Performance Evaluation of Active Database Management Systems Using the BEAST Benchmark

Andreas Geppert
University of Zurich[1]

Mikael Berndtsson
University of Skövde

Daniel Lieuwen
Lucent Technologies/
Bell Labs Innovations

Jürgen Zimmermann
University of Darmstadt

Email: geppert@ifi.unizh.ch, spiff@ida.his.se, lieuwen@allegra.att.com, zim@dvs1.informatik.th-darmstadt.de

Technical Report 96.01
Department of Computer Science
University of Zurich
February 1996

Abstract

This paper presents the first comparative performance study of object-oriented active database management systems by using the BEAST benchmark. BEAST stresses the performance-critical components of active systems: event detection, event composition, rule retrieval, and rule firing. For event detection both method invocation events and transactional events are taken into account; this also shows some performance contributions of the passive part of an ADBMS. Four systems, namely ACOOD, Ode, REACH, and SAMOS, have passed the benchmark tests of BEAST. The interpretation of the performance measurements shows several achievements in the area of active database technology, but also indicates tradeoffs (e.g. between performance and functionality). Finally, it helps to identify possible optimizations and open issues in designing and implementing active database systems.

Keywords: active database systems, database benchmarks

1 Introduction

In recent years, active database management systems (ADBMSs) [e.g., 28, 6] have become a hot topic of database research, and restricted ADBMS-functionality is already offered by some commercial systems [e.g., 24, 25]. An ADBMS implements “reactive behavior” since it is able to detect situations in the database and beyond and to perform corresponding actions specified by the user and/or DB-administrator. Applications using reactive behavior do not require “polling” techniques in order to detect relevant situations. Additionally, an ADBMS covers more application semantics than a passive DBMS because the implementation of situation detection and subsequent reactions is migrated from the application programs into the ADBMS.

As for any system, ADBMSs should implement their functionality *efficiently*. Indeed, performance issues have recently been considered as one of the most important topics to be addressed to meet the requirements of applications and potential users [27]. Furthermore, performance aspects also play a crucial role from a system point of view:

1. Contact author’s address: Institut für Informatik, Universität Zürich, Winterthurerstr. 190, CH-8057 Zurich, Switzerland. Fax: +41-1-363 0035

- ADBMS researchers have developed different techniques for the ADBMS tasks such as composite event detection [e.g., 9, 15, 19]; thus, it is interesting to compare the performance of these approaches.
- Different architectural approaches have been developed and need to be compared. For instance, in the past there were intense discussions about the advantages of integrated architectures and the restrictions of layered architectures [7].

As for now, figures describing the performance even of single ADBMSs are only scarcely available [16, 22]. In [16] a first approach was made to specify a benchmark for ADBMSs, and [29] gives an overview about the requirements a benchmark for ADBMSs must fulfill.

In this paper we describe the application of the BEAST benchmark [16] (**BE**nchmark for **A**ctive database **S**ys**T**ems) to four object-oriented ADBMSs (ACOOD [3], Ode [1], REACH [7], and SAMOS [14]). We interpret the benchmark results obtained for each system and make some general conclusions. The interpretations not only show several achievements in recent ADBMS-research but also illustrate performance drawbacks and open problems with respect to performance. Moreover, BEAST verifies some assumptions made elsewhere on the performance of ADBMSs, while rejecting others.

BEAST focusses on basic ADBMS-tasks such as event detection, rule retrieval, and rule execution. BEAST is intended for primarily testing the active functionality of DBMSs, since appropriate benchmarks for passive DBMSs have already been developed [e.g., 8, 18]. Furthermore, we concentrate on *object-oriented* ADBMSs, since — although we focus on the active part — the underlying data model has some influence on ADBMS performance.

The next section gives a short introduction of ADBMSs and the tested systems. Section 3 describes the benchmark, and section 4 presents the results. Section 5 concludes the paper.

2 Active Database Management Systems

In this section, we give a short introduction of ADBMSs. Details can be found in [28]. We then briefly describe the most important features of the systems tested with the BEAST benchmark.

2.1 Overview

An ADBMS is a DBMS that supports the specification and implementation of reactive behavior in addition to standard database functionality. Most ADBMSs support event-condition-action rules (ECA-rules) [11] for defining reactive behavior. An event is either an explicitly specified point in time or a description of a “happening of interest” to the user (that is detectable by the database system). After an event is detected, the corresponding rule will be fired. Events can be either *primitive* (e.g., a method invocation, a transaction begin or commit, a time

event, an abstract event[2]) or *composite* (e.g., conjunction, sequence, disjunction, negation, repeated occurrence). The condition is either a boolean function or a database query. If the condition evaluates to true (or returns a non-empty result), the action is executed. An action is typically written in the data manipulation language (DML) of the ADBMS.

The *execution model* of an ADBMS determines how condition evaluations and action executions are performed in terms of the transaction model. The *coupling modes* of a rule specify when the condition and action parts of a rule are executed with respect to the transaction that triggered the event. Typical coupling modes are *immediate* (directly after the event has been detected), *deferred* (at the end of the triggering transaction, but before commit), or *decoupled* (in a separate, independent transaction). We assume that the coupling modes for conditions relate condition evaluation to the triggering event, and that the coupling modes for actions relate action execution to condition evaluation. Finally, the execution model also defines how to process multiple rules that are triggered by the same event. One possibility is to let the user specify (partial) orders, e.g., by means of *rule priorities*.

2.2 The Tested Prototypes

We have tested four ADBMS prototypes whose major features are briefly described below:

- ACOOD [3, 12] is built on top of the commercial OODBMS ONTOS DB 3.0,
- Ode [1, 23] exists in two variants: a disk-based version built on EOS [4] and a main-memory version built on Dali [20] (the disk-based one was benchmarked),
- REACH [7, 5] is built with the Texas Instruments' Open OODB [26] Release 0.2.1a, and
- SAMOS [14, 17] uses the commercial OODBMS ObjectStore as a platform.

All the systems support method events. All except REACH also support abstract events and only ACOOD does not offer transaction events. The systems support different sets of composite event constructors, however those required for the BEAST tests can be expressed in the rule definition language of each system. The systems use four different techniques for composite event detection: arrays (ACOOD, [12]), extended finite state machines (Ode, [19, 23]), syntax trees (REACH), and Petri Nets (SAMOS, [15]). The provided consumption modes [9] are `chronicle` (Ode, REACH, SAMOS) and `recent` (ACOOD, REACH).

