Enterprise integration with agent-based engineering data management

M. Kollingbaum
Research Associate, Manufacturing Engineering Group,
University of Cambridge, Mill Lane, Cambridge, UK, CB2 1RX
Telephone: +44 1223 338077; Fax: +44 1223 338076
E-mail: mjk27@eng.cam.ac.uk

H. Stadlbauer
max.mobil. Dept. max.service.
Kelsenstr. 5-7, A-1030 Vienna, Austria
Telephone.: +43 1 79585 6498, Fax: +43 1 79585 6527
E-mail: harald.stadlbauer@maxmobil.at

Abstract

Information is the key factor in industry nowadays. Especially Computer Integrated Manufacturing intends to use information technology to integrate all parts of an enterprise. Product development often involves large teams of design specialists, produces a large amount of data and should consume as less time as possible. Therefore, CIM forces the application of information management technology in order to guarantee that the right data are present at the right location in the right time. Agents and multi-agent systems are an upcoming technology that can help to establish distributed intelligent applications for industrial environments. Agent-based applications seem well suited to the distributed character of an enterprise-wide information management. However, industry requires a data management that is robust and reliable. In this paper we want to show how database technology can be integrated with agent technology to provide a persistent, robust, and reliable information management within multi-agent systems.

Keywords
Information management, CIM, multi-agent system, DBMS

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35357-9_22

A. B. Baskin et al. (eds.), *Cooperative Knowledge Processing for Engineering Design*

1 INTRODUCTION

Collaborative product development is a situation where different designers are working together exchanging experience and knowledge for fulfilling their design task. Supporting distributed design teams with information management means to maintain large amounts of data of different engineering disciplines. All these data are strongly interrelated and form a «web of information.» The process of product development is rather evolutionary and characterised by iterations of design activities that initialise, transform and extend such a «web of (product) information.» Many restrictions like special requirements of specific production processes, material constraints, and technological aspects must be taken into account. Designers of different engineering disciplines and with specific distinct experience and knowledge are involved in this process of product development. They build a kind of «web of shared understanding» by discussing product details and making design decisions.

Agent technology is a new and attractive way of implementing intelligent and distributed information systems. The concept of co-operating agents seems well suited to the implementation of information systems that can automatically reason and keep track of the various restrictions and technological aspects of a design process. It seems as if this «web of shared understanding» can also be manifested in software using agent technology. However, for the industrial use of multi-agent systems a secure and reliable data management is very important.

The research project «IDEE Intelligent Data Engineering environment» is an effort to combine concepts of expert system shells and database system technology to enable the modelling and implementation of multi-agent systems. With IDEE a persistent and reliable data management can be provided for this «web of shared understanding.» IDEE demonstrates how a database system in combination with an expert system shell can build an environment that allows the development of industrial applications in an agent-oriented style. Currently we are able to implement agent-based applications in which the agents have reasoning capabilities, can be distributed within a computer network, and have access to an active realtime database system (Purimetla, 1993, Ramamritham, 1993, Chakravarthy, 1989, Dittrich, 1995). The database system is used by agents for communication and exchanging data and provides a network-wide event notification mechanism.

1.1 Intelligent data engineering environment

IDEE is based on the concepts of intelligent agents (Wooldridge, 1995) and supports the implementation of multi-agent systems.

As agents are «intelligent autonomous software entities» that «co-operate» with each other, IDEE must support the concurrent execution of a set of agents and enable co-operation. Co-operation means that agents work together to solve a specific problem. Therefore communication is essential for multi-agent systems. Agents communicate by exchanging «speech acts» (Austin, 1962, Cohen, 1988, Searle, 1969, Searle, 1985). KQML (Knowledge Query and Manipulation

Language) (Finin, 1994) is a protocol for agent communication and is based on the concept of speech acts. Agents must be able to «understand» speech acts. Understanding between agents can be reached by using ontologies, which represent a conceptualisation of a specific problem domain (Gruber, 1991).

