Towards Learning Notification Triggers

Steffen Staab
AIFB, Univ. Karlsruhe
76128 Karlsruhe, Germany
Steffen.Staab@aifb.uni-karlsruhe.de

Thomas Barnekow
Fraunhofer IAO, Nobelstraße 12
70569 Stuttgart, Germany
Thomas.Barnekow@iao.fhg.de

Abstract

The detection of complex events in old-fashioned legacy information systems is a prerequisite for their integration into modern intelligent workflow environments. We present an approach that facilitates the embedding of such legacy systems, since it frees the developer from explicitly defining the events that trigger notifications. For this purpose, we combine active database technology with learning mechanisms.

1 Introduction

In order to preserve their competitiveness in todays constantly evolving markets companies need integrated systems supporting cooperation within flexible, seamless business processes. However, business processes spanning many functional departments usually involve multiple isolated legacy information systems, which often resist the integration into common workflow management environments to a large extent. In particular, their design has typically not been developed such that modern facilities would exist that permit outside access into the system. While this situation provides a major obstacle to the implementation of any common workflow management system, it must be of special concern to adaptive, intelligent workflow approaches that require more generic access mechanisms to the individual system components. Nevertheless, the sheer amount of business logic intrinsic in such isolated information systems makes them very valuable company assets, which cannot be appropriately modified or replaced in a short period of time or at a low budget.

In this paper, we propose a solution for one part of this integration problem, namely the notification problem. This notification problem is depicted in Figure 1: We assume a monolithic information system with an underlying database. The problem is to detect complex events, *i.e.* events at the level of business operations, that are relevant to a particular user or that should trigger a transition in a corresponding workflow. Given that the manipulation of, possibly undocumented, code in the information system is too cumbersome, maybe too expensive, and lacks sufficient flexibility as it is developed over a longer period of time, the problem then is to define a mechanism that provides the notifications for relevant events.

Our approach builds on the fact that application code and database technology are usually clearly separated, *e.g.* by a SQL interface. Thus, we may build on active database technology for accessing the database operations from the outside. However, active databases do not quite solve our problem, yet, because database events are on a level of description, *e.g.* SQL operations, that is too unwieldy for specifying notification triggers. Yet worse, corresponding trigger programs that achieve an appropriate level of abstraction may become so complex that it is not conceivable they could be written by hand [Staudt & Jarke, 1996].

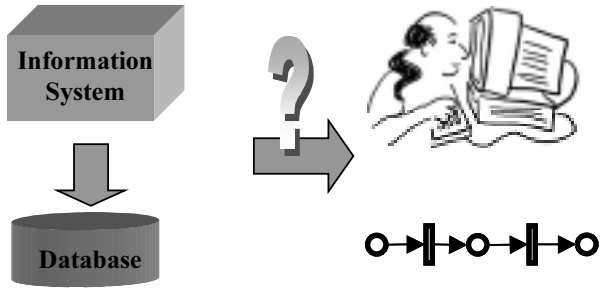

Figure 1: The notification problem

In order to overcome this difficulty we propose to combine active database technology that offers observation at a fairly low level of operations with learning mechanisms that achieve the abstraction necessary for workflow management systems, thus solving the notification problem sketched above.

In the following, we first consider a small example application that shows where our result may bring immediate benefits while leaving the implementation of the given information system untouched – an additional benefit when notifications are to be triggered from the

observation of mission-critical applications. Then, we give an overview of our architecture for learning complex events and detecting their occurrence in the running system. Subsequently, we consider two phases, *viz* the learning phase (cf. Section 4) and the execution phase (cf. Section 5).

2 Example Application

Old legacy information systems that are not integrated into the workflow easily lead to inconsistencies regarding data stored in different information systems and unwanted consequences arising from negligence of available data.

A common result derived from such lack of integration happens as follows:[1] You receive a letter from your insurance company. They offer information brochures and enclose a reply postcard with your address already filled in correctly. Then, you use this postcard to order the brochures - and your former landlord receives the information package, because a recent change of your address has not been propagated to the information system for distribution of information packages.

This situation might be avoided when address modifications in legacy informtion systems trigger corresponding notifications to other information systems. A solution for such a triggering mechanism is what we aim at.

3 Solving the Notification Problem

Our solution for the notification of business operations in an information system is based on the following observation: While many information systems - especially the older ones - are rather monolithic and their intrinsic information flows are hard to monitor, the basic operations in their associated database systems can be easily registered - either by way of listening at the interface or by reporting methods commonly supported in current active database systems, *i.e.* triggers on database tables [Eisenberg & Melton, 1999].