Conditions in ACOOD are expressed as ONTOS SQL queries while REACH and SAMOS allow arbitrary boolean functions as conditions. In Ode *masks* can be defined, which are conditions that must be evaluated to determine if a (sub)event of an event has occured or not. Triggers must be activated to have any effect. There can be many activations per trigger.

2. *Abstract events* are events that are not detected by the ADBMS, but that have to be signalled explicitly by the application or the user.

Actions in each of the systems are arbitrary statements in the underlying database programming language (C++, except in Ode which is based on the C++-extension O++). REACH and SAMOS can pass event parameters to conditions and rules; in Ode parameters used in masks/ actions are passed to the associated trigger at activation time.

Each of the systems implements further functionality that is irrelevant for BEAST.

3 BEAST: A Benchmark for ADBMSs

In this section, we first identify design decisions for the BEAST benchmark (see [21] for how to design a benchmark). We then describe BEAST in detail.

Style of tests. The intention of BEAST is to test the basic functionality of ADBMSs and to determine performance drawbacks of ADBMS-designs and -implementations. It does **not** propose a typical application and test the performance of ADBMSs for such an application (after all, what is the "typical" application of an ADBMS?). We are thus measuring the performance of ADBMSs on a micro level from a designer's perspective.

Influence of passive components. ADBMSs use the functionality of passive DBMSs. They need services from the passive part, such as persistence, transactions, and maybe query processing. BEAST thus tests the entire active DBMS, and the performance of passive parts typically will influence ADBMS-performance. We do not test ADBMSs at a finer-grained level (e.g. by turning off locking/logging) because the required functionality is not available in all systems. Consequently, an ADBMS that uses a slow platform or does not exploit the capabilities of the underlying system in an optimal way will incur a performance penalty. As the various measures nevertheless have shown, BEAST makes it possible to identify performance bottlenecks of the active part by comparing the different tests performed for a specific system.

Selection of metrics. Our major metric is CPU-time. BEAST tests invoke active behavior, which always performs several phases such as rule execution. CPU-time is then defined as the time interval that an operating system process spends to detect events and execute rules (exceptions to this definition are necessary for coupling modes other than immediate; see below).

3.1 Benchmark Design

BEAST is based on the 007 benchmark [8]. It uses the schema of 007 as well as the corresponding databases (i.e., programs to create and fill databases). One reason for reusing parts of 007 is to easily obtain a schema and database. Moreover, for a given object-oriented ADBMS, BEAST and 007 together measure the performance of both the active and the passive parts of a system, respectively.

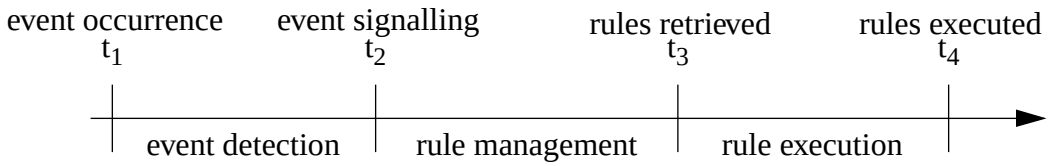

Figure 1. Phases of Active Behavior

BEAST considers three components where performance is crucial:

- event detection,
- rule management, and
- rule execution.

We have selected these components since they implement the three phases that comprise active behavior (see Fig. 1). They are thus contained in most ADBMS-architectures [e.g., 7, 9, 17].

After an event occurs, it must be *detected*, i.e., ADBMS components must recognize (or be notified) that the event has happened. At the end of the event detection phase, the event is *signalled*.[3] The second phase (rule management) starts as soon as the event has been signalled and determines whether (and which) rules must be executed. Internal information that links event descriptions with rule definitions must be taken into account. In the simple case of `immediate` coupling, rule management is directly followed by the rule execution phase (starting at t_3 in Fig. 1). In this phase, the triggered rules are executed. Thus, performance measurements of an ADBMS must consider each of the three phases discussed above.

Event detection is realized by the components that recognize the occurrence of specific events of interest. Two subtasks of event detection affect performance: detection of primitive and composite events. Primitive event detection and event composition can be implemented in several ways that may have different performance characteristics (see section 2.2).

Another task in this respect is *management of event occurrences*. If an ADBMS can compose events out of components that have occurred within different transactions, then the component occurrences must be made persistent at least until they have been consumed for composite events. Thus, the ADBMS must manage persistent component occurrences efficiently, and the retrieval of these components upon event composition is performance critical.

Rule management also influences the performance of an ADBMS. Some systems store event descriptions and rule definitions as objects in the database. Since these systems must retrieve information on rules after the signalling of an event, efficient identification and retrieval of corresponding rules is crucial for performance. Second, it is interesting to compare the performance of such systems with others that compile rules hard-wired into classes.

3. In general the precise point in time when an event occurred is not known. However, in the BEAST tests, we enforce event occurrence and thus know this point in time.

Rule execution refers to the identification of condition and action parts that have to be executed after event occurrences as well as the execution of these parts. In particular, it is interesting how efficiently the various coupling modes are implemented and how efficiently multiple rules triggered by the same event can be executed.

BEAST defines several tests for each component. Thus, the result of running BEAST is a collection of figures instead of a single figure for each ADBMS (much like OO7). Note that we cannot test the performance of each component directly, due to lacking access to internal interfaces of an ADBMS. Therefore, most BEAST tests specify one or more rules that are triggered when executing the test, i.e., the test actually causes the event occurrence. In order to stress the performance of single phases, we keep all other phases as small as possible. For instance, a rule testing the performance of event detection simply defines the condition to be `false`, so that condition evaluation is cheap and no action is executed. Additionally there is only one rule triggered by such an event in order to minimize the overhead of rule management.

We elaborate on each group of tests subsequently. We describe each test and show the corresponding rule(s) in pseudo-code. Note that the tests are not always enumerated consecutively, since some of the ones originally proposed [16] have been omitted in this paper (e.g., because some of the tested systems do not support the functionality required by these tests).

3.1.1 Tests for Event Detection

Event detection tests focus on the time it takes to detect primitive or composite events.

3.1.1.1 Tests for Primitive Event Detection

Two of the BEAST tests refer to primitive detection (Fig. 2):

1. detection of method invocation (ED-02),
2. detection of transaction events (ED-03).

```
RULE   ED-02
ON     before AtomicPart->DoNothing                    // method event
IF     false
DO     ...

Rule   ED-03
ON     before commit(ED03_TX)                          // commit event
IF     false
DO     ...
```

Figure 2: Rules for primitive event detection tests

We illustrate the execution of tests with the test ED-02. First, the actual time is obtained, and then the event is forced to occur multiple times (in this case, a method is invoked). Note that in this way we know the point in time of event occurrence. The ADBMS subsequently detects the

event, determines attached rules, and executes them. It then returns control to the test program. Finally, the test program again records the time and computes the consumed CPU time.

