Integration of Productmodel Databases into Multi-Agent Systems

Jochen Bilek, University of Bochum, Germany (bilek@inf.bi.rub.de)
Mirko Theiß, University of Darmstadt, Germany (theiss@iib.tu-darmstadt.de)
Dietrich Hartmann, University of Bochum, Germany (hartus@inf.bi.rub.de)
Udo Meissner, University of Darmstadt, Germany (sekretariat@iib.tu-darmstadt.de)
Uwe Rüppel, University of Darmstadt, Germany (rueppel@iib.tu-darmstadt.de)

Abstract
This paper deals with two different agent-based approaches aimed at the incorporation of complex design information into multi-agent planning systems. The first system facilitates collaborative structural design processes, the second one supports fire engineering in buildings. Both approaches are part of two different research projects that belong to the DFG[1] priority program 1103 entitled "Network-based Co-operative Planning Processes in Structural Engineering" (DFG 2000). The two approaches provide similar database wrapper agents to integrate relevant design information into two multi-agent systems: Database wrapper agents make the relevant product model data usable for further agents in the multi-agent system, independent on their physical location. Thus, database wrapper agents act as an interface between multi-agent system and heterogeneous database systems. The communication between the database wrapper agents and other requesting agents presumes a common vocabulary: a specific database ontology that maps database related message contents into database objects. Hereby, the software-wrapping technology enables the various design experts to plug in existing database systems and data resources into a specific multi-agent system easily. As a consequence, dynamic changes in the design information of large collaborative engineering projects are adequately supported. The flexible architecture of the database wrapper agent concept is demonstrated by the integration of an XML and a relational database system.

1 Introduction
Typical planning processes in building design are based on the distribution of work. The design and erection of buildings is characterized by the co-operation of experts of many disciplines; every expert produces and processes project-specific information during a project work. Through this, the team work based design of buildings permanently places high demands on co-operation and coordination such that a high quality is guaranteed and design deficiencies are prevented.

The increasing popularity of digital product models and digital communication over the internet reveals novel concepts and approaches to support distributed engineering dynamically. In this area, agent technologies are a promising concept for the support of distributed product models affiliated with collaborative team work and information processing. Hereby, a **multi-agent system (MAS)** can be considered as a loosly-coupled network of autonomous and interacting problem solvers (software agents), that collaborativeley work together to solve some problems being above their individual capabilities. The usefulness of multi-agent systems in the field of co-operative engineering is presented by two diverse engineering application examples in section 2. Both examples require the computer-based support of complex and distributed planning processes. In both cases, the analysis of the corresponding domain has led to two specific multi-agent systems that are modeled and implemented within the scope of two different research project works.

Within both projects we came to the conclusion that the integration of the project related planning information into the particular multi-agent systems is central for meeting the imprecated design objectives. The collaborative provision of information, thereby, requires the integration of legacy applications that are usually employed to process and store data. Typically, storage and transaction issues arising from complex information handling are transfered to standard database systems that have proved very effective and efficient in the past. Consequently,

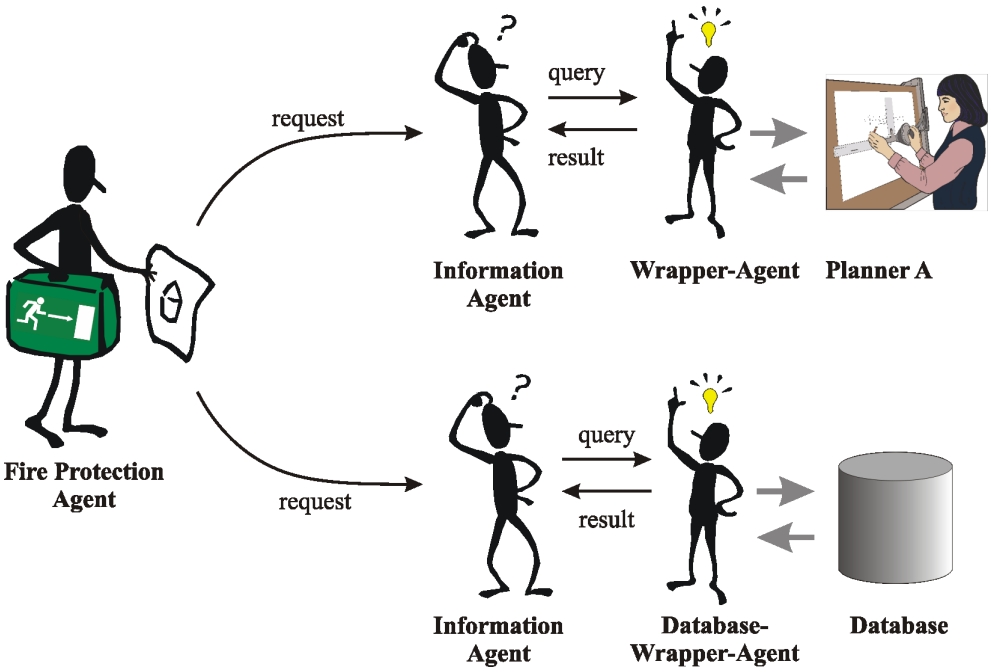

Figure 1: Processing fire protection information in the multi-agent system Madita