IDEE must contain specific features and sub-components to enable the execution and communication of intelligent agents. IDEE is the result of the combination of existing software components and was implemented by integrating the expert system shell CLIPS with a database system.

2 ARCHITECTURE

Important goals for the implementation of IDEE were minimisation of implementation effort and ease-of-use. The reduction of implementation effort was reached by using existing software components. The expert system shell CLIPS was integrated with a real-time in-memory database system. With this architecture we can implement multi-agent systems consisting of a set of CLIPS-processes representing agents, which access concurrently the database system. CLIPS was extended by a database interface, which provides persistence for CLIPS-objects. In case of IDEE a CLIPS-object is persistent, if a corresponding table is defined in the underlying database system and if there is a permanent synchronisation on any change between these two storage types. The «easy to use» requirement is fulfilled by the fact that many details like persistence, distribution, synchronisation are hidden from the implementing engineer. To implement a multi-agent system, an engineer needs programming experience only in CLIPS. The main task to build up such agent-based applications is to implement a set of rules in CLIPS.

2.1 Overview

When the multi-agent system is executed, each agent is a separate CLIPS process, as is shown in Figure 1. Agents use the database system for making CLIPS objects persistent. Ontologies used by agents can be stored in a database. The database system is also used as a communication mechanism.

A database system is basically a data storage facility. But if it allows concurrent access to data and provides transaction mechanisms to regulate such concurrent activities, then database structures can also be used for exchanging data. Two applications that want to exchange information (e.g. two agents exchanging speech acts), have to write to and read from a specifically defined database structure (a table in a relational database system (Codd, 1970, Korth, 1986) or an object in an object oriented database system (Dittrich, 1991, Kemper, 1994)).

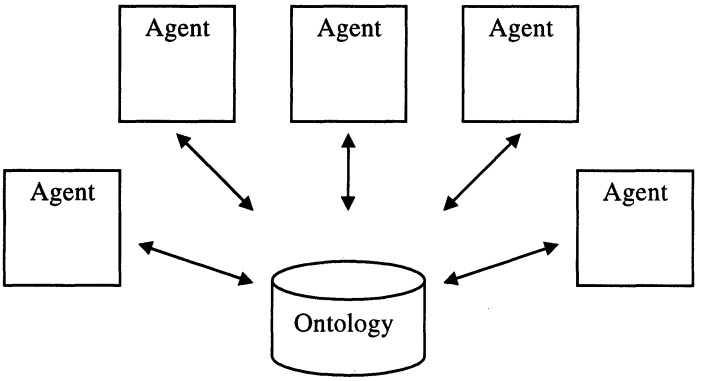

Figure 1: Agents communicate over a database system.

The database system we have chosen to implement IDEE has following features:

- It is a in-memory relational database system;
- it has a synchronisation mechanism that allows to mirror data into a commercial database system like ORACLE;
- it has a replication mechanism that allows the network-wide distribution of database structures (tables);
- it has a trigger mechanism that allows network-wide notifications when data manipulations occur.

We use these features to enable a network-wide implementation of multi-agent systems. The «in-memory» feature meets speed requirements. The replication helps to distribute ontologies network-wide. It also enables network-wide communication by replicating database tables that are specifically used for communication. The trigger mechanism is necessary to co-ordinate communication and to notify an agent, which is the addressee of a speech act, to «receive» that speech act (that means to read information from the table that is explicitly used for communication) and act on it.

Figure 2 provides an overview of IDEE. CLIPS agents communicate over replicated database tables, have access to the same ontological information on all network nodes (which is also kept in replicated database tables), and can exchange data with legacy (already existing) applications.