The tests ED-02 and ED-03 measure detection of single events. The corresponding rules for all tests have a false condition and an empty action in order to restrict the measured time to event detection, as far as possible. Coupling modes for actions and conditions are `immediate`. Another possible kind of primitive event would be *time events*; however, there is no way to measure the cost of detecting such events unless, at a minimum, one has access to the database internals and can modify them.

3.1.1.2 Tests for Composite Event Detection

Composite event detection typically starts after a (primitive or other composite) event has been detected. The event detector then checks whether the detected event participates in a composite event. This is generally done in a stepwise manner, e.g., by means of syntax trees [9], automata [19], or Petri nets [15]. Of course, the different approaches may have different performance characteristics and therefore need to be compared with respect to efficiency. This is accomplished through tests ED-06 through ED-11 (Fig. 3).

In order to stress the time needed for composite event detection, we use abstract events in the definitions of composite events wherever possible. Using abstract events enables more accurate measurements, since only the time for event signalling is required and primitive event *detection* is not necessary. In order to measure the entire event composition, the tests raise the component events directly one after the other.

BEAST contains six tests for the detection of composite events:

1. detection of a sequence of primitive events (ED-06)
2. detection of the non-occurrence of an event within a transaction (negative event, ED-07),
3. detection of the repeated occurrence of a primitive event (ED-08),
4. detection of a sequence of events that are in turn composite (ED-09),
5. detection of a conjunction of method events occurring for the same object (ED-10),
6. detection of a conjunction of events raised within the same transaction (ED-11).

Since we are interested in the time for event detection, conditions, actions, and coupling modes are kept as simple as possible. These are parts are equivalent to those in Fig. 2 and are thus ommitted in Fig. 3. Tests ED-06 through ED-08 measure event detection for common composite event constructors. Test ED-09 considers one specific constructor applied to events that are in turn composite. Finally, ED-10 and ED-11 measure the performance of event detection when the events of interest are restricted by event parameters.

```
RULE   ED-06
ON     EvED-061 ; EvED-062                    // composite event: sequence

RULE   ED-07                 // negative event within a named transaction
ON     ! EvED-07 within [begin(ED07_TX), commit(ED07_TX)]

RULE   ED-08
ON     times (EvED-081, 10)                   // EvED-081 occurs ten times

RULE   ED-09      // times event, then a disjunction, then abstract event
ON     times (EvED-091, 3) ; (EvED-092 | EvED-093) ; EvED-094

RULE   ED-10         // sequence of method events with identical receivers
ON     Module->DoNothing ; Module->setDate : same object

RULE ED-11        // conjunction of method events occuring in same trans.
ON     AtomicPart->setX & AtomicPart->setY : same transaction
```

Figure 3: Rules for composite event detection tests

3.1.2 Tests for Rule Management

The second group of tests considers *rule management*. It is based on the observation that an ADBMS has to store and retrieve the definition and implementation of rules, be it in the database, as external code linked to the code of the ADBMS, or as interpreted code. Apparently, the time it takes to retrieve rules influences ADBMS performance. Rule management tests measure rule retrieval time, but they do not consider *rule definition* and *rule storage*. These services are executed rather seldomly, and thus their efficient implementation is less important.

The test RM-1 (Fig. 4) raises an abstract event, evaluates a condition to `false`, and therefore does not execute any action. The three parts are kept such simple in order to restrict the measured time to the rule retrieval time as far as possible.

The second test specified for rule retrieval does not specify an ECA-rule. Instead, the purpose of this test is to retrieve information on event definitions and associated rules from the rule catalog using the ADBMS's query language (provided that this kind of retrieval is supported at all). If the ADBMS stores information on events and rules in the database, then this test helps finding out how much time the ADBMS needs for rule retrieval at runtime. In test RM-02 an event description is retrieved based on its identifier and subsequently all rules associated with this event description are retrieved as well.

```
RULE   RM-01
ON     EvRM-01                                  // abstract event
IF     false
DO     ...
```

Figure 4: Test for rule management

3.1.3 Tests for Rule Execution

The tests for rule execution are separated into two groups: one for the execution of single rules, and one for the execution of multiple rules. The first group of tests (RE-01 through RE-03) determines how quickly rules can be executed. The execution of a single rule consists of loading the code for conditions and actions and of processing or interpreting these code fragments. Different approaches for linking and processing condition and action parts can be compared by means of the tests in this group. Different strategies can also be applied for executing multiple rules all triggered by the same event (e.g., sequential or concurrent execution). The performance characteristics of these approaches are tested by the second subgroup.

For the execution of single rules, we consider three rules with different coupling modes. An abstract event is used, the condition is always true, and the action is a `print` command in rules RE-01, -02, and -03. The coupling mode of the condition is always `immediate`. The coupling modes of the actions are `immediate` (RE-01), `deferred` (RE-02), and `decoupled` (RE-03). The intention of these tests is to measure the overhead needed for storing the fact that the action still needs to be executed at the end of the transaction (`deferred`), as well as the overhead necessary to start a new transaction in the `decoupled` mode. In order to stress these aspects of rule execution, we use an abstract event in order to avoid event detection, and use a simple `true` condition and a simple action. Note that the performance of condition evaluation and action execution is not of interest, because it is determined by the "passive" part of the DBMS.

The test RE-04 (Fig. 5) considers four rules all triggered by the same event. Conditions and actions are more complex than in the previous tests, in order to observe the effects of optimizing the condition evaluation and of concurrency. All RE-04 rules have the same condition. Hence, an ADBMS that recognizes equality of conditions (e.g., if it is able to optimize sets of conditions) will perform better than a non-optimizing ADBMS. All rules have the coupling modes `(immediate, immediate)`. No ordering is defined for the four rules. An ADBMS that is able to process conditions and actions in parallel or at least concurrently will thus perform better in this test.

```
RULE      RE-04a
ON        Document->DoNothing                              // method event
IF        oid->searchString("I am") > 0              // oid is the receiver
DO        print("Document contains 'I am'");

RULE      RE-04b  ...                  //event and condition as in RE-04a
DO        oid->setAuthor();

RULE      RE-04c  ...                  //event and condition as in RE-04a
DO        oid->setDate();
```

```
RULE      RE-04d ...                    //event and condition as in RE-04a
DO        oid->replaceText("I am","This is");
```

Figure 5: Rules for rule execution tests

3.2 Factors and Modes

A crucial step when designing a benchmark is the proper identification of *factors* [21], i.e., parameters that influence performance measurements. Several parameters of a database can have an impact on the performance of an ADBMS. In addition to the database parameters relevant for benchmarking a passive DBMS (e.g., buffer size, page size, number of instances stored in the database), these include:

- the number of defined events,
- the number of defined rules,
- the number of initial components raised for composite events.