the integration of such database systems into the project specific multi-agent systems has turned out to be essential. Therefore, we have modeled and developed special **database wrapper agents** as elucidated in section 4. In the two proposed multi-agent systems database wrapper agents encapsulate legacy database systems and, thus, act as interfaces to all co-operating agents in the respective MAS. In fact, most database systems nowadays, used nowadays, are based on SQL[2]. For this reason, we accomplished the wrapping of the very common relational database type within our project works. Furthermore, information described in XML[3] are becoming more and more significant. Hence, we additionally developed a concept to encapsulate XML database systems based on XPath (W3C 1999). The database integration concept is modeled very flexible and general such that additional database types (e.g. object-orientated database systems) can be encapsulated easily as depicted in section 4.

Inside both MAS, the exchange of stored information expects that, on request, agents interact with the database wrapper agent. This is achieved by the exchange oft speech-act based messages. The flow of messages is thereby definded in **interaction protocols**. The exchange of messages is usually combined with domain-dependent **ontologies** in which a common vocabulary on particular areas of expertise is determined. Agent communication and ontology development required by the database wrapper agent is pointed out in section 5.

Finally, the mapping of the ontology-based message contents to the database schema has to be established. Depending on the extracted message contents, the database wrapper agent performs specific actions on the encapsulated database system and returns the queried information, subsequently. This mechanism is illustrated in section 6.

2 Application examples

In this paper we examined the application of database wrapper agents in the context of two research projects of the DFG priority program “Network-based co-operative planning processes in structural engineering“ (DFG 2000),specifically, the project “Collaborative structural design based on multi-agent systems and adaptive association networks“ (Bilek and Hartmann 2002), carried out at the University of Bochum (Germany), and the project “Agent-based model

compound for cooperative building design“ (Meissner and Rueppel 2000), conducted at the Technical University of Darmstadt (Germany). The Bochum project deals with the agent-based support of distributed structural design processes using a pedestrian arch bridge as a reference structure, whereas the Darmstadt project enables the agent-based check-up of fire engineering tasks in building design for every participating expert. The implementation of software agents in both projects deploys the multi-agent platform JADE[4] (Bellifemine 2004), a Java-based and FIPA[5]-conform (FIPA 2004) agent system architecture. The mutual use of JADE in both projects has emerged as very effective: both projects benefit from JADE specific knowledge exchange and discussions about JADE implementation and agent design issues. In this section, we briefly depict examples of use and discuss basic concepts of both project works. Furthermore, we elucidate how the agent-based integration of project-relevant data is achieved in both research projects by using database wrapper agents.

Structural fire protection and check-up requires a close collaboration of all participating engineers during the design process of a building to avoid personal and property damages. The central idea of the Darmstadt project therefore is to integrate all relevant design participants in a distributed system, i.e. the Multi-Agent-System for Distributed Fire Engineering Tasks *Madita*. Structural fire protection, as a part of fire engineering, is excellently feasible for the demonstration of an agent based planning system. Almost every activity during the design of a building affects structural fire protection issues such as fire protection guidelines for particular parts of the building or even the entire building. Moreover, all planning activities must fulfil the defined fire protection objectives. Considering the case that just one single component of the building has been designed incorrectly or erroneous, in the case of fire, the overall fire safety can no longer be guaranteed. As a consequence, however, it is reasonable to integrate all relevant individual planning activities of every participating designer into a distributed system such that design deficiencies can be detected as early as possible.

The central component within Madita is the mobile fire protection agent. The fire protection agent can be asked from every designer for checking his detailed planning with regard to structural fire protection issues (fig.1). To meet the fire protection requirements the rule-based conclusion system JESS (Java Expert System Shell) (Friedmann-Hill 2004) is integrated in the fire protection agent. The conclusion system checks the predetermined requirements by processing the defined fire protection rules associated with the relevant structural facts that can be derived from the building model directly. In advance to this activities the fire proctection agent needs to know the fire protection rules and facts that are maintained by the various participating designers. For the retrieval of this essential information within the distributed project environment additional, specialized mobile agents – the *information agents* – are provided. Thus the fire protection agent does not have to migrate to the platforms of the planning participants directly. Instead, it charges the information agents with retrieving the necessary information. The mobile information agents interact with the fire protection agent by using a common language and vocabulary. By that, they are able to pass the fire information request of the protection agent to the project participants correctly. Vice versa, after the reception of the requested product model information or fire protection rules the instructed information agents hand over the results to the fire protection agent. After having carried out the retrieval of rules and facts, the information agents migrate to the platforms of the project participants and attempt to determine the requested information there. Consequently, the system does not have a central database; neither there is any definition of a global modeling schema or database diagram. Therefore, a mapping of the information requests to the local database schema is necessary. This special task is carried out by the database wrapper agent, described in this contribution. A typical request for particular planning information is illustrated in the sequence diagram of figure 2.