Figure 2 summarizes the overall design of our approach: A business operation carried out on the information system affects the data in its underlying database system (arrow 1). The database events are reported by triggers on the database tables (arrow 2). These reports are used by a production system in order to detect relevant business operations (arrow 3). These business operations can either inititate a workflow transition (arrow 4) or they can trigger a notification to another consumer (arrow 5).

It would be rather time consuming or error-prone for a human programmer to manually code the automatic recovery of business events. Therefore, our design incorporates a learning process (cf. Section 4) during which the functional correspondences between sequences of atomic database events and higher-level business events are semi-automatically acquired by our system. The results of this learning process are then compiled into a set of production rules. At run-time the production rules are defined in an inference engine that can handle interleaved business operations commonly arising in multi-user systems (cf. Section 5).

4 The Learning Process

For each legacy information system, the learning process aims at finding hypothetical rules correctly representing correspondences between business events and sequences of database events. It should be stressed that the learner is not required to find the business events' exact specifications. In fact, the learner should find the most general rules correctly identifying the respective target business event among all the other relevant business events that need to be recovered. For this purpose we use FOIL [Quinlan & Cameron-Jones, 1995], a learning algorithm which, provided with sufficient training data, constructs logic programs that constitute the intensional definitions of target relations representing business events.

[1] This scenario just happened to one of the authors.

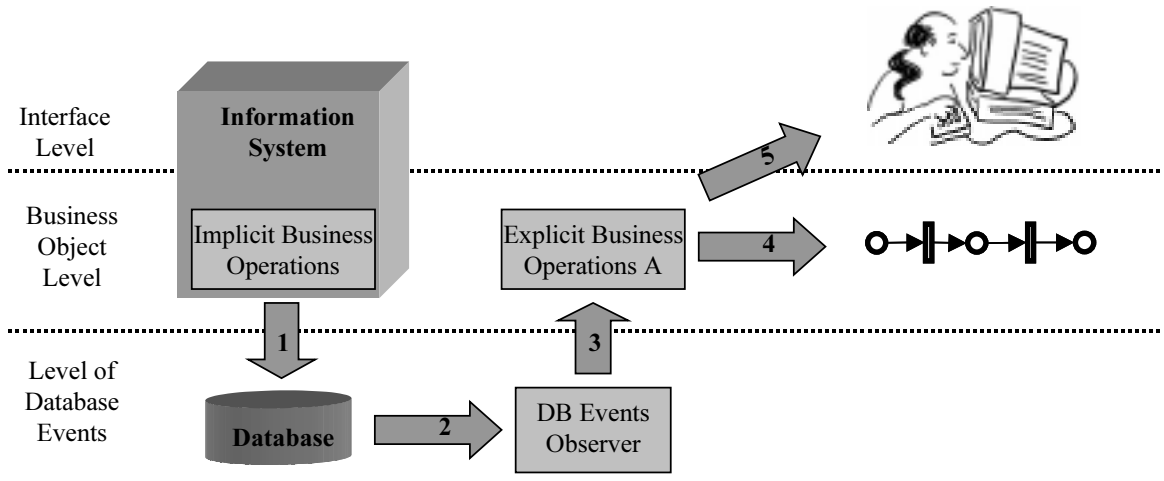

Figure 2: Detecting Business Operations by Observing Database Events

For example, learning the definition of the relation add-customer() representing a corresponding business event in a Computer Aided Selling system might produce the following rule:

```
add-customer(cid, name, age, street, city, zip) :-
    insert-person(cid, name, age),
    insert-address(aid, cid, street, city, zip).
```

In our approach, examples are pairs consisting of a high-level predicate denoting the paremeterized business event and a set of low-level predicates corresponding to database events. In order to acquire the necessary training data the human trainer performs sample business operations using the information system's standard user interface and, simultaneously, provides the corresponding high-level predicates that describe the business events signalling the successful processing of these business operations. In our previous example, the trainer starts a learning cycle by indicating to the system that a sequence of database events corresponding to the creation of a customer will occur next in the database system. Then, he creates a new sample customer. Finally, the trainer indicates the completion of the cycle to the learning mechanism. To sum up, the whole learning process consists of (1) the acquisition of training data for all relevant business events, and (2) the induction of a set of logic programs, one per target business event.