In the ideal case, the time to detect events is constant, i.e., independent of the number of defined events. However, especially for composite events, it may be the case that the event detection process for single events slows down as more events are added to the system. Furthermore, an ADBMS needs to store and retrieve internal information on event definitions during (or after) event detection. Apparently, a large number of event definitions can increase the time needed to retrieve event information. It is thus interesting to investigate how large CPU-times are when the number of events increases. This number is therefore included as a factor. In general, about 50% of the events are defined as composite events.

Furthermore, the total number of rules defined by a concrete database is relevant for performance. Recall that rule information has to be retrieved before rule execution. While a small number of rules can be entirely loaded into main memory without problems when the ADBMS starts execution, this is no longer possible if the rulebase is large. In the latter case, rules must be selectively loaded upon rule execution. It is therefore an important question how efficiently an ADBMS can handle large sets of rules, and how the system behaves when the number of rules grows larger.

Ultimately, the performance of composite event detectors can depend on the previous event history. Specifically, we expect that the performance of event composition depends on the number of events that are candidate components for composite event detection. For the tests ED-06 and ED-09 through ED-11, the number of component events which are used to initialize the composite event detector is thus a parameter.

For the three factors, we choose four possible values for an empty, a small, a medium, and a large (dummy-) rulebase (see Table 1). Tests for larger rulebases are simple to produce, since the values of all factors can be specified as parameters of the rulebase creation program. Many

factor	rulebase size			
	empty	small	medium	large
#events	0	50	250	500
#rules	0	50	250	500
# of component event occurrences	0	25	50	100

Table 1: Parameter Values for Different Rulebase Sizes

rules and events will actually not be used by the benchmark, i.e., their execution is not measured. However, they are important in order to increase the load of the ADBMS as well as the data/rulebase size. These "dummies" therefore indicate whether the ADBMS is able to handle large sets of rules with a performance comparable to small numbers of rules.

4 Benchmark Results

In this section, we present the results obtained by running BEAST on each of the four systems.

In order to run the benchmark for a concrete ADBMS, the 007 schema must be defined for the tested system and OO7 databases must be created. The next step consists of specifying and compiling the ECA-rules for the system. In the final step, the desired tests are executed. Each system has been tested with several dozens of test iterations. In each iteration, each test was run once; in each test, the corresponding rule(s) was (were) triggered ten times.

Each test computes the CPU-time the operating system process has spent for the test execution (due to the fact that this process is subject to operating system scheduling, process-specific CPU-time can be a fraction of the total elapsed time). All the results are given in milliseconds (ms). In order to not flood the text with numbers, we only give average CPU-times and refer to standard deviations only if they are exceptional. A complete description of all tests series including min/max values, standard deviation, and 90% confidence intervals [21] can be found in the appendix. Below we present the results and then discuss them in section 4.5.

4.1 Results for ACOOD

The tests for ACOOD (Table 2) have been executed on a SUN SPARCServer10/51 under SUNOS 4.1.3. ACOOD scales well, i.e. the CPU-time is almost constant and independent of the rulebase size. Primitive event detection in ACOOD is fast since event parameters are not passed to conditions and actions. Composite event detection is fast because an array technique is used which is not powerful enough to detect events as in ED10 and ED11, but is efficient for the detectable ones. Furthermore, the `recent` event consumption is used that is not sensitive to the event history size. Rule execution is efficient because rules are indexed by events.

	Rulebase Size			
Test	empty	small	medium	large
ED 2	244	246	261	261
ED 6	417	423	436	450
ED 7	151	144	145	156
ED 8	1015	1037	1048	1038
RM 1	231	240	248	249
RM 2	53	53	56	58
RE 1	259	260	261	252
RE 4	415	422	426	430

Table 2: BEAST Results for ACOOD

4.2 Results for Ode

Ode has been tested on a SUN-SparcServer 4/690 under SUNOS 4.1.3. For many tests, Ode was the fastest system (Table 3); it also scales well for growing rulebases.

An Ode trigger must be activated or it will never fire. If the corresponding event occurs, the event mask is evaluated, and if it evaluates to true then the trigger is fired. Given that Ode identifies both complex and primitive events using the same extended finite state machine mechanism [23], it takes exactly the same amount of time to detect that an event of interest has occured whether the event is simple or complex unless masks (conditions) must be evaluated. If a mask involves an expensive computation or if several masks must be evaluated[4], identifying a composite event will take proportionately more time. However, in the experiments, the masks were simple enough that identifying the occurence of either a primitive or a complex event of interest to a trigger activation took roughly the same amount of time.

Initially, we considered the event mask as the analogon to conditions in ECA-rules, and consequently specified them in such a way that they always evaluated to false for event detection tests. However, trigger activations are not deleted if this mask evaluates to false because they have never fired (and never will fire). Thus, the number of trigger activations in the system grows over time, and each activation must be alerted when an event is posted to its corresponding object. This is the reason for the increase in the measured times (e.g., for ED-08), and also for RM-01 being slower than RE-01.

The values for some tests also prove that argument: for instance, execution times for ED-08 in the large rulebase start with 410 ms and in the final tests end up with 1630 ms. In this case,

4. If a composite event involves several masks, more than one mask may need to be evaluated in response to a single basic event occurrence.

	Rulebase Size			
Test	empty	small	medium	large
ED 2	33	32	29	33
ED 3	221	230	223	234
ED 6	174	189	201	249
ED 7	158	181	206	222
ED 8	509	627	750	975
ED 9	269	256	275	276
ED 10	196	203	208	263
ED 11	831	875	951	985
RM 1	99	108	105	105
RE 1	32	33	37	43
RE 2	48	42	42	46
RE 3	55	58	64	66
RE 4	866	882	895	917

Table 3: BEAST Results for Ode

the standard deviation is 297. Alternatively, if the event mask always evaluates to true, ED-08 has an average execution time of 429 ms and a standard deviation of 49. The same effect can be observed for other tests as well.

