# FRAMBOISE – an Approach to construct Active Database Mechanisms

Hans Fritschi, Stella Gatziu, Klaus R. Dittrich

Technical Report 97.04

Institut für Informatik, Universität Zürich
{fritsch|gatziu|dittrich}@ifi.unizh.ch

April 15, 1997

**Abstract**

Current active database management systems (ADBMS) are typically realized as monolithic, tightly integrated software systems. Separating the active database mechanisms from the DBMS, would open the opportunity to use them in a variety of ways and environments. In order to build the respective decoupled active database facilities, construction systems are required. The realization of such a construction system is investigated in the FRAMBOISE project. FRAMBOISE is used to build a "ready-to-run" software system that allows the definition and execution of active database mechanisms in the context of specific DBMSs. It comprises specification mechanisms, construction tools and an object-oriented framework that provides implemented components realizing specific ADBMS tasks. This paper outlines the general approach of FRAMBOISE and discusses the first steps of the realization.

## 1 Introduction and Motivation

Active database management systems (ADBMS) support, beyond the traditional functionality of database management systems (DBMS), the monitoring of specific situations and react automatically in predefined ways when respective situations occur. In the last 10 years, considerable effort has been spent towards understanding ADBMSs, and many different proposals have been made (for a comprehensive overview see [18]). Even though the *Event-Condition-Action (ECA) rule* paradigm is now widely accepted for ADBMS, there are significant functional differences across the various systems and prototypes.

Today, active database functionality is rather tightly coupled[1] to the DBMS behind it. A substantial effort is required to adopt active database functionality for various DBMS, which generally aggravates the provision of active database mechanisms. Following a general direction that database research is currently about to take, namely to provide individual database management services that can be used and combined in a variety of ways and in a variety of environments [17, 2, 13], it makes sense to investigate how to provide active database mechanisms as an individual database service. Note that such an approach

[1]Note that this also applies for ADBMSs having a so-called *layered system architecture*. Although the active database components reside in a module built "on top" of a conventional DBMS, they are developed for a particular target DBMS.

would also allow to provide extensive active database features (e.g., expressive rule languages as proposed by existing research prototypes) as "add-on" capabilities of traditional DBMSs which offer no or only limited trigger facilities. This implies, however, the necessity of construction systems in order to build the "decoupled" active database mechanisms.

Inspired by those ideas, we explore in the FRAMBOISE[2] project the realization of an active database construction system. FRAMBOISE is a construction system to build active database mechanisms which are applied in conjunction with traditional DBMSs. These mechanisms behave, together with the underlying (mostly passive) DBMS, in accordance with the characteristics presented in the ADBMS-Manifesto [8] of the ACT-NET Consortium.

This paper outlines the principal concepts of FRAMBOISE. In the next section we sketch the general approach and architecture of the construction system. Section 3 presents the reference architecture of the software built by means of FRAMBOISE. Section 4 addresses the realization and summarizes the current state of the project. Section 5 briefly discusses issues of the application of FRAMBOISE and related work. Finally, chapter 6 concludes the paper.

# 2 The General Approach

FRAMBOISE is a *construction system* which is used to build "ready-to-run" software that allows the definition and execution of *Event-Condition-Action* (ECA) rules in the context of DBMSs. We call the executable software constructed by means of FRAMBOISE an *ECA-System* (ECAS). FRAMBOISE provides implemented components realizing specific ADBMS tasks (e.g., event definition or rule execution), mechanisms for selecting the respective components, and tools to build specially tailored ECASs. It incorporates a generic knowledge and rule execution model, that prescribes which active database behaviour can be realized under what conditions (e.g., depending on the data model of the underlying DBMS). The knowledge and the rule execution model of a concrete ECAS is subsequently derived from the generic knowledge and rule execution model of FRAMBOISE. ECASs can be constructed for DBMSs with either an object-oriented or a relational data model.

