Managing a Relational Database with Intelligent Agents

Ira Rudowsky[1], Olga Kulyba[1], Mikhail Kunin[1], Dmitri Ogarodnikov[2]
and Theodore Raphan[1,2]

[1]*Institute of Neural & Intelligent Systems*
Dept of Computer & Information Science
Brooklyn College of CUNY
e-mail: rudowsky@brooklyn.cuny.edu

[2]*Department of Neurology*
Mt Sinai School of Medicine
e-mail: raphan@nsi.brooklyn.cuny.edu

Abstract

A prototype relational database system was developed that has indexing capability, which threads into data acquisition and analysis programs used by a wide range of researchers. To streamline the user interface and table design, free-formatted table entries were used as descriptors for experiments. This approach potentially could increase data entry errors, compromising system index and retrieval capabilities. A methodology of integrating intelligent agents with the relational database was developed to cleanse and improve the data quality for search and retrieval. An intelligent agent was designed using JACK™ (Agent Oriented Software Group) and integrated with an Oracle-based relational database. The system was tested by triggering agent corrective measures and was found to improve the quality of the data entries. Wider testing protocols and metrics for assessing its performance are subjects for future studies. This methodology for designing intelligent-based database systems should be useful in developing robust large-scale database systems.

1. Introduction

The development of relational databases has significantly improved the performance of storage, search, and retrieval functions [1] [2]. With the expansion of scientific data to include non-textual information such as digital streams representing analog and video information, relational databases have been extended to incorporate these new modalities [3-7]. Recently, we have designed and implemented a relational database system, using Oracle 9i, and interfaced it with a data analysis application. This demonstrated that a flexible interface between applications and databases are potentially important ways to enhance data mining capabilities [8]. The system stores metadata describing external files of various modalities, including digitized analog channels, event channels and video images linked to the data. A graphical, user interface provides for robust querying of the metadata and links have been established between the query results and the visualization/analysis program. As a result, selected query results can be displayed immediately. The rapid query and link capability enables the user to find potential relationships between trials of experiments that would otherwise be difficult for researchers to determine. As the system developed, it became necessary to use free-format table entries as descriptors for experiments to maintain manageable tables. This has raised issues of maintaining data quality when it is entered so that the records are properly indexed for efficient searching.

Data quality problems due to a wide range of errors are prevalent in a number of database applications ranging from simple entry to issues of data quality in genomic studies, which could impact data analysis and further research [9, 10]. A number of software tools have been developed to address problems of data quality. One way to improve data quality is to utilize a technique commonly known as data cleansing, which entails examining the database following data entry to insure that errors in the data are minimized [11]. Methods of data cleansing include eliminating duplicate records as a result of merged databases [12] or decomposing data into elemental parts and reassembling the data and verifying them across records [13]. A more systematic technique represents the cleansing as a directed acyclic graph of data transformations. The data cleansing is performed as an iterative procedure that traverses the graph [14]. An alternate approach for data cleansing using artificial intelligence was to utilize an expert system engine, Java Expert System Shell (JESS),

together with a set of rules for data cleansing [15]. JESS is a rule engine and scripting environment written entirely in Java. JESS enables the creation of software that has the capacity to "reason" using knowledge supplied in the form of declarative rules. The validation and verification stage, however, requires human intervention. An overview of the problems with data cleansing and their solutions is found in [11, 16]. All presently proposed solutions to the data cleansing problem are generally “reactive” in the sense that the data cleansing is performed after the accumulation of large amounts of data.

Within the past decade, agent-oriented programming was introduced [17] and has the potential for efficiently dealing with improving the data quality of free formatted fields. These agent-based systems differ from classical rule-based expert systems in that they behave autonomously and interact with each other. An important idea driving agent-based programming is the representation of agent properties in terms of belief, desire, and intention (BDI). An agent is specified in terms of its capabilities (things the agent can do), a set of initial beliefs, a set of initial commitments (an agreement to perform a particular action at a particular time) and a set of commitment rules. Capabilities are used by agents to decide whether to adopt commitments; an agent will not adopt a commitment to perform an action if the agent determines that there is no possibility for performing that action. The set of commitment rules determines the performance characteristics of the agent. Each commitment rule contains a message condition, a mental condition and an action. To determine whether a commitment rule fires, the message condition is matched against the message the agent received and the mental condition is matched against the agent’s beliefs. If the rule fires, the agent then commits to performing the action. For example, agent A sends a commitment request in a message to agent B. Agent B will accept or reject the request based on the details of the request, its behavioral rules, and its current mental model. B will then send a message to A indicating acceptance or rejection of the request. If B accepts the request, it agrees to attempt to perform the requested action at the requested time if possible.