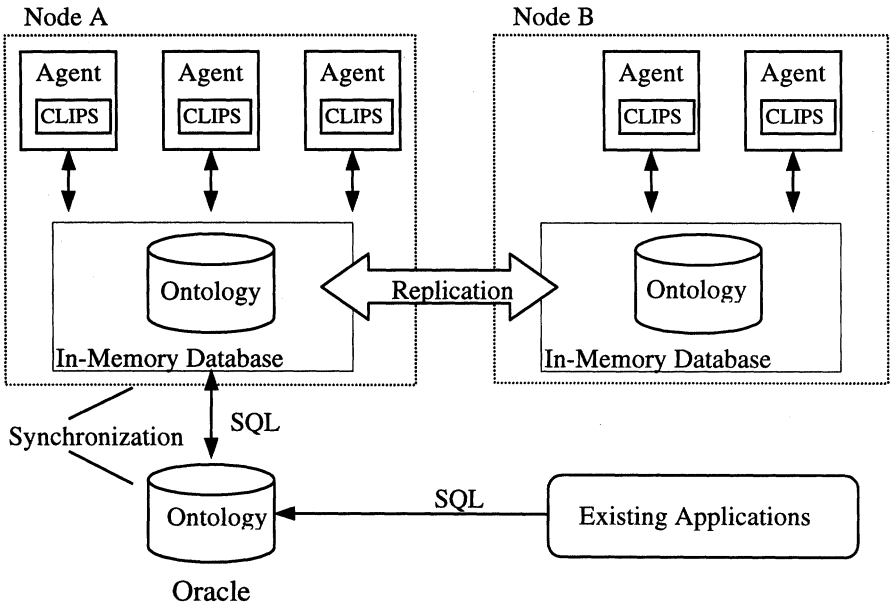

Figure 2: Agents communicating over replicated data, with connectivity to existing applications.

Persistence is a very important concept within the IDEE-approach. Persistence allows an IDEE-agent to permanently store ontological information, and it enables the communication of these agents. First we want to show how agent communication was implemented for IDEE using the database system as a communication facility, then we show how CLIPS-objects can be made persistent for IDEE, how persistence and communication are interrelated, and how both features enable the concept of distributed persistent objects for IDEE.

2.2 Communication

In IDEE the database system is the main facility for exchanging messages (speech acts) between agents. All agents of a multi-agent system have access to the same database and tables within this database, therefore tables can be used for communication. Table entries represent messages. A communication act can be done with following steps (shown in Figure 3):

- one agent, the sender, **writes** a message-table-entry into a specific table
- the receiving agent **is notified**, that a message is pending
- the receiver **reads** the pending message from the database and **deletes** it

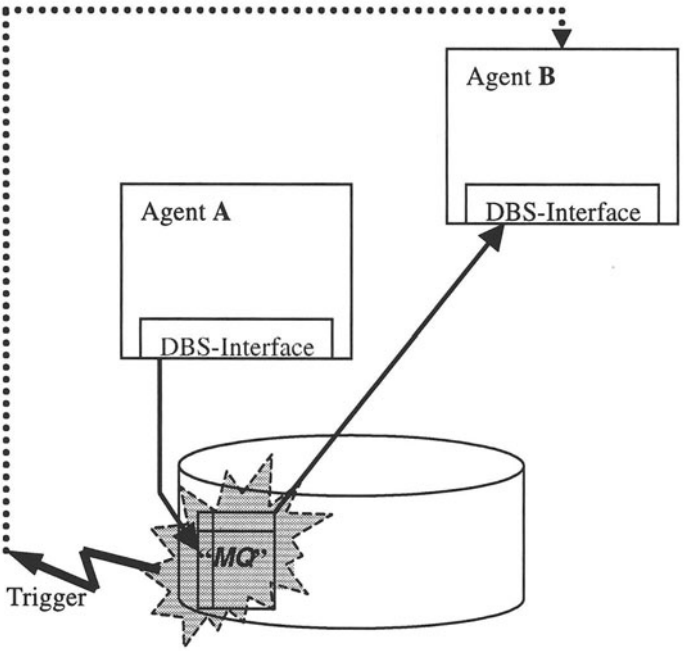

Figure 3: Agent communication.