4.3 Results for REACH

REACH has been tested on a SUN-SPARC 10/512 under Solaris 2.4. The results in Table 4 show two outstanding negative results for the tests ED3 and ED7 which contain the detection of commit events. Since Open OODB flushes the whole buffer during commit — even for read-only transactions — REACH's performance is negatively impacted by the underlying platform. REACH also updates the event history during commit; this update deteriorates performance for larger rulebases. Thus, the platform and commit processing must be improved.

Most other results are quite encouraging because REACH uses event detectors which are specialized for exactly one event type. Additionally Open OODB's compiler was modified to wrap each method so that each invocation is considered as a potential event independently whether the receiver object is persistent or transient. REACH is able to pass all arguments of a method call to the condition and action of a rule. The overhead for this flexibility can be neglected for method events (ED2), sequence and conjunction events (ED6, ED10, ED11) and firing the rules in the tests RE1 - RE4 for all rulebase sizes.

	Rulebase Size			
Test	empty	small	medium	large
ED 2	105	100	55	331
ED 3	11632	282732	805328	4180290
ED 6	147	144	150	710
ED 7	12076	3894042	1946300	11526000
ED 8	5004	6605	6608	7717
ED 9	3231	3225	2891	3726
ED 10	183	185	202	890
ED 11	1616	1588	1715	1747
RE 1	62	63	58	239
RE 2	34	35	51	229
RE 3	60	61	60	325
RE 4	252	245	161	762

Table 4: BEAST Results for REACH

Detecting the repeating events in ED8 and ED9 is slower. In this case, REACH aggregates the methods' arguments into the parameters of the composite event, and therefore performance deteriorates.

4.4 Results for SAMOS

SAMOS has been tested on a SUN-SparcServer 4/690 under SUNOS 4.1.3. For some tests, SAMOS (Table 5) is approximately ten times slower than Ode, while others are comparable. The major reasons for the high execution times in SAMOS are the complexity of the system, the additional functionality it has, and the way event detection is implemented.

SAMOS scales quite well for growing rulebases as far as primitive event detection and rule execution is concerned. This is due to indexing and clustering event descriptions and rule information. SAMOS scales worse for composite event detection, since (1) lots of objects forming the Petri Net used for composite event detection are stored on disk, and (2) no clustering is applied to those objects.

Furthermore, SAMOS (i.e., its composite event detector) is sensitive to the number of existing component event occurrences. In ED-11 for the large rulebase, e.g., 100 component events are raised before the tests actually start. These events are stored persistently and are considered for event composition during each test ED-11. Without these (useless) component events, the average execution time of ED-11 is 2023 ms for the large rulebase.

	Rulebase Size			
Test	empty	small	medium	large
ED 2	445	473	527	522
ED 3	472	525	545	570
ED 6	2059	3681	5165	8124
ED 7	2236	2363	2463	2473
ED 8	6269	7015	7433	7098
ED 9	5549	6592	7347	8378
ED 10	1816	3514	5046	7890
ED 11	1861	4272	6440	10536
RM 1	425	473	501	505
RM 2	43	41	42	45
RE 1	460	499	542	542
RE 2	448	495	527	539
RE 3	419	459	473	505
RE 4	911	967	1025	987

Table 5: BEAST results for SAMOS

4.5 Discussion of Results

We consider it an achievement that several object-oriented ADBMS-prototypes are now available. As the tests show, they perform some of their tasks in a timely manner (e.g., action execution does not seem to cause major performance problems).

Now, we will generalize the performance results. We are thereby primarily considering runtime performance while neglecting design decisions related to functionality or compile time performance. It is apparent that the four systems cannot be directly compared in such a way that the “fastest” system is determined, since different platforms have been used (only Ode and SAMOS have been executed on the same machine). However, we feel that a comparison is justified when the difference between two systems is (say) a factor of 10. Furthermore, it is sound to compare the tested systems with respect to the following questions:

- how well do they scale for growing rulebases?
- what is the relationship between the various tests (i.e., is composite event detection much more expensive than primitive event detection) for one system.

Trade-offs. The first conclusion to be drawn from the test results is the trade-off between functionality and performance. ACOOD and Ode offer less functionality than SAMOS and REACH in that they do not support event parameters being passed to conditions and actions. Primitive event detection and event composition are therefore potentially faster in ACOOD and

Ode. Second, ACOOD does not support explicit event restrictions (such as `same transaction`), which are supported in REACH and SAMOS. Upon event composition, ACOOD is thus potentially faster since it can take *any* event occurrence for composition, but does not need to select those event occurrences that fulfill the event restriction.

One the other hand, event parameters and event restrictions are considered useful constructs. If they are thus desired, then one has either to build them into the language (as is done in Ode for the `same object` restriction), or to accept the runtime cost.

Another trade-off is that between compile-time and runtime performance. For instance, if class-internal rules are supported, compile-time performance is worse, but runtime performance is improved.

Event and Rule Management. Those systems that store event descriptions and occurrences as well as rule definitions as objects in the rulebase are likely to suffer a performance penalty (this was previously hypothesized in [2]). Representing events, rules, or the internal states of event detectors as separate objects implies additional read or write accesses to the rule base and event history, which are not necessary when all this information is included in class definitions. In the latter case, rule and event definitions are already available with the class definition. This is one of the reasons for Ode being the fastest system for many tests, since while it stores the internal state of the extended finite state machines (28 bytes per trigger activation) in the database, it does not store events or rules there.

Event Detection. The first observation with respect to event detection refers to whether events are detected globally or locally to objects. In between these extremes, several event detectors exist which are responsible for certain sets of events. In the first approach, one central event detector is notified about all sorts of events from other ADBMS components (in ACOOD and SAMOS). The event detector then needs to retrieve the event description from the rulebase and to determine event parameters (in SAMOS). Furthermore, in SAMOS the composite event detector has to reconstruct the Petri Net parts it needs, which in turn are represented as objects and spread all over the database. In the local approach, most of the information is already available, since it is kept local to objects (in Ode) or to the various dedicated event detectors (in REACH). This explains why Ode (1) detects composite events faster and (2) scales well for growing rulebases. It also explains why for some tests the performance of REACH lies in between those of Ode and SAMOS.

Event History Management. For some tests, the number of initially raised component events is a factor, i.e., the event history is not empty when the tests start. Especially if event parameters are required for subsequent rule execution, then the event history must be maintained, ei-

ther explicitly or implicitly in the state of the event detector(s). Two observations are apparent with respect to event history management:

- The `recent` consumption mode seems to be more efficient, since upon event composition the entire event history might have to be scanned in `chronicle` consumption mode. This is the reason why ACOOD (which uses the `recent` mode) — unlike SAMOS — is not sensible to the size of the event history.
- If the `chronicle` consumption mode is used, then garbage collection of old event occurrences is a crucial task. For instance, in ED11, the initially raised component event occurrences are of no use, since a `same transaction` restriction is specified. Garbage collection would discard these occurrences even before the tests actually start and thus would make SAMOS five times faster for some tests (see the appendix).

