

The AOODB Workbench: An Environment for the Design of Active Object-Oriented Databases

W. Obermair¹ W. Retschitzegger² A. Hirnschall¹ G. Kramler² * G. Mosnik¹

¹Institut für Wirtschaftsinformatik
Universität Linz, Austria
www.dke.uni-linz.ac.at

²Abteilung für Informationssysteme
Universität Linz, Austria
www.ifs.uni-linz.ac.at

The design of active object-oriented databases includes modeling the structure of objects in the form of attributes and relationships, the passive behavior of objects in the form of operations, and the active behavior of objects in the form of business rules. Business rules state which predefined situations have to be detected by monitoring calendar time and the database state and which reactions have to be taken in order to meet a company's business policy.

An approach that has been followed successfully in conventional database design is to perform database design in two phases: In the first phase, the conceptual design, a high-level graphical representation of the database schema is developed. In the second phase, the logical design, the developed schema is mapped to the data model of the system used for the implementation. We have adopted this approach and provide an environment for the design of active object-oriented databases comprising a graphical editor, a class and rule generator, an active object oriented database system, and a rule debugger (cf. Figure 1):

Graphical Editor. We propose Active Object/Behavior Diagrams (AOBD, cf. [6, 7]) for developing the conceptuel schema of an active object-oriented database. The graphical editor for Active Object/Behavior Diagrams [5] performs syntactic and local semantic consistency checks during the interactive design process and allows to store the generated schema data and visualization data in a GemStone (GemStone

Systems, Inc.) database.

Class and Rule Generator. During logical design, a class and rule generator decomposes the high-level constructs of Active Object/Behavior Diagrams for modeling object structure, passive object behavior, and business rules into lower-level constructs of an existing active object-oriented database system. We have implemented such a mapping for TriGS (see below).

TriGS (Triggersystem for GemStone). TriGS [2, 3] is a prototype of an active object-oriented database system built on top of the commercial object-oriented database system GemStone. TriGS provides for a seamless integration of active rules into the underlying object-oriented data model: Rules are first-class objects, they can be specified and modified dynamically, and they are subject to inheritance and overriding. TriGS provides an expressive event language for specifying composite events whose components are allowed to span not only different transactions but also different applications. Further, the TriGS rule execution model allows the specification of arbitrary points in time for both condition evaluation and action execution and supports serial as well as parallel rule execution.

TriGS-Debugger. The TriGS-Debugger [4] has been implemented to monitor and debug the execution of active rules. The TriGS-Debugger consists of two parts: (1) The tracer implements debugging functions like disabling of rules, event simulation, and breakpoints and keeps track of

*The financial support by SIEMENS PSE Austria under grant No. 038CE-G-Z360-158680 is gratefully acknowledged.

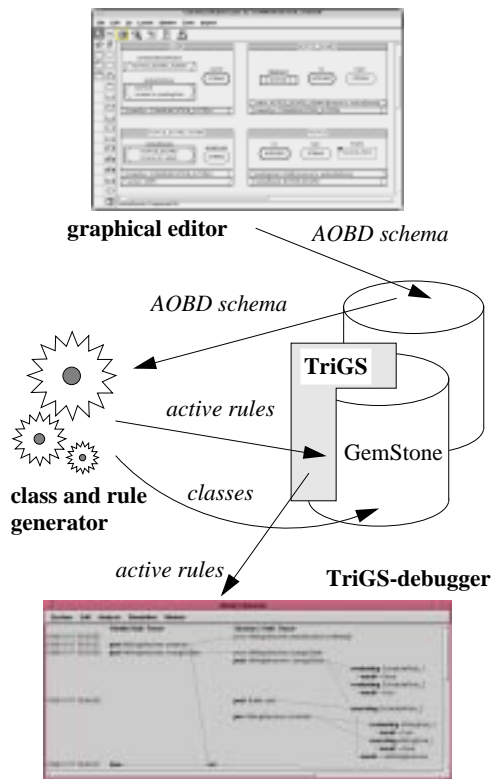


Figure 1: Design Environment

rule execution independent of a user interface. (2) The user interface explains rule behaviour by means of a graphical presentation. This presentation can be customized using a sophisticated filter mechanism.

In the recent literature, other approaches to active database design have been presented (e.g., the IDEA Methodology [1]). However, the emphasize is different in our design environment. We concentrate on the following issues:

- Declarative event language: Similarly as QBE and SQL are better accepted by end users than Relational Algebra, end users are assumed to prefer a declarative event language over an operational one during conceptual design.
- Future calendar events and intervals: Business rules are able to refer to calendar time, including future calendar events as well as future calendar intervals. For example, our event language allows to specify events like “at closing hour of the last Friday of a

month”.

- Reliable and controllable rule execution: Modelers and system users expect business rules to be executed reliably, i.e., transactions performing rule executions are restarted automatically if they are aborted due to serialization failures, and controllably, i.e., specific types of exceptional situations occurring during rule execution are reported by events, which can again be handled by business rules.

References

- [1] S. Ceri and P. Fraternali, “Designing Database Applications with Objects and Rules: the IDEA Methodology,” Addison-Wesley, 1997.
- [2] G. Kappel and W. Retschitzegger, “The TriGS Active Object-Oriented Database System—An Overview,” in *ACM SIGMOD RECORD*, 27(3), 1998.
- [3] G. Kappel, S. Rausch-Schott, and W. Retschitzegger, “A Tour on the TriGS Active Database System—Architecture and Implementation,” in *ACM SAC’98*, ACM Press, 1998.
- [4] G. Kappel, G. Kramler, and W. Retschitzegger, “TriGS Debugger—A Tool for Debugging Active Database Behavior,” submitted for Publication, 2000.
- [5] P. Lang, W. Obermair, W. Kraus, and T. Thalhammer, “A Graphical Editor for the Conceptual Design of Business Rules,” in *ICDE’98*, Comp. Soc. Press, 1998.
- [6] P. Lang, W. Obermair, and M. Schrefl, “Situation Diagrams,” in *DEXA’96*, LNCS 1134, 1996.
- [7] P. Lang, W. Obermair, and M. Schrefl, “Modeling Business Rules with Situation/Activation Diagrams,” in *ICDE’97*, Comp. Soc. Press, 1997.