Since FOIL does not scale well with regard to the arity of the background relations [Cohen, 1994] the learning problem must be minimized by applying a number of heuristic filters during the problem formulation step. For example, invariable base relation columns are usually irrelevant and can in most cases be discarded. In preliminary practical tests with SAP R/3 we found out that FOIL was not able to solve a learning problem with background relations having arities of up to 165 in 12 hours. After formulating a minimized equivalent problem as described above FOIL found a correct rule definition in less than 10 seconds.

5 Infering Business Events by Production Rules

In an operating information system the observation of database events does not generally yield well-sorted sequences of database operations that clearly correspond to business operations. Due to multi-user interactions these sequences may be interleaved and their beginnings and endings are commonly unmarked.

In order to untangle these sequences of database events the predicates learned during the learning phase are compiled into a set of CLIPS production rules, which are then defined in the forward-chaining inference system Jess [Friedman-Hill, 1999]. At runtime, facts corresponding to the observed database events are asserted in the inference system. This will eventually trigger previously learned rules and, thus, infer assertions corresponding to the business events.

6 Related Work

Our work builds on approaches from the fields of computer supported cooperative work (CSCW), database technology and learning theory. We may thus classify related work accordingly.

6.1 CSCW

Notification mechanisms for complex events are widely used for monitoring web sites [Pannu & Sycara, 1996], or modern groupware technology [Patterson et al., 1996]. They contribute to such diverse fields as concurrent engineering and result sharing of mult-agent systems [Berndtsson *et al.*, 1997]. Nevertheless, this kind of technology is hardly ever mentioned with regard to the integration of old-fashioned legacy applications. This may turn out to be a large drawback, in particular for intelligent workflow management systems, because the support of these (often) mission-critical applications may determine their breakthrough to a very large extent.

This negligence may be surprising, because as far as common modeling methodologies are concerned the heterogenity of common information technology environments have been treated by well-known modeling approaches [Krishnakumar & Sheth, 1995]. However, these modeling approaches typically start at a level of description that lies beyond the one available in old-fashioned information systems. Here, our approach covers a significant part of the gap and mediates between the information system with its low level database events and the high-level description of business operations considered in the workflow management system.

6.2 Database Technology

With regard to database technology, our approach roughly compares to the problem of monitoring materialized views over time. Database monitoring is not a new problem. Several approaches exist, such as polling by clients, active database technology, or distributed programming languages. "However, the question how to compute the necessary changes is not answered by these base technologies if the view definition is reasonably complex or if views are defined on top of each other. The corresponding trigger programs become so complex that it is not conceivable they could be written by client developers."

This citation from Staudt & Jarke [1996] makes clear why common database monitoring techniques may not be applied to our notification problem. While new monitoring methods such as proposed by Staudt & Jarke [1996] may alleviate the monitoring problem in general, for the purpose of our notification problem one would have to make a rather inappropriate assumption, namely that the exact view one wants to monitor is known to the devel-

oper of the notification system. While this may apply to particular problem settings, it does not hold in general – and it is not required by our approach.

6.3 Machine Learning

Our approach builds on induction of logic predicates and logic views the methodology for which is presented, by Quinlan & Cameron-Jones [1995] and Cohen [1994], respectively. To the best of our knowledge no comparable approach has been established for learning from processes.

7 Conclusion and Future Work

In general, learning approaches cannot be expected to guarantee 100% precision and recall. Here, our approach clearly meets its inherent limits for some applications. However, for many applications, *e.g.*, for maintaining customer addresses in a landscape of different information systems, a well-tested solution that has been learned automatically and, hence, is cheap, easy and fast to implement is highly favorable over systems that require to maintain consistency by hand – and thus may easily fall prey to exactly the same problem of correctness, but with larger costs for realization.

At the moment, we are developing an experimental setting that will allow us to determine the overall efficacy of our approach, as well as precise factors for precision and recall. The results gained from this experiment will then allow us to estimate the suitability of our notification mechanism for particular systems.

For the future we think of two extensions for our approach. First, one may pursue the learning of an integration beyond the notification problem treated in this paper (cf. Figure 3). The idea is to include a corresponding learning phase for another information system B – which then reflects the changes caused by information system A. Thus, the old-fashioned legacy information system could not only take effect on the workflow, but also vice versa.