To implement a notification mechanism for pending messages we use the database triggers provided by the database system we use in IDEE. The following design decisions were made:

- One specific table in the database is defined to act as a means of communication, we call it «message queue» (in Figure 4 «***MQ***»), as it will contain table entries, which represent messages (or speech acts) sent to a specific agent
- These messages specify the addressed agent and the action this agent is intended to do
- This specific «message queue»-table is the only one which is bound to a trigger routine; a write operation on this table activates the trigger, which issues a (network-wide) notification to wake up agents to read the «message queue»
- The «message queue»-table is maintained by the receiving agent; receiving a message means to read the message and subsequently delete the message; as long as the addressed agent does not perform its receiving activities (read and delete) the messages remain in the «message queue»

This communication concept of IDEE, which provides persistent messages, makes the communication of IDEE-agents robust and reliable. Messages cannot get lost,

if an agent intended to receive a message is temporarily unreachable (because of network failures or system crash).

For this communication schema CLIPS was extended with a database interface, which allows basic database operations like inserting new tuples into a database table (the «message») or reading them from the database system. This database interface is also necessary to implement persistence for CLIPS-objects.

2.3 Persistence

To provide agents within IDEE with persistent data handling requires extending CLIPS itself with this ability. To allow the object system of CLIPS to make specific classes persistent we have to extend CLIPS with a database system interface (which is also used for the communication schema of IDEE – therefore a message sent by an agent is itself a persistent CLIPS object).

2.3.1 Making objects persistent in CLIPS

In CLIPS we have an object management system that allows the specification of class hierarchies and provides inheritance and message handling. The database system we want to use within IDEE allows us to specify tables for storing data. The structuring element of CLIPS classes are «slots» (slot-values express the state of a specific object); the structuring element of the tables of our database system are «attributes» (expressing the current state of a specific table entry). The mapping between slots and attributes is straightforward. But a simple mapping does not provide us with persistence.

For IDEE we want to define persistence as follows:

- A CLIPS-class is persistent, if (1.) there is a database table with equal name and equal structure and (2.) there is a synchronisation between these two representations of a data object on each manipulation of both the CLIPS-object or the table entry in the database.

The aspect of synchronising both representations of a data object, the CLIPS object and its «shadow» in the database, is very important. Synchronisation from the CLIPS's point of view means that if there is a manipulative operation done on an object (new instantiation, change of a slot-value, deletion), corresponding operations have to be done on the database.

2.3.2 The database interface in CLIPS

To allow such operations, we extended CLIPS with a database interface that provides the basic operations INSERT, UPDATE, DELETE and SELECT. These operations take a CLIPS object and save it into (retrieve it from, delete it in) the database. Therefore there must be a class definition in CLIPS and a corresponding table definition in the database schema. To meet the persistence-criteria we specified, these operations must be used in the following way:

- After an object is instantiated within CLIPS, this new object (which has to be persistent) must be inserted into the database by using the INSERT-operation
- If a «persistent» CLIPS-object is deleted in CLIPS it must also be deleted in the database

- If the corresponding database table was modified, the CLIPS-object must be resynchronised by using SELECT to get the new values from the database
- Any modification of a CLIPS-object must be accompanied by an UPDATE-operation on the database

A CLIPS programmer implementing rule bases for agents can decide to explicitly do these database operations or use the message handling system of CLIPS. This message handling system allows a comfortable way of automatically activating activities in case of manipulations of CLIPS-objects. The message handling mechanism of CLIPS can be compared to trigger mechanisms of modern database systems. Message handlers can be provided which completely hide all database operations.