Similar to collaborative fire engineering, structural design tasks are customarily handled by temporary teams of design experts. A team work is urgently required due to the high degree of

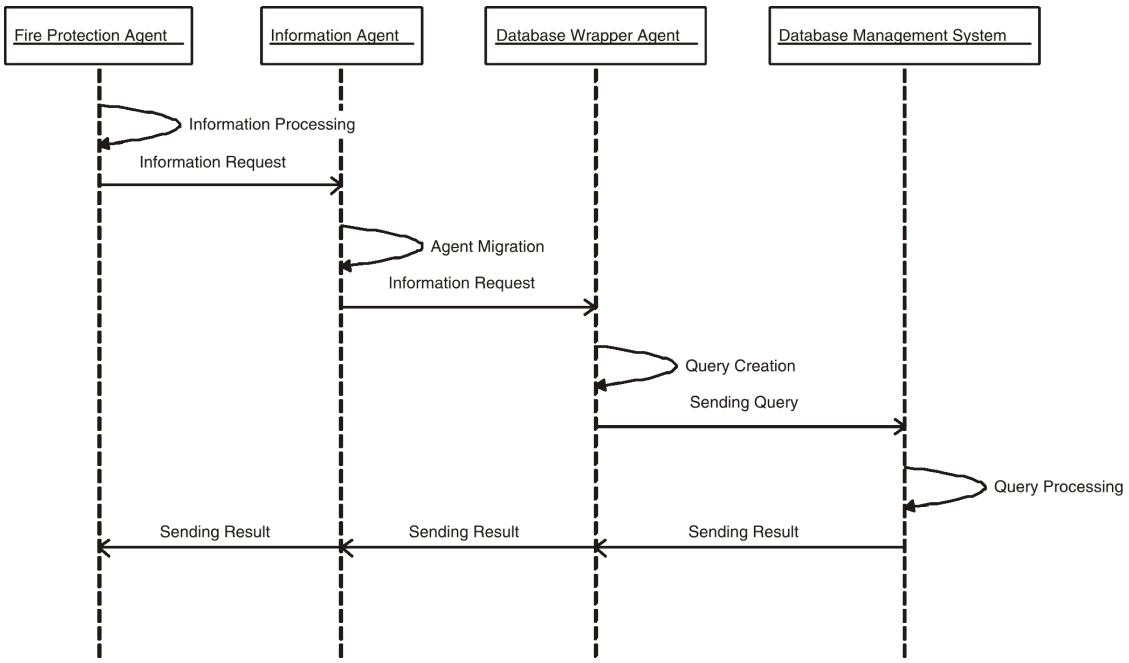

Figure 2: Communication flow to request planning information in Madita

complexity of design tasks, the limited individual domain knowledge of the participating structural designers and, finally, an intense competition pressure. During the coordination of the dynamic and interlaced planning processes typical deficiencies, conflicts and quality losses can occur affecting the productivity of the whole design process. In the analysis of the entire planning process four substantial domains can be identified that, then, must be embedded in a multi-agent system for structural design: i) the participating specialized design experts and organizations, ii) the specific structural design processes, iii) the associated (partial) product models and iv) the applied, heterogeneous engineering standard software (Bilek and Hartmann 2003).

Based on this decomposition, a general agent model for structural design has been developed for supporting the individual designers in their work and minimizing typical planning conflicts. Accordingly, the agent model for collaborative structural design (ACOS) contains the following four submodels: i) the agent-based co-operation model, ii) the agent-based process support model, iii) the agent-based software integration model and iv) the agent-based product model. Each submodel implies a set of task-specific interacting software agents. Figure 3 illustrates the arrangement of the relevant agents as developed in the research project.

Within the *agent-based co-operation model* human experts and their assigned organizations (e.g. specialized planning offices, authorities, building companies) are represented by co-operation agents. The individual designers thereby are directly assisted by their assigned personal agents with respect to coordination and cooperation issues. By that, the personal agents operate as interfaces between human interaction and multi-agent system.

The *agent-based process support model* is to control and distribute the complex workflows, related to the projects, to the appropriate designers. In particular, the workflow agent links design activities to project resources such as product models, applications/ standard software and human experts. For this reason, we have implemented a workflow agent based on the petrinet theory.