Second, the learning environment and parameters could be stored for easy adaptation to modifications of the information system. Updates to the structure of the information system, such as might have appeared due to Y2K problems or due to the introduction of the Euro, might affect the definition of the notification triggers. When the learning environment saves all its parameters for later reuse, the adaptation to such modifications might be performed almost automatically in a renewed learning phase.

Acknowledgements

We thank Jürgen Ziegler für helpful comments on a first draft of this paper. The first author was supported by the German Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) under grant number 01IN802 (project "GETESS").

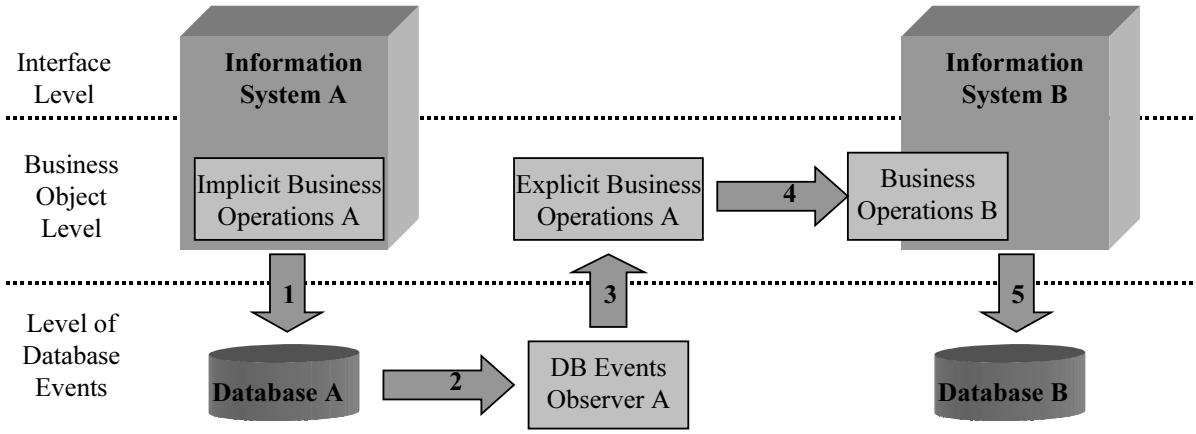

Figure 3: Integrating Information Systems

References

[Berndtsson *et al.*, 1997] M. Berndtsson, S. Chakravarthy & B. Lings. Result sharing among agents using reactive rules. In*: CIA-97 - Proceedings of the 1st Workshop on Cooperative Information Agents*. LNCS, Springer, 1997, pp. 126-137.

[Cohen, 1994] William Cohen. Recovering software specifications with inductive logic programming. In: *AAAI-94: Proceedings of the 11th National Conference in Artificial Intelligence*. Seattle, Washington, July 31-August 4, 1994, pp. 142-148, Menlo Park/Cambridge, AAAI Press/MIT Press.

[Eisenberg & Melton, 1999] Andrew Eisenberg & Jim Melton. SQL:1999, formerly known as SQL3. In*: SIGMOD Record*, 28(1), 131-138, 1999.

[Friedman-Hill, 1999] Ernest J. Friedman-Hill. Jess, The Java Expert System Shell. Technical Report SAND98-8206 (revised), Distributed Computing Systems, Sandia National Laboratories, Livemore, CA, 1999. Available at http://herzberg.ca.sandia.gov/jess.

[Krishnakumar & Sheth, 1995] Narayanan Krishnakumar & Amit Sheth. Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations. In: *Distributed and Parallel Databases*, 2(3), 155-186, 1995.

[Patterson *et al.*, 1996] John F. Patterson, Mark Day & Jakov Kucan. Notification Servers for Synchronous Groupware. In: *CSCW-96 - Proceedings of the ACM Conference on Computer Supported Cooperative Work '96*, pp. 122 –129.

[Quinlan & Cameron-Jones, 1995] John R. Quinlan & R. M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. In*: New Generation Computing*, 13, 287-312, 1995.

[Staudt & Jarke, 1996] Martin Staudt & Matthias Jarke. Incremental Maintenance of Externally Materialized Views. In: *Proceedings of the 22nd VLDB Conference*, Bombay, India, 1996.

[Sycara & Pannu, 1996] Katia Sycara & Anandeep S. Pannu. A Learning Personal Agent for Text Filtering and Notification. In: *KBCS-96 - Proceedings of the International Conference of Knowledge Based Systems*, 1996.