Condition Evaluation. The current prototypes do not perform any kind of optimization or pre-analysis of conditions. A very basic approach might be to perform pre-analysis, during which constant expressions would be detected (none of the systems recognized the constant, false condition or mask in event detection tests). Condition evaluation optimization might also be improved, since none of the systems recognized that the same condition was used in each of the four associated rules for RE-04. Thus, optimizing *sets of conditions* together or incrementally might further improve performance of rule execution (note that this has already been investigated for relational and production-rule systems [e.g., 13]).

Observations on Architectural Styles. It is not generally justified based on our results to conclude that integrated architectures have better performance than layered architectures. Actually, even integrated architectures use some kind of lower-level platform (be it a toolkit like Open OODB), and the performance of this platform is also decisive in addition to the chosen implementation techniques. Nevertheless, in integrated architectures there is a higher degree of freedom when choosing techniques (e.g., for event detection) — for instance, some of the techniques used in Ode are not applicable in a layered system.

5 Conclusion and Future Work

Four ADBMSs have been benchmarked. This benchmarking was not possible when the work on BEAST started in 1994, since far fewer systems were operational back then. The systems we have tested are quite powerful and efficient for certain tasks.

Furthermore, the tests have also helped stabilize each of the systems, since implementing a pre-defined benchmark determined several bugs and limitations, and also helped understanding the performance of ADBMSs. Concretely, we learned about the performance characteristics of

event detection techniques (using a centralized, single event detector vs. usage of many event detectors dedicated to objects or event descriptions) as well as the performance of composite event detectors. We also better understand in which cases factors such as the rulebase size or the size of the event history influence performance. The remaining performance problems can be subdivided into two classes:

- trade-offs between performance and functionality in some aspects, where either functionality must be reduced or its cost be accepted (e.g., class-independent rules),
- open problems still to be addressed (e.g., event history garbage collection and condition optimization)

As for future work, it would be interesting to test further systems (e.g., Sentinel [9], NAOS [10], and Monet [22]) as soon as they are available. Furthermore, ADBMS-performance evaluation in multi-user mode is a challenging topic.

Acknowledgements

The work of the three European authors has been supported in part by ACT-NET. ACT-NET is a HCM network funded by the Comission of the European Union; the Swiss part in ACT-NET has been funded by the “Bundesamt für Bildung und Wissenschaft”, BBW.

References

1. R. Agrawal, N. H. Gehani: *Rationale for the Design of Persistence and Query Processing Facilities in the Database Programming Language O++*. Proc. 2nd DBPL, Salishan, 1989.
2. E. Anwar, L. Maugis, S. Chakravarthy: *A New Perspective on Rule Support for Object-Oriented Databases*. Proc. SIGMOD, Washington, DC, May 1993.
3. M. Berndtsson: *Reactive Object-Oriented Databases and CIM*. Proc. 5th Intl. Conf. on Database and Expert Systems Applications, Athens, Greece, September 1994.
4. A. Biliris, E. Panagos: *Transactions in the Client-Server EOS Object Store*. Proc. 11th ICDE, Taipei, Taiwan, March 1995.
5. H. Branding, A. Buchmann, T. Kudrass, J. Zimmermann: *Rules in an Open System: The REACH Rule System*. In N.W. Paton, H.W. Williams (eds): Proc. 1st Intl. Workshop on Rules in Database Systems, Edinburgh, UK, September 1993.
6. A.P. Buchmann: *Active Object Systems*. In A. Dogac, T.M. Ozsu, A. Biliris, T. Sellis (eds): Advances in Object-Oriented Database Systems. Computer and System Sciences Vol 130, Springer, 1994.
7. A.P. Buchmann, J. Zimmermann, J.A. Blakeley, D.L. Wells: *REACH: A Tightly Integrated Active OODBMS*. Proc. 11th ICDE, Taipei, Taiwan, March 1995.
8. M.J. Carey, D.J. DeWitt, J.F. Naughton: *The 007 Benchmark*. Proc. SIGMOD, Washington, DC, May 1993.
9. S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim: *Composite Events for Active Databases: Semantics, Contexts, and Detection*. Proc. 20th VLDB, Chile, Sept. 1994.

10. C. Collet, T. Coupaye, T. Svensen: *NAOS: Efficient and Modular Reactive Capabilities in an Object-Oriented Database System*. Proc. 20th VLDB, Santiago, Chile, Sept. 1994.
11. U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ladin, D. McCarthy, A. Rosenthal, S. Sarin: *The HiPAC Project: Combining Active Databases and Timing Constraints*. SIGMOD Record 17:1, March 1988.
12. J. Eriksson: *CEDE: Composite Event Detector in an Active Object-Oriented Database*. Master's thesis, Department of Computer Science, University of Skövde, 1993.
13. F. Fabret, M. Regnier, E. Simon: *An Adaptive Algorithm for Incremental Evaluation of Production Rules in Databases*. Proc. 19th VLDB, Dublin, Ireland, August 1993.
14. S. Gatziu, K.R. Dittrich: *SAMOS: An Active, Object-Oriented Database System*. Bulletin of the IEEE-TC on Data Engineering 15:1-4, 1992.
15. S. Gatziu, K.R. Dittrich: *Detecting Composite Events in an Active Database Systems Using Petri Nets*. Proc. 4th Intl. Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, February 1994.
16. A. Geppert, S. Gatziu, K.R. Dittrich: *A Designer's Benchmark for Active Database Management Systems: 007 Meets the Beast*. Proc. 2nd Intl. Workshop on Rules in Database Systems, Athens, Greece, September 1995.
17. A. Geppert, S. Gatziu, K.R. Dittrich, H. Fitschi, A. Vaduva: *Architecture and Implementation of an Active Object-Oriented Database Management System: the Layered Approach*. Technical Report 95.29, Institut für Informatik, Universität Zürich, Nov. 1995.
18. J. Gray (ed): *The Benchmark Handbook for Database and Transaction Processing Systems*. 2nd ed., Morgan Kaufmann Publishers, 1993.
19. N.H. Gehani, H.V. Jagadish, O. Shmueli: *Composite Event Specification in Active Databases: Model & Implementation*. Proc. 18th VLDB, Vancouver, August 1992.
20. H.V. Jagadish, D. Lieuwen, R. Rastogi, A. Silberschatz, S. Sudarshan: *Dali: A High Performance Main Memory Storage Manager*. Proc. 20th VLDB, Santiago, Sept. 1994.
21. R. Jain: *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.
22. M.L. Kersten: *An Active Component for a Parallel Database Kernel*. Proc. 2nd Intl. Workshop on Rules in Database Systems, Athens, Greece, September 1995.
23. D. F. Lieuwen, N. Gehani, R. Arlein: *The Ode Active Database: Trigger Semantics and Implementation*. Accepted for 12th ICDE, New Orleans, March 1996.
24. Oracle Corporation: *Oracle7 Server: SQL Reference*. Release 7.2, April 1995.
25. Sybase Inc.: *SYBASE - Data Server*. Berkeley, CA, 1988.
26. D. L. Wells, J. A. Blakeley, C. W. Thompson: *Architecture of an Open Object-Oriented Database Management System*. IEEE Computer 25:10, October 1992.
27. J. Widom: *Research Issues in Active Database Systems*. SIGMOD Record 23:3, September 1994.
28. J. Widom, S. Ceri (eds): *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publishers, 1995.
29. J. Zimmermann, A. Buchmann: *Benchmarking Active Database Systems: A Requirements Analysis*. OOPSLA'95 Workshop on Object Database Behavior, Benchmarks, and Performance; Austin, Texas, 1995.