The integration of heterogeneous engineering standard software like CAD (Computer Aided Design) or finite element applications, is provided through the implementation of application-specific wrapper agents. Within the *agent-based software integration model* wrapper

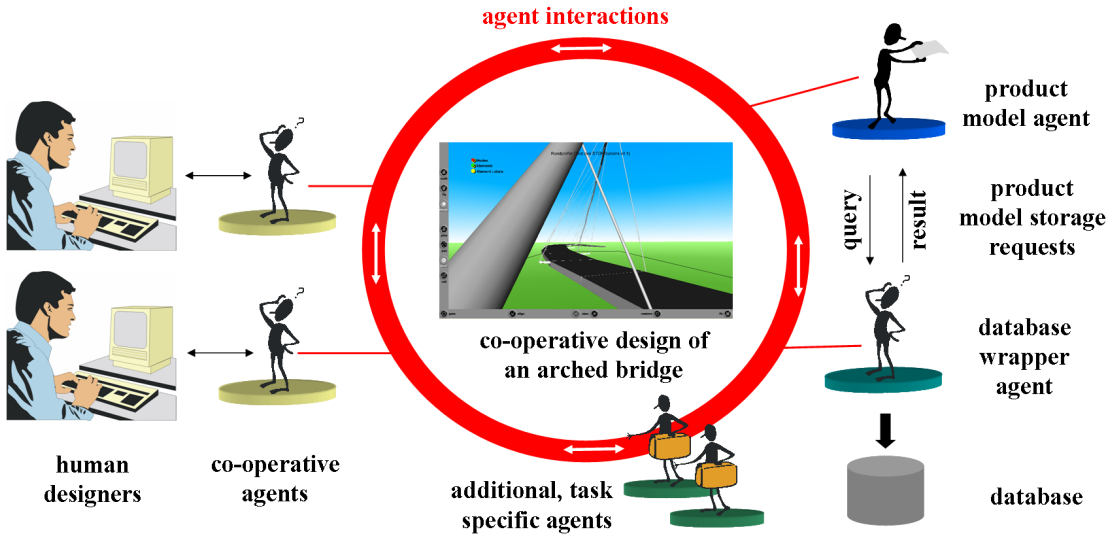

Figure 3: Elementary software agents and product model processing within ACOS

agents act as interfaces between MAS and locally installed software. Through this, corresponding software is made available to all the other agents and/or human design experts in the MAS. In this context, a database wrapper agent has been modeled and implemented, mainly, to store the product model data processed by the product model agents.

The *agent-based product model* is based on the decomposition of the entire structure into smaller structural subsystems. Each of these subsystems contains structural entities that have to be persistently stored and co-operatively managed. For that, a product model agent has been implemented checking structural dependencies, retaining product model consistency and interacting with the database wrapper agent for storage reasons. The product model used is in conformity with the CIS/2[6] standard, as specified in ISO 10303 part 230.

Comparing both approaches in the context of information retrieval, numerous similarities can be identified. On the one hand, both MAS make use of database wrapper agents to store complex, project-specific information, on the other hand, there are agents of a similar category (information agents, product model agent) that interact with database wrapper agents.

3 Information wrapping

Specific information and data have to be stored persistently in a variety of heterogeneous hard- and software systems. For this reason, the embedding of existent legacy database systems or additional external resources into new software systems is mandatory. Accordingly, the concept of *information wrapping* appears as an important aspect – besides the targetted support of concurrent design activities.

The information wrapping approach is already established in research and practice (Sneed 1999). The basics of this approach imply a kind of wrapping module that offers the capability to access external resources independently from their type or technical conditions. By that, the wrapping module can be used as an application specific interface between the encapsulated external resource and the requiring software system. Even the Foundation for Intelligent Physical Agents (FIPA) has already incorporated this concept in its specifications for the design of multi-agent systems (FIPA 2000). The *FIPA software integration specification* makes use of so-called wrapper agents that agentify affiliated external resources. These wrapper agents provide a public interface to specific external resources although these resources usually cannot be accessed from other software agents in the MAS directly.

Wrapper agents are usually implemented in terms of stationary agents. They act on the host where the resource applied is located. In the case of encapsulating database systems, however, this is not mandatory, in general. Customarily, database systems are accessed remotely from any authorized host. Wrapper agents, however, provide an easy-to-use interface to the services provided by the integrated resources. Thus, they hide the often complex, technical specifications required for accessing the encapsulated data resources. In the case of accessing relational or XML-based database systems, the wrapper agents must be capable of establishing connections to these resources. This can be achieved by using standardized, resource-specific access protocols and specifications like SQL (ISO 1999) or XQuery (W3C 2003).

When providing knowledge in a distributed network, it is essential to protect this knowledge against inadmissible access. Usually, this is accomplished by authorization and protection mechanisms using unambiguous logins and passwords. Especially within enterprises, the provision of confidential information is a very sensitive subject. Thereby, it is necessary to separate between business-critical information, required by external users or third parties, and data queried by internal company members. Considering collaborative building design, the conditions are much more complicated: customarily, a plentitude of companies and company members are working together in a heterogeneous, technical project environment. Also, various data resources are to be embeded into such project environments. Herein, the particular data resources are maintained by one of the participating companies. During a collaborative project work, the separation from information, open to all co-operation partners, and specific business-critical information is extremely difficult. This is because the ingredients of the project work are permanently changing (e.g. organisational structures, participating company members and their affiliated roles and access rights in the project). Thus, collaborative building design requires flexible and dynamic authorization methods. By contrast, conventional database systems rather provide static and intricate access methods. As a result, the protection of private information is transfered to the database wrapper agent. This is reasonable because wrapper agents offer a convenient way in protecting confidential information: A wrapper agent can pass only such information to the public which it has access to.

