Get Autonomous Status

only give an impression necessary for the discussion in this paper. For a full introduction the reader is referred to [1].

The core of DEGAS is formed by autonomous objects. Object autonomy means that an object is as independent as possible, both in its behaviour and its specification. Thus, an autonomous object encapsulates its complete behaviour. Its structure is given by attributes. The behaviour of a DEGAS object consists of *potential* behaviour, specified by methods and lifecycles, and *actual* behaviour, specified by rules. Methods specify the actions an object can execute, while an object's lifecycle specifies the possible execution sequences of its actions. Rules specify actual actions of the object in specified situations. They are given in the Event-Condition-Action format [6] commonly accepted in active databases.

An example of a DEGAS object is given in Figure 1. This object models an investor, who uses information from his newspaper to take decisions about his share portfolio. The syntax of attributes and methods is straightforward. The attributes specify the information stored by the investor, such as his name and birth data, the identity of his shares and his newspaper, and the currently perceived "reasonable" price for his shares. The methods specify the possible actions of the investor. He has methods to sell his shares and methods to deal with good and bad news about his shares.

Lifecycles are defined using process algebra [3] extended with guard conditions. In process algebraic terms, the process executed by the object must be a trace of the process specified by the lifecycle. For a further discussion of lifecycles in DEGAS, the reader is referred to [2]. In this example, the lifecycle regulates two things, sequence and access to methods. The first lifecycle specifies that a `tryToSell` action must always be followed by a `Sell` action or a `cancelSupply` action. The latter action withdraws an offer to sell shares that did not succeed. The two other lines in the lifecycle specification restrict access to the `goodNews` and `badNews` actions to the subscribed newspaper.

The event specification of a rule is also a process algebraic expression. The condition can be any condition on the object, while the action is a method call, either on the object itself, or to another object. After each method execution, the rule set of an object is checked against the object's history. If the specified event did occur and the rule's condition is satisfied, the action is executed. For each event, a variable is bound to the time the event occurred. This timestamp can be used in the condition to refer to historical values of the object's attributes. For example, the first rule of the `investor` object specifies that he tries to buy additional shares, if good news breaks twice within a week.

Interaction between objects takes place through message passing. In addition, objects can engage in relations. Relations allows objects to share and exchange information on a more permanent basis. More on the subject of relations between objects in DEGAS can be found in [1].

Object Investor
Attributes
name : string
birthday : time
birthplace : string
share : Oid
subscription : Oid
transactionPrice : real
Methods
tryToSell(company:string, number:integer, minPrice:real) = {
SupplyClass.initiateShareholder(company,number,minPrice)
}
Sell(buyer,price) = {
share.transferOwnership(buyer,price)
Supply.drop
}
cancelSupply = {
Supply.drop
}
goodNews(company : string) = {
transactionPrice = subscription.priceAdvice(company)
}
badNews(company : string) = {
transactionPrice = subscription.priceAdvice(company)
}
Lifecycles
(tryToSell;(Sell+cancelSupply))*
([sender==subscription]goodNews*)
([sender==subscription]badNews*)
Rules
On goodNews(company)(t_1);goodNews(company)(t_2)
if $t_2 - t_1 \leq 7$ days
do tryToBuy(company,transactionPrice)
On badNews(company)(t_1);badNews(company)(t_2)
if $(t_2 - t_1) \leq 7$ days && transactionPrice(t_2) $\leq$ transactionPrice(t_1)
do tryToSell(transactionPrice)
On goodNews(t_1);badNews(t_2)
if $t_2 - t_1 \leq 7$ days && transactionPrice(t_1) == max(transactionPrice, t_1, t_2)
do tryToSell(transactionPrice)
EndObject

Fig. 1. An investor modelled in DEGAS

3 Agency in Active Databases

In the previous section, we saw a state-of-the-art active database model based on objects with considerable autonomy. Hence, we could say that these objects are simple agents. In this section, we identify the level of agency offered by active databases in general, and DEGAS in particular.

Research on agents generally distinguishes *weak* and *strong* agency. A software system is said to have weak agency, if it possesses the following four properties [13]:

- autonomy
- social ability
- reactivity
- pro-activeness

Object *autonomy* is one of the base assumptions in DEGAS. Each DEGAS object is itself a process. Furthermore, its dependence on other objects is as small as possible through complete encapsulation and minimal assumptions about the behaviour of other objects. Hence, the criterion of autonomy is satisfied by DEGAS objects. *Social ability* means that agents interact with other agents in the system through an agent communication language. DEGAS objects pass messages to other objects and engage in relations with them. DEGAS objects *react* to their environment by answering messages. Furthermore, rules also specify reactions to situation that occur in the DEGAS database. *Pro-activeness* means that agents can take the initiative to achieve certain goals. Although goals are not explicit in a DEGAS object, active rules are instrumental to achieving a goal.

There is less consensus over stronger levels of agency. In general, strong agency is concerned with mentalistic notions. For the discussion in this paper, we take the four dimensions formulated by Shoham [9]:

- knowledge
- belief
- intention
- obligation

These notions are not explicitly supported by DEGAS objects. Although rules can be used to express obligations, they are not formulated as such. An obligation of an agent is specified by a goal that must be achieved. Instead an ECA rule is just an instruction to execute a certain action in a specified situation, although this action will be instrumental in fulfilling the obligation.

