An Architecture for Recovering Business Events Bottom-up[1]

Thomas Barnekow[*], Steffen Staab[+], Jürgen Ziegler[*], and Rudi Studer[+]
[*]Fraunhofer IAO, Nobelstrasse 12, 70569 Stuttgart, Germany
{Thomas.Barnekow, Juergen.Ziegler}@iao.fhg.de
[+]AIFB, Univ. Karlsruhe, PO-Box 6980, 76128 Karlsruhe, Germany
{Steffen.Staab, Rudi.Studer}@aifb.uni-karlsruhe.de

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. Yet, the sheer amount of business logic intrinsic in these information systems makes them very valuable assets, whose modification or extension, when applying standard approaches, is a time consuming and costly process.

In this paper, we address one aspect of the integration problem, namely the detection of complex events[2] relevant to users or to the automatic workflow enactment. We describe a minimally intrusive system recovering high-level business events in legacy information systems by monitoring low-level database events[3] and infering target business events using previously learned rules.

---

[1] To appear in the Proceedings of the HCI International 1999 conference.

[2] Complex events are composed of other, atomic or complex, events. Events signal the fact that corresponding time-consuming operations have been performed successfully. A business event is an event at the business process level. Database events correspond to committed database operations.

[3] We generally assume that information systems are built on top of database systems.

2 Solving the Recovery Problem

Our solution for the automatic recovery of business events in an information system is based on the following observation: While many information systems - especially older ones - are rather monolithic and their intrinsic information flows are hard to monitor, basic events in their associated database systems can easily be registered - either by way of listening at the interface or by reporting methods commonly supported in current database systems, e.g. triggers on database tables.

Figure 1 summarizes the overall design of our approach: A business operation carried out on the information system transforms the state of its underlying database system (arrow 1). Resulting database events are reported by triggers on the database tables (arrow 2). The reports are used by a production system in order to recover relevant business events (arrow 3). These can either initiate a workflow transition (arrow 4) or trigger a notification to a user (arrow 5).

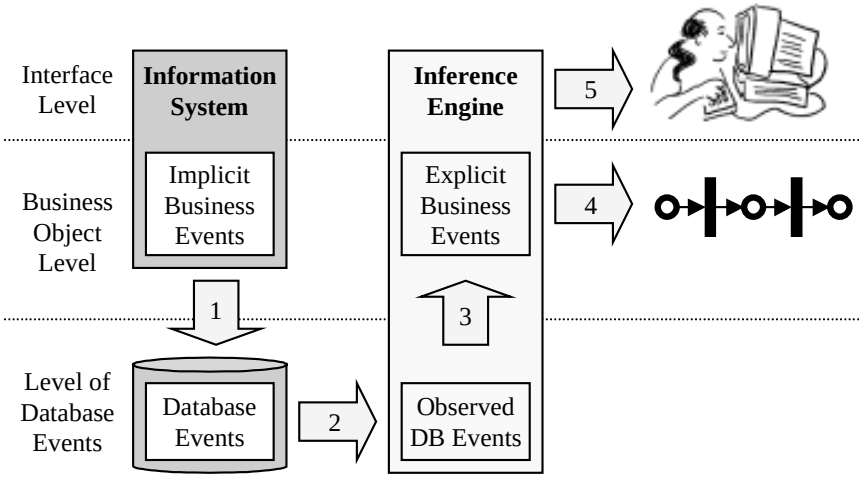

Figure 1: Architecture of the recovery process.

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 3) 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 4).

3 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 and 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.

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 to the system the completion of the cycle. For each information system, 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.

4 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 production rules, which are then defined in a forward-chaining inference system (Forgy 1982). 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.

5 Related Work

Our work builds on approaches from the fields of Computer-Supported Cooparative Work (CSCW), Distributed Artificial Intelligence (DAI), database technology and machine learning.

In the field of Computer-Supported Collaborative Work (CSCW) notification mechanisms for complex events are used for supporting awareness in cooperative working environments (cf. Fuchs et al. 1995, Loevstrand 1991). In the field of Distributed Artificial Intelligence (DAI) complex events are used for result sharing in distributed problem solving architectures (cf. Berndtson et al. 1997). Business events recovered by our system can be communicated using the above mechanisms.

With regard to database technology our approach roughly compares to the problem of monitoring materialized views over time. However, due to the complexity of the general problem (cf. Staudt and Jarke 1996) common database monitoring techniques can not be applied to our recovery problem.

In order to solve the complexity problems our approach builds on algorithms for the induction of logic programs (cf. Quinlan and Cameron-Jones 1995; Cohen 1994), where complex predicates, i.e. rules describing how business events are composed from atomic database events, can be learned from examples.

6 Conclusions and Future Work

We have presented a minimally intrusive system that can recover business events in legacy information systems by monitoring database events and infering target business events using inductively learned production rules.

We are currently developing an experimental setting that will allow us to determine the overall efficacy of our solution, as well as precise factors for its completeness and correctness. The results gained from this experiment will then allow us to estimate the suitability of our integration approach for particular system classes. As of today, our solution appears to be a rather promising approach towards integrating legacy information systems into modern groupware and workflow environments.

References

Berndtsson, M., Chakravarthy, S., & Lings, B. (1997). Result Sharing Among Agents Using Reactive Rules. In P. Kandzia & M. Klusch (Eds.), *Cooperative Information Agents: Proceedings of the First International Workshop, CIA '97, Kiel, Germany, February 26-28, 1997,* pp. 126-137. Berlin: Springer.

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

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence, 19*(1), 17-37.

Fuchs, L., Pankoke-Babatz, U., & Prinz, W. (1995). Supporting Cooperative Awareness with Local Event Mechanisms: The GroupDesk System, *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work - ECSCW'95,* pp. 247-262. Dordrecht, NL: Kluwer Academic Publishers.

Loevstrand, L. (1991). Being Selectively Aware with the Khronika System, *Proceedings of the Second European Conference on Computer-Supported Cooperative Work - ECSCW'91*, pp. 265-278. Dordrecht, NL: Kluwer Academic Publishers.

Quinlan, J. R., & Cameron-Jones, R. M. (1995). Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing, 13*, 287-312.

Staudt, M., & Jarke, M. (1996). Incremental Maintenance of Externally Materialized Views. *VLDB'96, Proceedings of the 22nd International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India,* pp. 75-86. San Fransisco: Morgan Kaufmann.