As already pointed out before, in the Darmstadt project wrapper agents communicate with so-called information agents, which are mobile in many cases. In the Bochum project, on the other hand, database wrapper agents interact with the product model agent. In the further context of this paper, we define both agent types as *client-like agents*. Client-like agents define their information queries in a public known query language that must be interpreted by the requested database wrapper agent. After the reception of such a query, the wrapper agent maps it into a database-dependent query. It is evident, that the client-like agents must own knowledge about what they are querying. However, they do not know how to query the desired information from the database itself.

In the two above mentioned research projects, the XML-based database system Tamino (Software AG 2003) and XIndice[7] as well as the open source, relational database system MySQL[8] have been integrated into the two multi-agent systems by means of database-specific wrapper agents.

4 Modeling the database wrapper agent

The architecture of the database wrapper agent consists of three main components as shown in figure 4: i) the **communication component** acting as a public interface to various agents in the MAS, ii) the **processing component** for compiling and mapping of incoming queries and iii) the **database component** for establishing a database connection.

As a result, it is possible to adapt the wrapper agent architecture to various tasks within both projects. Thus, in adapting the database component, a great number of database types can be

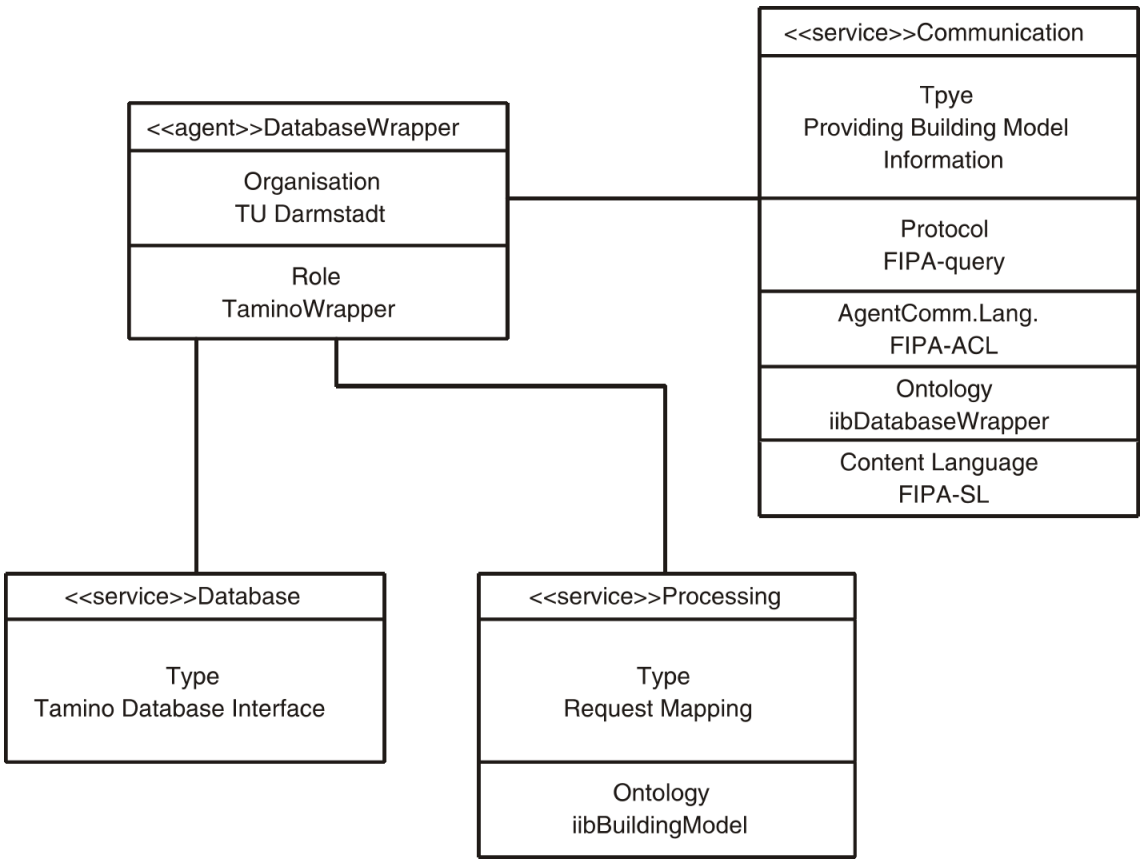

Figure 4: Service structure of the database wrapper agent referred to the Darmstadt project

integrated into the multi-agent systems. Moreover, it is possible to replace the database module by similar components aiming at the integration of conventional engineering software. In doing so, the wrapper agent concept is no longer limited to merely accessing database systems, it is rather flexible to integrate additional engineering applications.