A specific ECAS is built by the so-called *ADBMS-implementor* (ADBI) according to the following procedure (see Figure 1): First, the ADBI specifies the ECAS to be built, using constructs offered by the

[2] a **FRAM**ework using o**B**ject-**O**r**I**ented technology for **S**upplying active m**E**chanisms

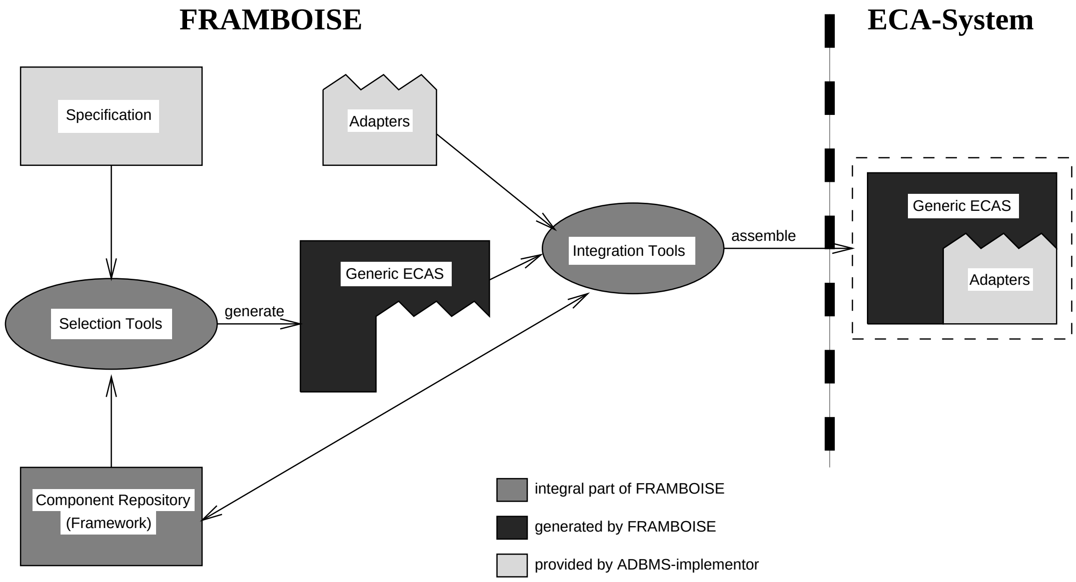


Figure 1: Construction of an ECA-System with FRAMBOISE

specification language of FRAMBOISE. The specification language provides means to describe the generic knowledge and rule execution model and characteristics of the underlying DBMS (data model, existence of proprietary event notification mechanisms etc.). Using the respective specification, the *selection tools* of FRAMBOISE choose the specific components from the *component repository* and subsequently generate a *generic ECAS*. The component repository incorporates the components themselves (i.e., the *component library*) as well as metadata containing further information about the components (e.g. interrelationships with regard to knowledge and execution models etc.). A generic ECAS is not executable, but must be enhanced with DBMS-specific adapters in order to become operational. The adapters are supplied by the ADBI. The *integration tools* of FRAMBOISE support the ADBI in the provision of the adapters and assemble them and the generic ECAS to the operational ECA-System. All ECASs have the same application programming interface (API) and consequently the same code implementing the common API. In order to assist application programmers with the use of the ECAS, FRAMBOISE provides additional tools like rule browser, rule compiler or a rule analyzer (not depicted in Figure 1). These tools are identical for all concrete systems due to the uniform API of all ECASs.

**The Component Library** The construction of the generic ECAS is *generative and specification driven.* Such approaches, which are typically application area specific, adopt a standard system structure (a *reference architecture* or *generic architecture*) and standard component interfaces. We implement the component library as an object-oriented framework[3], because such frameworks are specialized class libraries that embody a generic design [7]. Frameworks are effectively generic solutions for a specific problem domain. Thus, developing an application by means of a framework consists of providing the code that implements the specifics of the particular application which is not already addressed in the general solution of the framework itself [16]. This code, called an *ensemble* [16], is typically represented as specializations of abstract classes provided by the framework. Subsequently, the ensemble is put together with the framework in order to provide the final solution.