The purpose of this study was to develop and present a methodology for integrating intelligent agents with databases to improve the efficiency and robustness of the integrated system. We have implemented a prototype agent design using JACK™ (Agent Oriented Software Group) and developed an interface to Oracle. We have also tested the system by deliberately triggering agent actions. The results of this experimental study show that agent actions improve data integrity. A complete experimental study with metrics for assessing the performance over a wide range of users is ongoing, but is beyond the scope of this paper and is a subject for a future paper.

2. Prototype Intelligent Agent

In this study, an intelligent agent package, JACK™ Agent Language (Agent Oriented Software Group http://www.agent-software.com/shared/home), was utilized to build the intelligent agent database interaction. The JACK™ Agent Language is a development environment that extends and is fully integrated with the Java programming language. It defines new base classes, interfaces, and methods as well as provides extensions to the Java syntax to support new agent-oriented classes, definitions and statements. By enabling an agent to pursue its given goals (desires) and adopt the appropriate plans (intentions) according to its current set of data (beliefs), it follows the BDI model of artificial intelligence [17].

The class-level constructs that JACK™ employs include Agents, Events, Plans and BeliefSets. Agent classes are used to define the behavior of an intelligent software agent by specifying all internal and external events that it will handle, events the agent can post internally to be handled by other plans, events the agent can send externally to other agents, plans the agent can execute, and beliefsets the agent can refer to. When an agent is instantiated, it waits until it is given a goal to achieve or experiences an event that it must respond to. The types of events an agent responds to include internal stimuli representing events an agent sends to itself or external stimuli which are messages from other agents or percepts that an agent receives from its own environment. JACK™ provides two categories of events.

1. A normal event in which the agent reacts to transient information in the system. In the database quality application, this would occur if typing errors were made while entering free formatted information. The agent would then select the first applicable plan instance for the event and execute only that plan.

2. A BDI or goal directed event that commits the agent to a desired outcome rather than a specific method to achieve that outcome. In this case the agent selects from a set of plans based on relevancy and applicability. If the selected plan fails to execute, the agent executes an alternative plan until it succeeds or runs out of plans from which to choose.

A plan is analogous to an agent’s functions i.e., the instructions the agent follows that attempt to achieve its goals and handle its designated events. Each plan handles a single event, but multiple plans