Within the research work, the database wrapper agent is charged with its tasks primarily from client-like agents, being information agents in the Darmstadt project or product model agents in the Bochum project. Each of these client-like agents internally is based on an own specific model of reality needed to perform particular tasks. While requesting wrapper agents for some information, the client-like agent has to submit relevant parts of its internal model to the wrapper agent. As a consequence, the wrapper agent must be capable of dealing with the client agent's domain-specific knowledge. Both interacting agents, i.e. the client-like agent and the wrapper agent, need to "speak" the same language. This language has to be interlocked with a common vocabulary, as defined in so-called ontologies. For that reason, the next two sections deal with the communication processes and their underlying ontologies, needed to establish appropriate agent interaction.

5 Agent communication and ontologies

To support agent communication, the FIPA has defined so-called agent **interaction protocols (IP)** and **communication acts (CA)**. An interaction protocol, hereby, describes a complete communication dialogue between two interacting agents instructed with the solution of a common problem. Figure 5 shows the FIPA-query-inter-action-protocol described in AUML[9]. The FIPA-query-interaction-protocol models the interaction between two agents - one querying (initiator) and one answering (participant). After receiving the communication act *"query"*, the

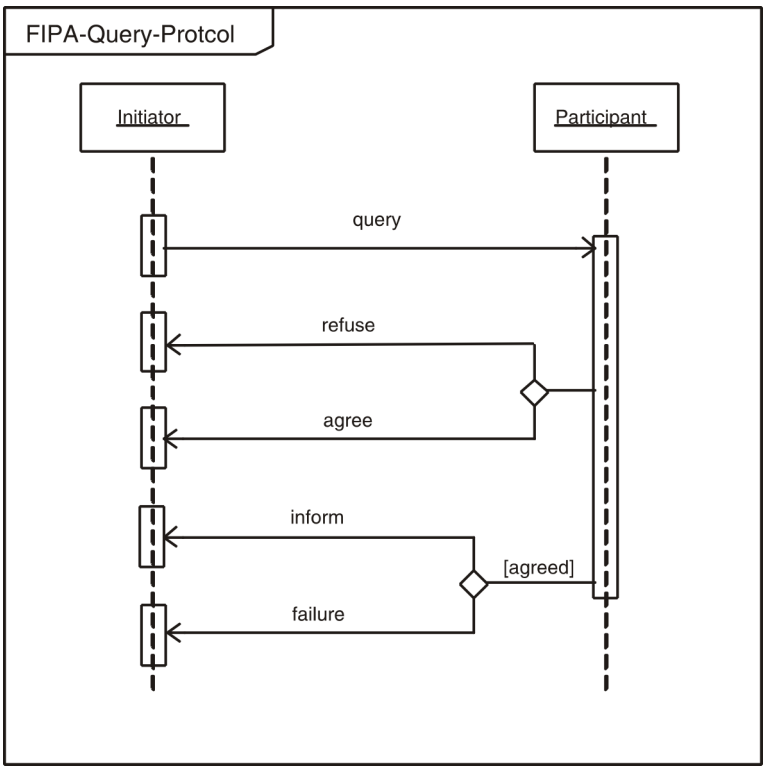

Figure 5: AUML sequence diagram of the FIPA-Query-Protocol

participant will inform the initiator whether it *"refuses"* or *"agrees"* in accepting the received query. Subsequently, the participant will use its specific competency modules to process and prepare the queried information. If the data processing is successfull the participant will transmit the solution to the initiator using the CA *"inform"*. Otherwise, the participant answers with a *"failure"*.

The wrapper agent's communication component is based on the illustrated FIPA-query-interaction-protocol. By that, the wrapper agents as well as the client-like agents need to implement the above introduced communication acts in terms of JADE-behaviours on a technical level. Thus, the particular CAs are linked to internal agent actions. The client-like agent e.g. starts the communication protocol by sending a message with the CA *"query"*. The message content, thereby, specifies the identification data, such as login and password, and the data request string expressed in the database-specific query language. Next, the wrapper agent replies the query either using *"agree"* if the database login is successful or with *"failure"* if failed. In the case of failure, the wrapper agent closes the communication. If the login has been successful the database wrapper agent passes the received request string to its processing component which then transforms the request to a database transaction. Details are given in section 6.

In addition to the realized communication acts and interaction protocols, several ontologies havae been developed by which the homogeneous vocabulary used in the message contents is defined. The first ontology (the *request ontology*) determines the query model defining the three action tags *"select"*, *"insert"* and *"update"* (Fig.6). The request ontology is fundamental for each database wrapper agent implementation because it matches the agent's core functionality. Depending on the project related tasks and knowledge, further ontologies are obligatory describing the specific technical contents and product models. In the Darmstadt project, an ontology defining the structure of a building is implemented (see Fig. 7). By contrast, in the Bochum project an ontology defining the structural system of an arched bridge has been created. Both ontologies are different to each other because of the distinguished models used in Bochum