The FRAMBOISE framework basically incorporates the DBMS-independent components, necessary to build the generic ECASs. Furthermore it includes alternative components, e.g., different mechanisms to detect complex events (cf. section 3). The components are organized in the framework according to the reference architecture of an ECA-System. The selection tools choose the components that implement the functionality of the respective specification and subsequently provide the generic ECAS as particular configured classes. In order to realize an operational ECAS, the DBMS-specific adapters to be supplied by the ADBI are provided as an ensemble to the framework which constitutes the generic ECAS. Note that these DBMS-specific adapters could also be made available for reuse by integrating them into the component repository as an enhancement of the framework. From a software technical viewpoint, ECASs are subsets of the FRAMBOISE framework. In particular, they share the same reference architecture like the FRAMBOISE framework. Hence, issues concerning the reference architecture (cf. section 3) apply for the FRAMBOISE framework as well as for ECASs.

**The Tools** The selection and integration tools assist the ADBI during the realization of an ECAS by supporting them to operate efficiently with the framework. The tools offer functionality similar to commercially available software development environments, e.g., class browsers, management of project configuration data, etc. Moreover, they take into account domain specific characteristics of FRAMBOISE, e.g., based on the specification, the selection tools are able to generate stub classes which are to be completed in order to implement operational adapters. The integration tools subsequently lead the ADBI to the methods of these stub classes. Access to the repository and adequate presentation of the information is another service offered by the selection and the integration tools. Previously implemented

[3] An object-oriented framework is an object-oriented class hierarchy plus a built-in model of interaction which defines how the objects derived from the class hierarchy interact with each other [15].

ECASs and/or adapters can afterwards be designated by both tool sets for incorporation into a new ECAS.

## 3 The Reference Architecture

The reference architecture (Figure 2) includes the major components of an ECAS. It abstracts from object-oriented and framework-specific issues and embodies only the so-called *strategic decisions*[4]. We will subsequently explain how principal tasks of an ECAS are performed by the various components depicted in Figure 2.