Likewise intention is only implicitly present, and as far as it is present, not arrived at by the DEGAS object itself. We could say, that a rule to maintain a database constraint expresses the intention to maintain that constraint. Intention and instrument to realise this intention, however, are fixed to each other. If intention

is an independent notion to an agent, it first derives its intention and then reasons about the actions to realise it.

Knowledge and belief are unknown notions in an active database. Although a lot of information is stored, the way to process these data is fixed by the methods and rules specified. Furthermore, a database usually lacks the ability to reason with and about the information it contains in a general way.

4 Extending the Level of Agency in an Active Database

In the previous section, we saw that DEGAS supports weak agency. In addition, limited representation of obligation and intention are present. In this section, we look at the potential results of extending the level of agency in an active database, taking DEGAS as a starting point. In particular, we consider the benefits of stronger agency for general database functionality.

Stronger agency is introduced in an active database by extending the capabilities of the objects in the database. While DEGAS currently is a database of autonomous objects, we would then have a database of agents. The agents in such a database[2] each manage a part of real world data, like an object represents a piece of data. This means that an agent contains a piece of data, and additionally possesses a number of goals it has to achieve or maintain. Furthermore, a data agent will have a number of obligations. In part, these will exist to facilitate DBMS functionality, e.g., an obligation to answer queries. Another part of the obligations will be to other agents, caused by relations between agents.

The key advantage of the promotion of autonomous objects to agents lies in the reasoning ability of agents. This allows a more abstract specification of the database, both in application modelling and in implementing database functionality. Triggers implement a tight binding between goals and means, so that an object has only one means achieve a specific goal. By formulating goals and means separately, more flexible solutions are possible for a lot of information systems functionality.

A prime example of the additional flexibility provided by separate goals and means is given by constraints. In Section 2, we mentioned that triggers were originally devised to deal with constraints more flexibly. The improvement triggers offered over existing mechanisms was, that we could use different strategies to maintain different constraints, instead of a single strategy for all constraints. Strong agency, by separating goals and means, gives us additional flexibility by allowing multiple strategies to maintain a single constraint. The agent can then infer which strategy is optimal in the current situation. In addition, the presence of general problem-solving strategies in the database obviates the need for specialised compilers, e.g., to produce rules to maintain constraints [5].

[2] or would we have to call it a data management society?

As an example consider the limit on the negative balance of a bank account. A limit of 2000 in the red is specified by the constraint:

$$balance \geq -2000$$

Suppose now that a requested payment violates this constraint. In this situation, we have a number of strategies to enforce this constraint, of which we mention four:

1. Refuse the payment
2. Transfer funds from a savings account
3. Sell some shares
4. Arrange a temporary loan

If we were to use triggers, we can specify only one action to enforce the limit on this account. With its enhanced reasoning capabilities, an agent can choose the best strategy to enforce this constraint, given its other goals, such as the quality of its relation with the customer, income in the near future etc.

Another advantage of stronger agency over triggers, is found in the problem of deciding termination of trigger sets. In general, we can decide termination only for very simple trigger languages [10]. Such languages, however, are too simple for most applications. Hence, we must find another way of dealing with this problem than deciding it in advance or imposing conservative pre-conditions on trigger sets. Stronger agency can help counter the problem of non-termination in two ways. First, the separation of multiple means to achieve the goal of a trigger, allows an agent to choose another means to achieve its goal, if the means originally chosen has undesirable side-effects. Second, not every possible execution of a non-terminating trigger set is non-terminating. This means, that the agents can cooperate to avoid the non-terminating execution sequence of their triggered actions.

As an example, consider the trigger activation graph given in Figure 2. In this graph the nodes are database states and the edges are trigger executions. For example, in database state c trigger $t3$ is triggered, whose execution brings the database in state e. The trigger set in this graph is non-terminating, since there exists a cycle of triggers $t2$ and $t1$ in this activation graph, which keeps the database shuttling between database states b and c. We can easily see, however, that there is also a terminating execution sequence for this trigger set, viz., the sequence $t1; t3; t2; t1$ that leads to the stable state f.

The advantages given above of strong agency over the weak agency in an active database also apply to dynamic database constraints. In DEGAS, the dynamic constraints of an autonomous object are given by the lifecycles. This lifecycle is fixed. Hence, a message that does not fit in the lifecycle is rejected outright. If an autonomous object had higher level reasoning facilities, it would be possible to negotiate a deal with the sending agent. For example, the receiving object might

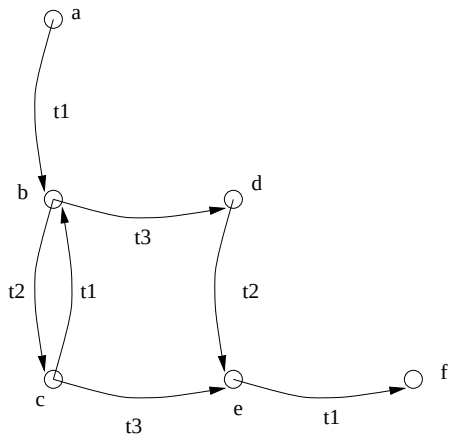

Fig. 2. A trigger activation graph

give an indication of the time when it will be able to execute the requested action. The sending object can then decide, depending on its other goals and obligations, whether it can wait or take another course to achieve its goal.