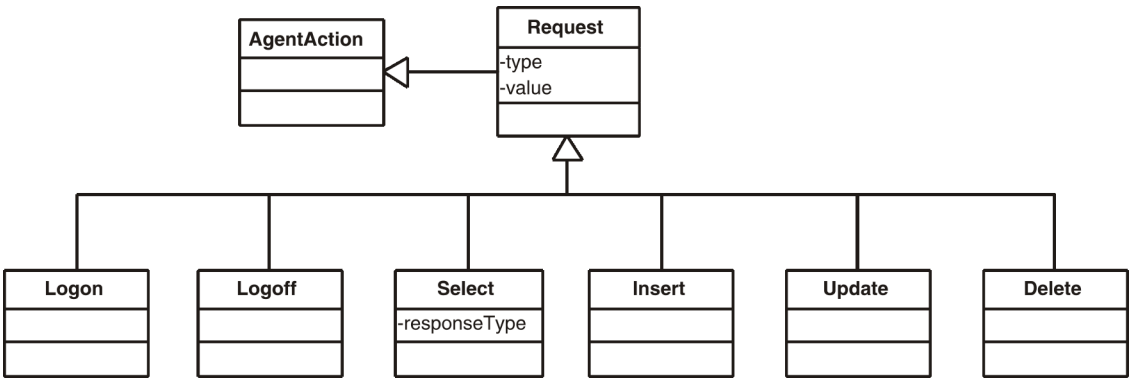

Figure 6: Class diagram of the request ontology

and Darmstadt.

Within the request ontology a “select“-query contains three attributes:

1. The *selectType*-attribute determines the query language. Possible values are “XPath“ for the XPath-specification of the W3C (W3C 1999), “FLWR“ for the XQuery-specification of the W3C (W3C 2003) and “SQL“ for the Structured Query Language (ISO 1999).
2. The *response*-attribute defines the format of the expected result. Presently, possible values are “TEXT“, “OBJECT“ or “XMLSTRING“.
3. The *value*-attribute contains the query string itself. The query string thereby has to comply with the given selectType.

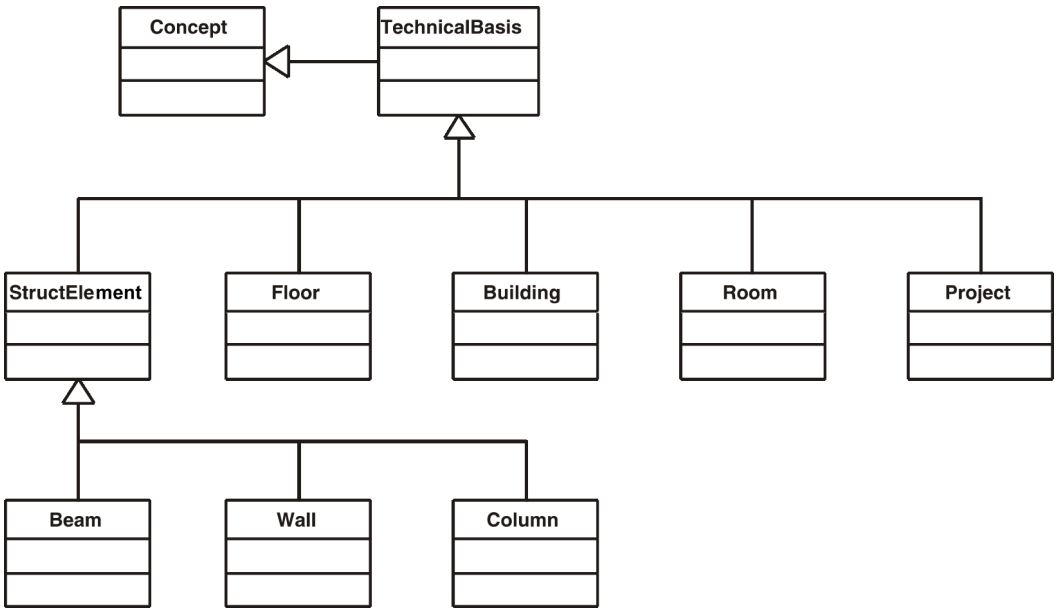

Figure 7: Class diagram of the building design ontology

6 Mapping of queries and results

To describe technical contents, an ontology adapted to a product model is used. The product model specification enables agents, communicating, in interpreting the received message contents correctly. All necessary entities and methods of the product model are incorporated in the ontology. Consequently, in both research projects domain-specific ontologies have been

implemented: one ontology for describing the structure of a building (Darmstadt), one for describing the structural system of an arched bridge (Bochum). Figure 7 exemplifies parts of the technical ontology for the fire protection domain. The building design ontology contains information about the building structure and the affiliated fire protection elements. By contrast, in the domain of structural design the related ontology specifies structural elements, structural subsystems, materials, loads and forces, constraints, etc.

The wrapper agent's processing component maps received ontology-based entities to internal objects which then can be used to transmit appropriate database actions. This presumes that the wrapper agent has knowledge about the relation between the ontology entities and their storage location, inside the database. Thus, the wrapper agent implementation of the processing module is always related to a data specific ontology. If the wrapper agent is requested for some information it queries the database, takes the received results and maps them to ontology-based entities. Next, the wrapper agent takes these entities and sends them back to the requesting client-like agent. Subsequently, the client-like agent can easily handle the received entities.