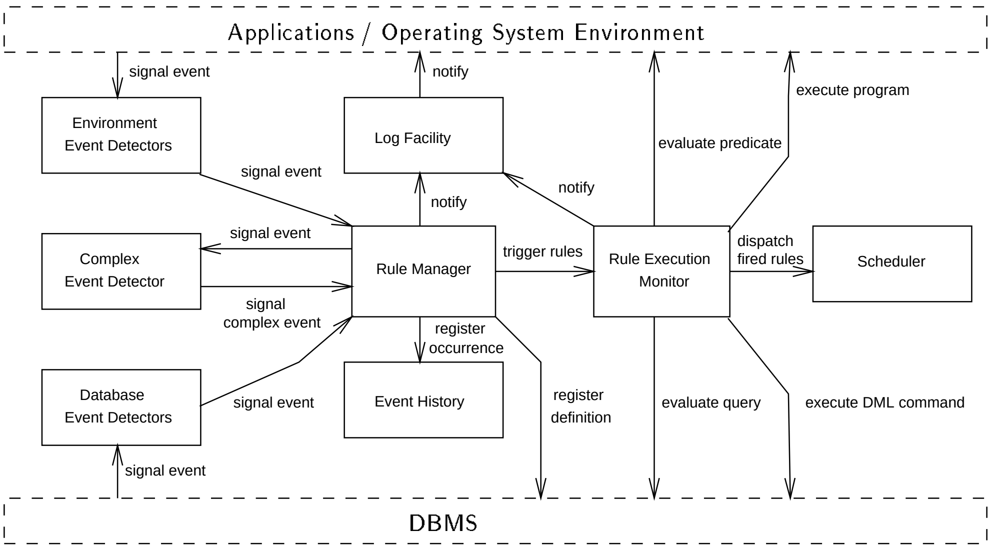


Figure 2: The Reference Architecture of an ECA-System

**Management of ECA rules:** The rule schema is represented in an ECA-System by means of *rule objects* which denote ECA rules and their constituents. In order to achieve independence from concrete DBMSs, the rule objects are realized in a DBMS-independent manner. Mechanisms to store the rule schema persistently are accessible to the ADBMS-implementors. Hence, the ADBI can provide their own adapters to store the rulebase persistently by means of the DBMS. This gives the opportunity to inspect the rulebase using the retrieval facilities of the respective DBMS [8]. Application of storage managers like [1], distributed object servers like CORBA-compliant systems or even the raw operating system facilities is also conceivable. In each case, the ADBI must provide specialized classes in the respective ensemble, which stand for the "persistent representation" of the rule schema. This specialized classes are

[4] *Strategic decisions* are defined as those which have sweeping architectural implications [3].

derived from abstracted classes implemented in the FRAMBOISE framework. Components which are completely implemented within FRAMBOISE (in particular the *rule manager*), arrange at runtime the instantiation of the rule objects and ensure the consistency with their persistent representations. The ADBMS-implementors are not bothered with this task.

**Event Detection and Signalisation:** Event detection is performed by the so-called *event detectors*. Events occur in the underlying DBMS (*database* events), in applications or the operating system and finally in the ECA-System itself (*complex* events). Database events may be raised upon data manipulation, starting or ending a transaction, sending a message to an object stored in the database etc. They are detected by different database event detectors. In the FRAMBOISE framework these detectors are typically abstract components which have to be specialized for the respective DBMS. In the best case, the database event occurrences are detected within the DBMS itself by a proprietary event notification mechanism and are propagated to the ECAS via the respective specialized event detector. Otherwise, the DBMS-specific event detectors wrap the access to the DBMS and trap the respective operation by themselves to signal an event occurrence.

Events stemming from applications or from the operating system (e.g., signalling of a particular point in time due to the state of the system clock) are detected by event detectors which are identical for all ECASs.

Complex events like conjunction or sequence of events are combinations of other defined events. They are detected by a complex event detector which is likewise identical for all different ECASs. FRAMBOISE provides different detection strategies in order to allow the ADBMS-implementor to choose the adequate one. All event occurrences are subsequently managed by the *event history* component.

**Rule Triggering and Execution:** Triggering of rules is controlled by the rule manager, which acts as a mediator between the different event detectors, the log facility, the event history and the *rule execution monitor*. The latter handles the global rule execution process upon the triggering of rules. It creates temporary *rule triggering objects*, which contain all the information needed for condition evaluation and action execution. The *scheduler* determines the rule execution order based on the conflict resolution strategy, i.e., how the complete set of rule triggering objects is processed. In order to provide different conflict resolution strategies, various schedulers are implemented. Condition evaluation and action execution are guided by the rule execution monitor, which delegates the particular operation to specific condition or action objects. They act as adaptors to the DBMS or the system environment (operational default implementations are provided for this case).

**Dispatching of Messages to Trace Facilities:** A log facility provides read-only access to the rule execution process. It collects all informations about event signalling as well as rule execution and dispatches them to the tools, which observe the rule execution behavior (e.g., trace facilities or debugging tools).

# 4 Design and Realization

At the present stage, we focus on the development of the framework and a concise mechanism to specify ECA-Systems. Frameworks are becoming increasingly common and can be regarded as a well enough established technology from a database-technology-centered view; designing and implementing a framework, however, is recognized as a complex task[9][5]. Thus, we aim at the realization of a comprehensive and technically sound implementation of the framework.