Appendix: Results of Active Database Management Systems for the BEAST Benchmark

This appendix contains detailed information about the performance measurements of ACOOD, Ode, REACH, and SAMOS using the BEAST benchmark.

A Results for ACOOD

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**244**	170	310	42	229	259
ED6	**417**	360	530	45	401	432
ED7	**151**	110	200	28	141	160
ED8	**1015**	930	1080	52	996	1034
RM1	**231**	170	300	35	219	243
RM2	**53**	30	80	16	48	59
RE1	**259**	220	330	33	247	270
RE4	**415**	370	480	38	402	428

Table 6: BEAST Results for Acood (empty rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**246**	160	350	46	231	260
ED6	**423**	330	550	53	406	439
ED7	**144**	100	190	25	137	152
ED8	**1037**	910	1270	87	1012	1062
RM1	**240**	150	310	41	228	252
RM2	**53**	30	100	16	48	58
RE1	**260**	200	360	40	248	272
RE4	**422**	350	490	42	409	435

Table 7: BEAST Results for Acood (small rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**261**	190	320	35	252	270
ED6	**436**	320	560	58	421	451
ED7	**145**	70	200	30	137	153
ED8	**1048**	930	1180	70	1030	1067
RM1	**248**	190	340	37	238	257
RM2	**56**	30	90	15	52	59
RE1	**261**	180	350	42	250	272
RE4	**426**	360	490	39	416	437

Table 8: BEAST Results for Acood (medium rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**261**	180	330	41	250	273
ED6	**450**	350	540	56	435	465
ED7	**156**	110	200	29	148	164
ED8	**1038**	940	1180	79	1017	1059
RM1	**249**	170	320	38	239	259
RM2	**58**	30	80	15	54	62
RE1	**252**	180	350	47	240	265
RE4	**430**	350	500	41	419	440

Table 9: BEAST Results for Acood (large rulebase)

B Results for Ode

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**33**	10	50	12	29	36
ED3	**221**	150	300	51	207	236
ED6	**174**	90	250	51	159	190
ED7	**158**	60	270	67	138	178
ED8	**509**	80	960	251	434	583
ED9	**269**	90	440	113	236	302
ED10	**196**	100	310	74	175	218
ED11	**831**	440	1250	301	739	923
RM1	**99**	30	160	38	88	110
RE1	**32**	10	60	15	28	36
RE2	**48**	10	100	19	43	53
RE3	**55**	20	90	15	51	59
RE4	**866**	810	1080	55	851	881

Table 10: BEAST Results for Ode (empty rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**32**	10	70	13	29	36
ED3	**230**	150	340	49	216	243
ED6	**189**	110	260	48	175	203
ED7	**181**	100	240	44	168	193
ED8	**627**	300	920	199	567	688
ED9	**256**	80	430	109	224	289
ED10	**203**	70	350	85	180	226
ED11	**875**	410	1410	328	780	969
RM1	**108**	20	200	52	95	122
RE1	**33**	10	60	13	30	37
RE2	**42**	10	80	15	38	46
RE3	**58**	20	100	18	54	63
RE4	**882**	830	950	35	873	892

Table 11: BEAST Results for Ode (small rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**29**	10	60	14	25	33
ED3	**223**	120	330	72	203	243
ED6	**201**	100	310	61	183	218
ED7	**206**	100	310	62	188	223
ED8	**750**	240	1200	295	673	828
ED9	**275**	60	490	131	239	310
ED10	**208**	80	330	80	187	229
ED11	**951**	360	1490	381	844	1058
RM1	**105**	30	180	44	93	117
RE1	**37**	10	80	17	32	41
RE2	**42**	0	80	18	37	47
RE3	**64**	40	90	14	60	68
RE4	**895**	810	1210	90	871	919

Table 12: BEAST Results for Ode (medium rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**33**	20	50	12	29	36
ED3	**234**	120	320	60	216	251
ED6	**249**	120	440	80	225	272
ED7	**222**	100	330	70	198	246
ED8	**975**	410	1630	297	887	1063
ED9	**276**	50	540	130	238	315
ED10	**263**	90	400	88	235	290
ED11	**985**	450	1650	358	878	1091
RM1	**105**	20	190	50	91	120
RE1	**43**	10	80	18	38	48
RE2	**46**	20	70	14	42	50
RE3	**66**	30	120	23	59	73
RE4	**917**	840	1200	105	886	949

Table 13: BEAST Results for Ode (large rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**44**	10	70	14	41	48
ED3	**253**	170	360	48	241	266
ED6	**83**	50	110	16	79	87
ED7	**164**	90	250	48	151	176
ED8	**266**	210	330	29	259	274
ED9	**45**	20	80	16	41	49
ED10	**101**	50	150	22	95	107
ED11	**328**	240	430	45	316	340
RM1	**33**	20	60	10	30	36
RE1	**39**	10	70	16	35	43
RE2	**41**	20	80	16	37	45
RE3	**67**	20	120	19	62	72
RE4	**878**	800	1180	70	859	897

Table 14: BEAST Results for Ode (medium rulebase)

Remarks: in this test series all triggers ED02-ED11 fire, i.e., their event mask evaluates to true.