One might argue that the necessity to adapt the wrapper agent's processing component and product model ontology to the project related requirements forms a limitation. Due to the fact that the database wrapper agent provides an ontology-based interface to the encapsulated database, it is, of course, not very time-consuming to implement various agent modules, capable of interacting with the same wrapper agent. The definition of a common ontology that can be used by several interacting agents is more comfortable than implementing a static point-to-point communication that can be transfered to other agent implementations only with considerable effort.

7 Conclusions

In this contribution, a database wrapper agent concept has been presented for integrating complex design information in distributed engineering environments. So-called database wrapper agents, introduced in the paper, are capable of accessing legacy databases. These agents act as interfaces to specific agent-based engineering systems. Every "client-like" agent may make use of the wrapper agent to obtain project specific information and to fulfill its assigned tasks. The implemented wrapper agents consist of three basic components that have been modeled with respect to reusability such that a plentitude of resources, particularly database systems, may be "wrapped" easily.

We experienced that the wrapping technique proves as a qualified method to integrate legacy resources into multi-agent systems. The basic concept of our approach maps database queries, affiliated to a public, data-specific ontology, to transactions that are performed on the encapsulated database. The located information can then be reprocessed in the multi-agent system.

The wrapping concept is not only limited to database systems but can also be transfered to the integration of additional engineering standard software like finite element applications or CAD software. First experiences in this context verified this hypothesis.

8 Endnotes

1) DFG : Deutsche Forschungsgemeinschaft (German Research Foundation)

2) SQL : Structured Query Language, part of ISO 9075 (ISO 1999)

3) XML : Extended Markup Language, part of W3C specifications

4) JADE : Java Agent Development Framework, `http://jade.cselt.it`. JADE is an open source agent platform developed by the Telecom Labs Italia.

5) FIPA : Foundation for Intelligent Physical Agents, http://www.fipa.org. FIPA is a non-profit organisation aimed at producing standards for the interoperation of heterogeneous software agents.

6) CIS/2 : CIM steel integration standard, http://www.cis2.org. CIS/2 is a set of formal computing specifications that allows the modeling of steel structures from the analysis model over the design model to the fabrication model.

7) XIndice is part of the open source XML Apache projects, http://xml.apache.org/xindice/.

8) MySQL is prevalently used in web-applications, http://www.mysql.com.

9) AUML : Agent Unified Modeling Language, http://www.auml.org. The FIPA is actually working to adopt the AUML methodology into its specifications.

9 References

Bellifemine, F. (2004). Jade – Java Agent Development Framework. Turin, Italia. Tilab Italia, http://jade.cselt.it (as of 1.4.2004).

Bilek, J. and D. Hartmann (2002, April). Collaborative structural engineering based on multiagent systems. In *Proceedings of 9th International Workshop of the European Group for Intelligent Computing in Engineering (EG-ICE)*, Darmstadt, Germany, pp. 49–60. VDI Verlag.

Bilek, J. and D. Hartmann (2003, April). Development of an agent-based workbench supporting collaborative structural design. In R. Amor (Ed.), *Proceedings of the* 20th *CIB W78 Conference on Information Technology in Construction*, Weiheke Island, Neuseeland, pp. 39–46. University of Auckland.

DFG (2000). DFG Schwerpunktprogramm 1103 – Vernetzt−kooperative Planungsprozesse im Konstruktiven Ingenieurbau. http://www.dfg-spp1103.de (as of 1.4.2004).

FIPA (2000). Fipa agent software integration specification, id: 00079. Foundation for Physical Intelligent Agents, http://www.fipa.org/specs/fipa00079 (as of 1.4.2004).

FIPA (2004). The FIPA specification life cycle. Foundation for Physical Intelligent Agents, http://www.fipa.org (as of 1.4.2004).

Friedmann-Hill, E. (2004). *Jess in action – Java rule-based systems*. Manning Publications.

ISO (1999). Information technology – database languages – SQL – part 1-5, INCITS/ISO/IEC/ 9075-1/-5. International Organization for Standardization (ISO).

Meissner, U. F. and U. Rueppel (2000). Agentenbasierter Modellverbund für die kooperative Gebäudeplanung. Antrag zum Forschungsprojekt des DFG-SPP 1103.

Sneed, H. M. (1999, May). Software-Kapselung – Eine Technik für die Einbindung bestehender Software-Bausteine in neue Applikationen. In *Workshop Software-Reengineering, Bad Honnef, Germany*.

Software AG (2003). Tamino XML server. Darmstadt, Germany. http://www.tamino.de (as of 1.4.2004).

W3C (1999). XML Path language (XPath), version 1.0, W3C recommendation. World Wide Web Consortium, http://www.w3.org/TR/xpath (as of 1.4.2004).

W3C (2003). XQuery 1.0: An XML query language, W3C working draft. World Wide Web Consortium, http://www.w3.org/TR/xquery (as of 1.4.2004).