In order to achieve a high level of design and code reuse with FRAMBOISE, we rely on the design patterns presented in [9]. These patterns provide a catalog of well established design solutions for the development of reusable software elements and serve as a means to transfer specific programming expertise in framework design to our team (i.e., the framework developers). Additionally, this allows to establish a common terminology for the ADBI, in order to specify the interactions between them and the framework. Supporting this interaction will be one of the primary tasks of the integration tools to be developed at a later stage.

It is recommended to design object-oriented frameworks by making a transition from stand-alone applications to a reusable framework [16]. In particular, the identification of the *primary abstractions* (i.e., abstractions that are used by the ADBIs) of a framework should be initiated by examining existing solutions. Therefore, we are eager to incorporate our experiences made during realization of the object-oriented ADBMS SAMOS [10] in the design process of FRAMBOISE. We take over the terminology from SAMOS and derive the knowledge model and the rule definition language of FRAMBOISE from the one of SAMOS. The FRAMBOISE framework is developed incrementally, i.e., we implement a framework

[5] "If applications are hard to design, and toolkits are harder, then frameworks are hardest of all. A framework designer gambles that one architecture will work for all applications in the domain" ibid.

that applies to a specific subset of the rule definition language of SAMOS and rework it to support different features of this language, up to its full extent. In the first increment of the framework developed so far, we focused on the rule schema, event detection (including the event history) and rule triggering. These features are operational now and ensembles for the object-oriented DBMS ObjectStore and Versant are implemented and tested. An ensemble for a relational DBMS (Oracle) will be realized soon. The implementation of the DBMS-specific adapters to store the rule schema by means of the DBMS, presents itself at first glance as rather expensive. However the experiences made so far have shown that the specific steps to be performed are very simple (i.e., implementation of mere wrapper classes) and mostly repetitive. This work will typically be minimized by adequate integration tools.

The rule execution component being currently implemented offers the same rule execution semantics as the one realized in the SAMOS prototype. Afterwards, we will start the tuning process of the FRAMBOISE library by conducting first test series against the BEAST benchmarks [14]. Efficient rule execution will be of paramount importance for the overall judgment of our approach, in particular with regard to the fact that ADBMS constructed in a layered approach are considered as having potential for poor performance [18].

# 5 Discussion

The main focus of FRAMBOISE is to provide a system for the construction of DBMS-specific ECAS. This allows to offer active database mechanisms combined with any DBMS. Moreover, FRAMBOISE could be used to build ECASs which provide active functionality used independently from a DBMS, i.e., without "added" DBMS features. Thus, it is for instance conceivable to build ECASs which rely completely on CORBA-compliant services. In that case, the operational ECASs are actually middleware with functionality analogous to the *active modules* proposed in [4]. An ECAS can then be used to monitor and control several heterogeneous information sources.

The flexibility of the framework approach applied in FRAMBOISE is advantageous in order to refine operational ECASs. In particular FRAMBOISE supports:

- *Optimization of components.* A typical case is the tuning of event detectors. Efficient event detection is one of the most important prerequisites for the efficiency of active database mechanisms. Thus, it is predictable that the detectors will undergo optimization by the ADBIs. This applies especially for the detection of primitive events which relies exclusively on knowledge of the underlying DBMS or the operating system, respectively.

- *Selection between different rule execution behavior.* There are different conflict resolution strategies which are fully implemented in the FRAMBOISE framework and are provided as diverse scheduler classes.

- *Choice among several prefabricated[6] detectors.* The detection of complex events (e.g., conjunction or sequence of two events or the negation of an event; [10]) is a complex procedure where different approaches like Petri-Nets [11], trees [5, 6] and automata [12] have been examined. The ubiquitous trade-off between efficiency (i.e., detection time) and functionality (e.g., conservation of event parameters with implication to the possible strategies) prominently manifests itself in this subject. Different detection strategies are implemented, so that the adequate one can be selected.