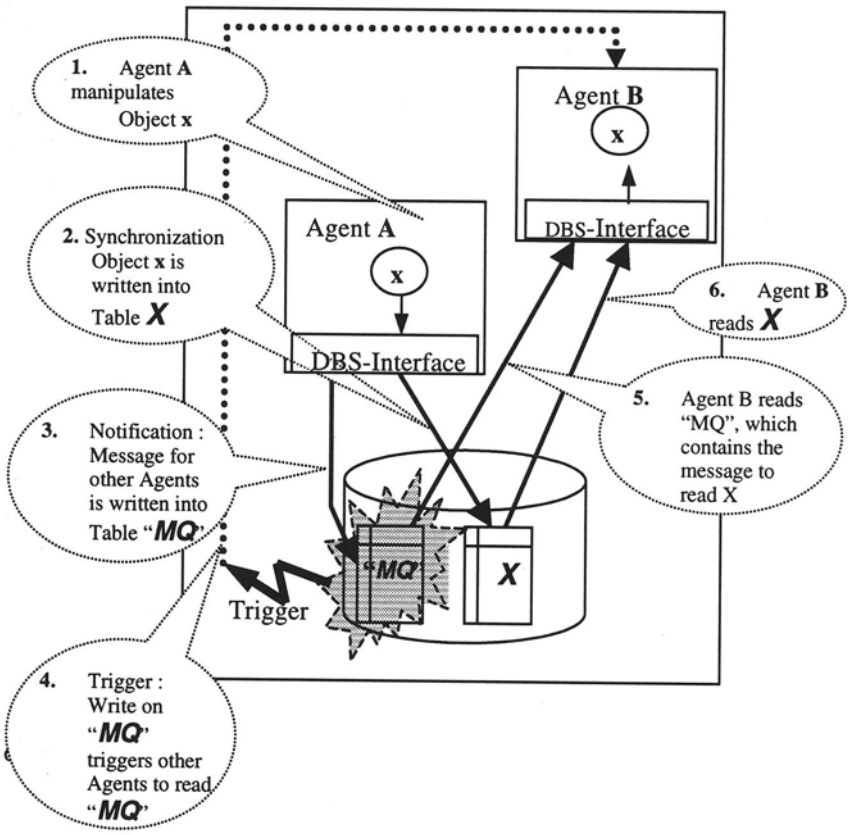

Figure 4: Communication and distribution of objects.

2.3.3 Manipulating the database

If there is a manipulative operation done on a database table there must be a resynchronisation of the corresponding CLIPS-object of an agent. The IDEE-agent must be notified to do corresponding operations: reading a changed version of a data object from the database using a SELECT-operation, or instantiating a new CLIPS-object, or deleting a CLIPS-object.

We use our concept of agent communication over the database system to send an agent a notification.

In Figure 4 two agents interact because of a manipulation of object **x**. The communication and notification is done with the table ***MQ***. Agent **A** manipulates object **x** (step 1), which is followed by a subsequent write operation into the database table ***X*** (step 2). Other agents within the multi-agent system must be informed that table ***X*** is updated. Therefore, the database interface of Agent **A** automatically writes a subsequent «notification message» into ***MQ*** (step 3). ***MQ*** stores table entries, which contain at least following information: sending agent, receiving agent and desired action. Therefore ***MQ*** will contain following «message»: «agent **A**» (the sending agent), «agent **B**» (the receiving agent) and «read **x** from ***X***» (the desired action). This write-operation activates the trigger bound to ***MQ*** (step 4). Agent **B** is activated because of the trigger and reads ***MQ*** to get the information what to do (step 5). ***MQ*** contains a message for Agent **B** to read ***X***. At last agent **B** reads ***X*** and updates its own version of **x** (step 6). After this synchronisation step the CLIPS-part of agent **B** is started which allows agent **B** to reason about the changed state of **x**.