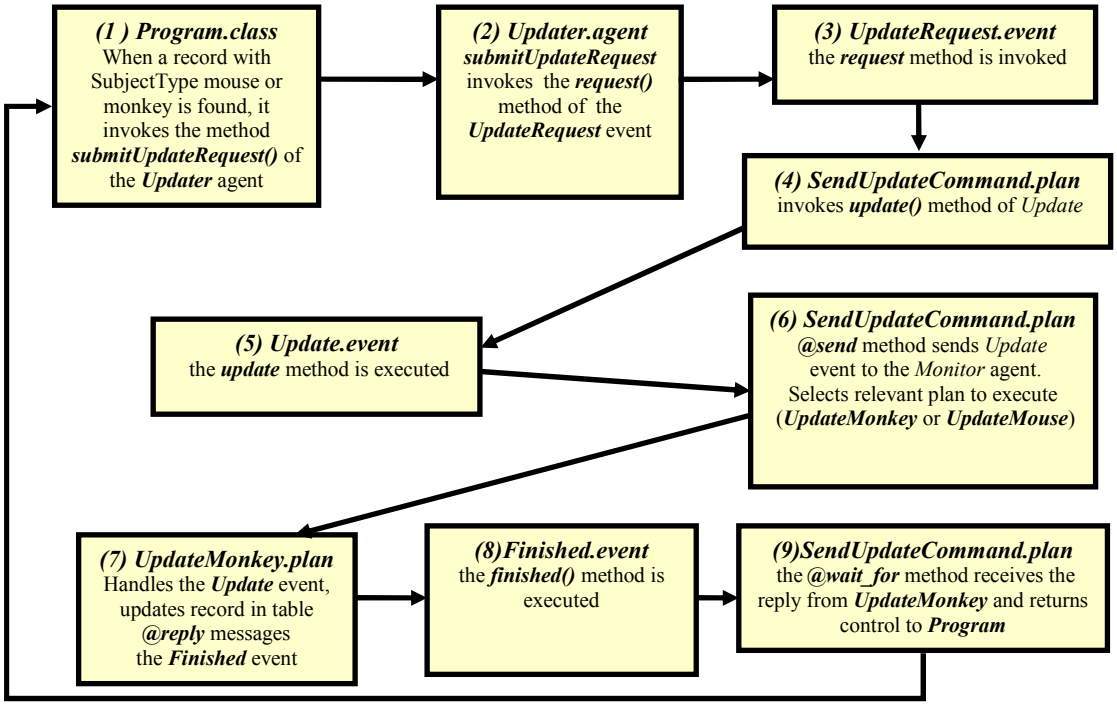
may handle the same event. An agent can discriminate further between plans by executing the plan's relevant() method to determine whether it is relevant for the instance of a given event. From those selected as relevant, the agent can further decide which plans are applicable by executing each plans context() method. Both relevant() and context() are functions within the JACK™ environment.

An agent's beliefs about the world are stored in a beliefset using a tuple-based relational model. In a Closed World relation the tuples stored are believed to be true, those not stored are assumed false. In an Open World relation both true and false tuples are stored; anything not stored is "unknown". Events can be posted when changes are made to the beliefset and thus initiate action within the agent based on a change of beliefs.

3. Intelligent Agent-Database Prototype Experiment

To test these ideas, a prototype system was implemented and experiments were done to deliberately invoke agent corrective measures. This change was accomplished by two agents, Monitor and Updater. The agent system also has two events, Update and UpdateRequest and three plans, SendUpdateCommand, UpdateMonkey and UpdateMouse. Figure 1 illustrates the flow of action steps. When the driver class, Program, found the record that contains 'monkey' or 'mouse' in the field SubjectType (box(1)), it invoked the submitUpdateRequest() method of the Updater agent, (box(2)). This method, in turn, posted a synchronous UpdateRequest event and invoked the request() method of the UpdateRequest event (box(3)). The UpdateRequest event was added to the event queue of the Updater agent to await processing. The Updater agent includes the statement *#uses plan SendUpdateCommand;* which informs the agent

Figure 1. Sequence of events within the system



what plan it should execute to handle any events it receives. Thus, the system progressed to box(4) where the SendUpdateCommand plan handles the UpdateRequest event. First the plan instantiated an Update event and invoked the update method of Update (box(5)). Then the SendUpdatePlan "sent" an update event to the Monitor agent and waited for a reply before continuing. The Monitor agent evaluated the relevant() and context () methods of UpdateMouse and UpdateMonkey in order to choose between two plans of action (box (6)). The selected plan executed and updated the value of SelectType to 'animal' for the given record (box(7)). Upon completion of the plan, an @reply with a Finished event wass issued which invoked the finished method of the Finished event (box(8)). The @wait_for command in SendUpdateCommand received this message and interpreted the response (box(9)). SendUpdateCommand terminated and control returned to Program (box(1)). Thus, through a series of events and messages, the system monitored the database table and under the proper conditions triggered a sequence of steps to update the record. More complicated textual fields could be handled in a similar fashion.

To test the performance of the system, the context of the prototype intelligent agent-database interaction was used with a program that is used to obtain data in scientific experiments performed by a number of investigators using different subjects, storing them in an Oracle 9i database [8, 18, 19]. The relational database contains a field SubjectType within a table named Experiment. This field should contain a keyword that specifies whether the subject is "human" or "animal." However, a researcher may mistakenly type the name of the specific animal being used, which requires corrective measures. The intelligent agent must recognize the character of the information and make appropriate corrections in the field. In the experiment, we deliberately entered this erroneous information. When the agent found 'monkey' or 'mouse' entered in the SubjectType field, this error was detected and the entry was changed to the value 'animal'.