Moreover, the specialization of an ECAS for a specific application area is also conceivable, but needs to be examined further. This concerns mainly the interaction between active mechanisms implemented with FRAMBOISE and the respective active applications (e.g., the kind of information to be exchanged between the rule execution mechanisms and the respective applications). It is furthermore desirable to provide specialized adaptors in order incorporate domain specific activities directly in the condition evaluation and action execution process (e.g., the use of a specialized language developed for the specific application area).

# 6 Conclusion

In this paper, we have presented the rationale for providing active database mechanisms as construction-system-based "add-on" capabilities of traditional DBMSs. We have argued to base such a construction system, called FRAMBOISE, on an object-oriented framework. Subsequently, we presented the reference architecture of FRAMBOISE and sketched our further proceeding. We are about to implement the above ideas and test their effectiveness and efficiency.

# References

[1] A. Biliris and E. Panagos. A high performance configural storage manager. In *Proc. 11th Intl. Conference on Data Engineering*, Tapei, Taiwan, 1995.

[6] In this context "prefabricated" means that such parts are operational components of FRAMBOISE and ready for use by the ADBMS-implementor

[2] J. A. Blakeley. The Masses through OLE DB. In *Proc. SIGMOD*, pages 161–172, 1996.

[3] G. Booch. *Object-Oriented Analysis and Design.* The Benjamin/Cummings Publishing Company, Inc, 2 edition, 1994.

[4] O. Boulcema, J. Dalrymple, M. Doherty, J.C. Franchitti, R. Hull, R. King, and G. Zhou. Incorporating active and multi database services into an osa-compliant introperability toolkit. *The Collected Arcadia Papers, Second Edition*, 1995.

[5] A.P. Buchmann, J. Zimmermann, J.A. Blakeley, and D.L. Wells. Building an integrated active oodbms: Requirements, architecture and design decisions. In *Proc. of the 11nth intl Conference on Data Engineering*, pages 117–128, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proc. of the 20th Intl. Conference on Very Large Databases*, Santiago, Chile, September 1994.

[7] S. Cotter and M. Potel. *Inside Taligent Technology.* Addison-Wesley, 1995.

[8] K.R. Dittrich, S. Gatziu, and A. Geppert. The active database management system manifesto. In T. Sellis, editor, *Proc. of 2nd Intl Workshop on Rules in Database Systems*, Athens, Greece, September 1995. Springer.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison Wesley professional computing series. Addison Wesley, 1st edition, 1995.

[10] S. Gatziu. *Events in an Active Object-Oriented Database System.* PhD thesis, University of Zurich, 1994.

[11] S. Gatziu and K.R. Dittrich. Detecting composite events in an active database system using petri nets. In J. Widom and S. Chakravarty, editors, *Proc. 4th Intl. Workshop on Research Issues in Data Engineering: Active Database Systems*, pages 2–9, Houston TX, 1994. IEEE Computer Society Press.

[12] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases. In *Proc. of the 18th Intl. Conference on Very Large Databases*, Vancouver, August 1992.

[13] A. Geppert and K.R. Dittrich. Constructing the Next 100 Database Management Systems: Like the Handyman or like the Engineer? *SIGMOD Record*, 23:27–33, 1994.

[14] A. Geppert, S. Gatziu, and K.R. Dittrich. A designer's benchmark for active database management systems: OO7 meets the beast. In T. Sellis, editor, *Proc. of 2nd Intl Workshop on Rules in Database Systems*, Athens, Greece, 1995. Springer.

[15] T. Lewis, L. Rosenstein, W. Pree, A. Weinand, E. Gamma, P. Calder, G. Andert, J. Vlissides, and K. Schmucker. *Object Oriented Application Frameworks.* Manning Publications Co., 1995.

[16] Inc. Taligent, editor. *The Power of Frameworks.* Addison-Wesley, 1995.

[17] D. Vaskevitch. Database in crisis and transition: A technical agenda for the year 2001. In *Proc. SIGMOD*, Minneapolis, USA, May 1994.

[18] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Data Management Systems. Morgan-Kaufman, 1996.