If negotiations between agents are to take place between different agents in the databases, we need, besides a language to conduct the negotiations in, a measure of the value of the different propositions being negotiated. This implies the use of a monetary model. The use of such a model is experimented in a somewhat different context by Stonebraker in the Mariposa system [11]. Here, the allocation of data storage and query processing in a distributed database is managed through a bidding system. The fixed bidding protocol in Mariposa, however, leads to undesirable effects in the allocation of data. In fact, it turns out that the richest site will end up with all the data, which means that it only gets richer by consequently under-bidding the other sites for query processing. Clearly, more sophisticated bidding and negotiation protocols are needed, which leaves the components (agents) in the database more room for manoeuvring to avoid undesired outcomes. For example, in the example of the richest site taking all data, the other sites might temporarily adopt a "dumping" strategy, i.e. working at a loss in order to gain back work and data.

The examples given of agents working out solutions for databases problems by negotiating between themselves, can only work under certain assumptions about their intentions. As research in game theory has shown [8], it is difficult to come up with strategies without undesirable outcomes, if not all players are cooperative. Therefore, agent research often assumes, as Shoham does [9], the veracity and benevolence of agents towards each other. This assumption cannot be held if agents in our information system must interact with agents owned by other people or organisations. Hence, agent research must find a solution for dealing with uncooperative, lying and malevolent agents in order to be able to

form the foundation of an information system, or an information infrastructure in general.

Further potential of agent technology in databases is found on the architectural side. If agent technology matures enough to form the basis of a database management system, this will implicate a large gain in flexibility of a DBMS. The different components of a DBMS, such as query optimiser, storage manager, etc., can each be an agent with its own goals and strategies. Besides the increased flexibility of the individual components, this also means increased freedom for a systems designer to choose the components constituting his DBMS.

5 Conclusions

In this paper, we discussed the autonomous behaviour that active databases add to traditional databases. We saw that DEGAS, a state-of-the-art active database model, implements weak agency, since DEGAS objects interact with each other, react to their environment, and autonomously pursue their defined goals. These functions, however, are limited by their fixed, pre-programmed character.

The extension of autonomous DEGAS objects to stronger agents with general purpose reasoning abilities greatly increases the adaptability and flexibility of an active database system. The improvements originate in the ability to adapt strategies to the actual situation, and the ability of agents to cooperate with each other. Hence, the incorporation of agent technology in databases opens up new perspective in tackling long-standing database issues. Increased coupling and inter-operation of information systems, however, also poses some new challenges for agent technology in order to deal with lying or malicious agents from outside.

References

1. Johan van den Akker and Arno Siebes. DEGAS: Capturing dynamics in objects. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Advanced Informations Systems Engineering - Proc. of CAiSE'96*, pages 82-98, Heraklion, Crete, Greece, May 1996. Springer. LNCS 1080.
2. Johan van den Akker and Arno Siebes. Object histories as a foundation for an active OODB. In R. Wagner and H. Thoma, editors, *Proceedings of the 7th International Workshop on Database and Expert Systems Applications (DEXA'96)*, pages 2-8, Zürich, Switzerland, 1996. IEEE Computer Society.
3. J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1990.
4. James Bailey et al. Active databases and agent systems - a comparison. In Timos Sellis, editor, *Proc. of the 2nd Intl. Workshop on Rules in Databases (RIDS'95)*, pages 342-356, Athens, Greece, 1995. Springer. LNCS 985.
5. Stefano Ceri and Jennifer Widom. Deriving production rules for constraint maintenance. In D. MacLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 566-577, 1990.

6. U. Dayal et al. The HiPAC project: Combining active databases and timing constraints. *SIGMOD Record*, 17(1):51–70, March 1988.
7. Gerti Kappel and Michael Schrefl. Modeling object behavior: To use methods or rules or both? In R. Wagner and H. Thoma, editors, *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA'96)*, pages 584-602. Springer, 1996. LNCS 1134.
8. Jeffrey S. Rosenschein and Gilad Zlotkin. Designing conventions for automated negotiation. *AI Magazine*, 15(3):29-46, 1994.
9. Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51-92, 1993.
10. A.P.J.M. Siebes, J.F.P. van den Akker, and M.H. van der Voort. (un)decidability results for trigger design theories. Technical Report CS-R9556, CWI, Amsterdam, The Netherlands, 1995. Available through WWW (`http://www.cwi.nl/~vdakker/`).
11. Michael Stonebraker et al. Mariposa: A wide-area distributed database system. *VLDB Journal*, 5(1):48-63, 1996.
12. Jennifer Widom and Stefano Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Francisco, CA, USA, 1995.
13. Michael Wooldridge and Nicholas R. Jenning. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115-152, 1995.

This article was processed using the LaTeX macro package with LLNCS style