Using this notification mechanism, persistent objects can be easily distributed within a multi-agent system. Any number of agents of a multi-agent system can operate on the same persistent data object. Each agent acts on its own private CLIPS object, but all agents have access to the same database and synchronise the object's state with the content of the same table in the database. Therefore in IDEE persistent objects are distributed too.

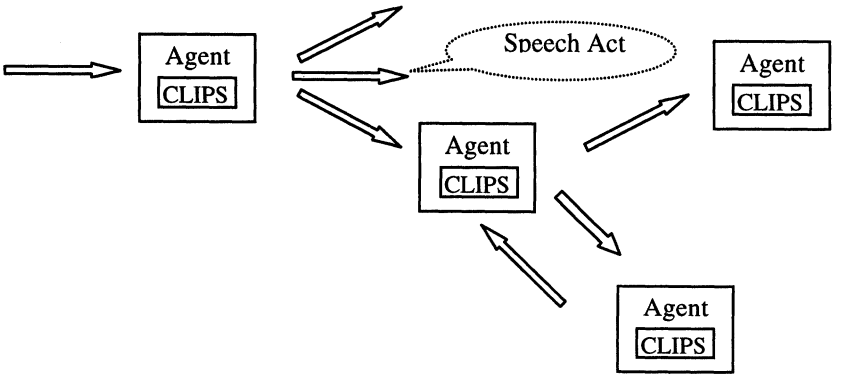

Figure 5: Execution of a multi-agent system.

If the database system allows distribution within a network, persistent objects are accessible network-wide.

2.4 Execution of a multi-agent system

IDEE must co-ordinate a complete multi-agent system in its execution. The execution is determined by speech acts agents are sending each other (Figure 5).

Speech acts in case of IDEE are messages, which contain a specific action the agent has to perform. Actions can be e.g. «quit yourself», «reset your CLIPS data structures» or «read entry **x** of database table ***X***».

For each agent a set of rules (the rule-base) must be implemented in CLIPS. IF an agent performs an action, which is intended by a receiving speech act and which is a manipulative operation on a CLIPS data structure (facts, objects), rules of the agent's rule-base can become activated. To execute or «fire» these activated rules, the inference engine of CLIPS must be started.

Therefore an agent always has to perform two activities after receiving a speech act:

1• Perform the action, intended by the speech act, in most cases the agent has to read the new state of a persistent CLIPS-object from the database;

2• start the inference engine of CLIPS to reason about manipulated persistent CLIPS-objects.

When the inference engine is started, activated rules are fired. The reasoning of an agent can result in new speech acts sent to other agents of the multi-agent system. Each of these affected agents has to perform these two steps of execution.

3 CONCLUSION

In this paper we have presented IDEE, an engineering environment for the development and execution of multi-agent systems. IDEE is the integration of expert and database system technology. We have shown how such an environment can be implemented at low costs using database system features like triggers or the ability to distribute databases. IDEE-agents have reasoning abilities and access to a real-time distributed database system. Each agent can therefore manage persistent data objects and reason about their actual state. The execution of a multi-agent system in IDEE is based on the concept of speech acts. Agents send speech acts to other agents, intending these agents to perform specific data manipulation actions. The database system is used as a communication facility and allows the distribution of data objects within a multi-agent system. Using the database system for this purpose provides the opportunity of persistence in communication. It helps to make a multi-agent system more robust in case of system crashes. And it reduces the implementation effort drastically, only a small database interface for agents has to be implemented. The power of the complete environment lies in the interaction of its sub-components. The expert system shell helps to implement

intelligent agents with reasoning abilities. The database system covers all required features like persistence, concurrency, recovery, distribution, real-time data management and triggers. It makes IDEE reliable and robust, which are main requirements for industrial applications and software systems. The implementation of multi-agent systems is as simple as implementing a single expert system. A knowledge engineer can concentrate on the local functionality of one agent, using the well known architecture and language constructs of the expert system shell CLIPS.

4 REFERENCES

Austin,J.L.(1962): *How to do things with words.* Oxford University Press, NY, 1962.