Thus, we have developed a simple intelligent agent-database interaction, which can be extended to consider more complex data entries. It was tested by deliberately introducing errors in data entry and was found to be responsive in correcting errors.

4. Acknowledgements

This work was supported by grants 65397-00 34 and 66442-00 35 from PSC-CUNY, P30 DC05204, DC05222, EY04148 from the NIH.

5. References

[1] C. Meghini, F. Sebastiani, and U. Straccia, "A model of Multimedia Information Retrieval," Journal of the ACM, vol. 48, pp. 909-970, 2001.

[2] R. Weber, J. Bolliger, T. Gross, and H.-J. Schek, "Architecture of a Networked Image Search and Retrieval Syestem," presented at 8th Intl Conference on Information and Knowlwedge Management, Kansas City, Missouri, 1999.

[3] S. Chaudhuri and L. Gravano, "Optimizing Queries over Multimedia Repositories," presented at Proceedings of SIGMOD '96, Montreal, Canada, 1996.

[4] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz, "Efficient and Effective querying by image content," J. Intell. Inf. Syst., vol. 3, pp. 231-262, 1994.

[5] V. E. Ogle and M. Stonebraker, "Chabot: Retrieval from a Relational Database of Images," IEEE Computer, vol. 28, pp. 40-56, 1995.

[6] B. Ozden, R. Rastogi, and A. Silberschatz, "A multimedia support for databases," presented at Proc. 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems, 1997.

[7] C. Traina Jr., A. J. M. Traina, R. R. dos Santos, and E. Y. Senzako, "Support to Content-Based Image Query in Object-Oriented Databases," presented at Proc. of the ACM Symposium on Applied Computing, 1998.

[8] I. Rudowsky, O. Kulyba, M. Kunin, D. Ogarodnikov, and T. Raphan, "Relational Database Linkage of Scientific Applications and Their Data Files," presented at Proc of the IEEE International Workshop on Soft Computing in Industrial Applications, 2003.

[9] T. Redman, "The Impact of Poor Data Quality on the Typical Enterprise," CACM, vol. 41, pp. 79-82, 1998.

[10] K. Orr, "Data Quality and Systems Theory," CACM, vol. 41, pp. 66-71, 1998.

[11] E. Rahm and H. H. Do, "Data Cleaning: Problems and Current Approaches," IEEE Bulletin of the Technical Committee in data Engineering, vol. 23, pp. 1-11, 2000.

[12] M. A. Hernandez and S. J. Stolfo, "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem," Data Mining and Knowledge Discovery, vol. 2, pp. 9-37, 1998.

[13] R. Kimball, "Dealing with Dirty Data," DBMS, vol. 9, pp. 55, 1996.

[14] H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "Declaratively cleaning your data using AJAX," Journees Bases de Donnees, 2000.

[15] E. J. Friedman-Hill, "JESS, the rule engine for the Java platform Available at URL http://herzberg.ca.sandia.gov/jess/," 1999.

[16] J. Maletic and A. Marcus, "Data Cleansing: Beyond Integrity Analysis," presented at Information Quality 2000 - IQ 2000, Boston, MA, 2000.

[17] Y. Shoham, "Agent Oriented Programming," Journal of Artificial Intelligence, vol. 60, pp. 51-92, 1993.

[18] I. Rudowsky, O. Kulyba, M. Kunin, D. Ogarodnikov, and T. Raphan, "Flexible Security and Search Capability for a relational Database with Externally Linked Multimedia data Files," presented at Proceedings of SCI2004 - The 8th

World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, FL, 2004.
[19] I. Rudowsky, O. Kulyba, M. Kunin, D. Ogarodnikov, and T. Raphan, "Relational Database Integrity with Externally Linked Multimedia Data Files," Journal of Integrative Neuroscience, 2004.