Chakravarthy,S. (1989): *Rule Management and Evaluation: An Active DBMS Perspective,* SIGMOD RECORD, Vol.18, No.3, September 1989

Codd,E.F. (1970): *A Relational Model of Data for Large Shared Data Banks*, Communication of the ACM 13.

Cohen,P.R.,Perrault, C.R. (1988): *Elements of a plan-based theory of speech acts.* Alan H. Bond, Les Gasser (Ed.), Readings in Distributed Artificial Intelligence, pp. 169-186, Morgan Kaufman Publishers, San Mateo, CA.

Dittrich,K.R. (1991): *Object-Oriented Database Systems: The Notation and the Issues*, In: Dittrich,K.R.,Dayal,U.,Buchmann,A.p.:»On Object-Oriented Database Systems», Springer Verlag.

Dittrich,K.R., Gatzin,S., Geppert,A. (1995): *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*, in: Sellis,T. (Ed.): «Rules in Database Systems, Second International Workshop, RIDS'95, Glyfada, Athens, Greece, September 25-27, 1995», Lecture Notes in Computer Science, Springer Verlag.

Finin, T.,Fritzson, R.,McKay, D., McEntire, R.(1994): *KQML as an Agent Communication Language.* Proceedings of the Third International Conference on Information and Knowledge management (CIKM'94), ACM Press, November 1994.

Gruber,T.R. (1991): The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases, J.A.Allen, R.Fikes, E.Sandewall (Ed.), «Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR&R-91)», Morgan Kaufman Publishers, San Mateo, CA.

Kemper,A., Moerkotte,G. (1994): *Object-Oriented Database Management: Appications in Engineering and Computer Science*, Prentice-Hall.

Korth,H.F, Silberschatz A. (1986): *Database System Concepts*, McGraw-Hill, NY.

Purimetla,B., Sivasankaran,R., Stankovic,J. (1993): *A Study of Distributed Real-Time Active Database Applications*, IEEE Workshop on Parallel and Distributed Realtime Systems, April 1993

Ramamritham,K. (1993): *Real-Time Databases*, Journal of Distributed and Parallel Databases, Vol 1, No 2.

Searle, J.R. (1969): *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press: Cambridge, UK.

Searle,J., Vanderveken,D. (1985): *Foundations of illocutionary logic*, Cambridge University Press, Cambridge, UK.

Wooldridge,M., Jennings,N.R. (1995): *Agent Theories, Architecture and Languages: A Survey*, Lecture Notes in Artificial Intelligence, Vol. 890, Springer Verlag

5 BIOGRAPHIES

Martin **Kollingbaum** studied Computer Science at the Technical University of Vienna. In 1987 he became research assistant at the Institute of Flexible Automation of the TU Vienna and was involved in the development of a non-standard database system for telecommunication facilities. 1990 he finished his study with the degree of a Dipl.Ing. Between 1990 and 1994 he worked in industry as a software engineer. 1994 he returned to the Institute of Flexible Automation and was involved in a research project on agent technology for industrial environments. 1997 he became a research scholar at the University of South Carolina. He is currently a research associate at the University of Cambridge, UK.

Harald **Stadlbauer** was born in 1963 in Austria. After graduating in computer science at the University of Linz, Prof. Bruno Buchberger, he joined a research and development team for Robotics simulation and programming at the Austrian company VOEST. In 1988, he joined the Institute of Flexible Automation, Technical University of Vienna, in the area of production simulation, expert systems for production planning, CAD and engineering design. In 1992, he finished his PhD thesis in the area of theoretical frameworks for automating engineering design and built up a profit centre in research with industry. In 1995 he joined the environmental centre Europe of SONY in Stuttgart. In 1996, he then left to return to Austria for CAP GEMINI in quality management. Since August 1997, he is a BPR specialist at max.mobil., an Austrian telecom provider.