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**43**	10	70	17	38	47
ED3	**244**	150	340	54	230	258
ED6	**120**	80	170	18	115	124
ED7	**175**	100	240	42	163	186
ED8	**975**	410	1630	297	887	1063
ED9	**276**	50	540	130	238	315
ED10	**263**	90	400	88	235	290
ED11	**985**	450	1650	358	878	1091
RM1	**105**	20	190	50	91	120
RE1	**43**	10	80	18	38	48
RE2	**46**	20	70	14	42	50
RE3	**68**	30	100	20	63	73
RE4	**877**	820	1010	42	866	888

Table 15: BEAST Results for Ode (large rulebase)

Remarks: in this test series all triggers ED02-ED11 fire, i.e., their event mask evaluates to true.

C Results for Reach

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**105**	102	113	3	104	106
ED3	**11632**	11632	19095	6790	8278	14986
ED6	**147**	141	154	4	146	149
ED7	**12076**	3995	16921	8396	8396	15756
ED8	**5004**	3516	6512	1341	4119	5889
ED9	**3231**	2119	4828	1145	2558	3904
ED10	**183**	177	195	5	182	185
ED11	**1616**	189	2059	587	1426	1806
RE1	**62**	61	66	1	62	63
RE2	**34**	32	36	1	33	34
RE3	**60**	57	68	3	59	60
RE4	**252**	246	263	5	251	254

Table 16: BEAST Results for REACH (empty rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**100**	97	103	2	99	100
ED3	**282732**	258149	320789	23463	267244	298219
ED6	**144**	140	151	3	143	144
ED7	**3894042**					
ED8	**6605**	3895	9249	2165	5607	7603
ED9	**3225**	1336	4599	1477	2357	4093
ED10	**185**	180	200	4	184	186
ED11	**1588**	347	2042	593	1315	1862
RE1	**63**	59	69	3	62	65
RE2	**35**	32	44	4	33	37
RE3	**61**	55	75	6	58	63
RE4	**245**	237	256	6	242	248

Table 17: BEAST Results for REACH (small rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**55**	54	58	1	55	56
ED3	**805328**	759079	856798	42767	772547	838109
ED6	**150**	143	154	3	149	150
ED7	**1946300**	1945690	1946900	856	1945150	1947440
ED8	**6608**	6087	7002	385	6313	6904
ED9	**2891**	2244	4167	791	2468	3314
ED10	**202**	197	206	2	201	203
ED11	**1715**	1134	1816	174	1667	1763
RE1	**58**	56	65	2	57	58
RE2	**51**	50	54	1	50	51
RE3	**60**	59	66	1	60	60
RE4	**161**	153	210	10	158	163

Table 18: BEAST Results for REACH (medium rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**331**	324	347	7	330	333
ED3	**4180290**					
ED6	**710**	682	802	25	704	717
ED7	**11526000**					
ED8	**7717**	7195	8239	739	6732	8702
ED9	**3726**	3669	3783	81	3618	3833
ED10	**890**	866	1065	40	879	900
ED11	**1747**	1534	2059	194	1668	1827
RE1	**239**	236	251	3	239	240
RE2	**229**	224	237	3	228	230
RE3	**325**	317	343	6	324	327
RE4	**762**	734	878	38	751	772

Table 19: BEAST Results for REACH (large rulebase)

D Results for Samos

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**445**	390	500	30	436	455
ED3	**472**	390	570	44	458	486
ED6	**2059**	1980	2250	72	2036	2083
ED7	**2236**	2100	2380	85	2209	2262
ED8	**6269**	6020	6460	130	6224	6314
ED9	**5549**	5360	5720	106	5515	5584
ED10	**1816**	1740	1920	49	1800	1832
ED11	**1861**	1750	1960	61	1842	1879
RM1	**425**	370	490	30	416	434
RM2	**43**	30	60	10	40	45
RE1	**460**	430	510	20	454	466
RE2	**448**	410	530	32	438	458
RE3	**419**	370	490	39	407	431
RE4	**911**	850	1000	42	897	924

Table 20: BEAST Results for Samos (empty rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**473**	440	540	27	464	481
ED3	**525**	460	590	35	515	536
ED6	**3681**	3500	3890	110	3644	3718
ED7	**2363**	2270	2510	74	2339	2387
ED8	**7015**	6730	7270	186	6955	7075
ED9	**6592**	6350	6770	143	6546	6639
ED10	**3514**	3370	3820	140	3468	3559
ED11	**4272**	4080	4600	157	4223	4322
RM1	**473**	430	540	29	464	482
RM2	**41**	30	60	9	38	44
RE1	**499**	460	580	28	491	508
RE2	**495**	440	550	28	487	504
RE3	**459**	400	500	32	450	469
RE4	**967**	920	1020	35	956	978

Table 21: BEAST Results for Samos (small rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**527**	460	620	42	515	538
ED3	**545**	480	600	36	535	555
ED6	**5165**	4940	5490	164	5115	5215
ED7	**2463**	2330	2660	84	2440	2486
ED8	**7433**	7210	7760	142	7391	7476
ED9	**7347**	7100	7730	188	7289	7404
ED10	**5046**	4720	5370	156	5001	5091
ED11	**6440**	6200	6870	201	6381	6499
RM1	**501**	450	600	36	491	511
RM2	**42**	30	60	7	40	44
RE1	**542**	480	610	34	533	551
RE2	**527**	460	590	36	517	536
RE3	**473**	420	540	30	465	481
RE4	**1025**	970	1090	35	1015	1035

Table 22: BEAST Results for Samos (medium rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**522**	460	600	38	512	532
ED3	**570**	520	630	32	561	579
ED6	**8124**	7740	8590	234	8053	8196
ED7	**2473**	2260	2650	96	2447	2499
ED8	**7098**	6800	7380	170	7049	7147
ED9	**8378**	8100	8660	173	8327	8430
ED10	**7890**	7630	8180	143	7848	7931
ED11	**10536**	10310	10920	173	10484	10589
RM1	**505**	470	560	28	498	513
RM2	**45**	30	60	7	43	47
RE1	**542**	470	590	28	534	549
RE2	**539**	480	610	37	529	549
RE3	**505**	420	580	32	496	514
RE4	**987**	920	1040	36	976	997

Table 23: BEAST Results for Samos (large rulebase)

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**485**	420	560	33	476	494
ED3	**519**	480	590	28	511	526
ED6	**2142**	2030	2300	72	2123	2162
ED7	**2310**	2170	2470	73	2290	2330
ED8	**6487**	6300	6840	146	6444	6531
ED9	**5956**	5680	6320	182	5905	6007
ED10	**2014**	1900	2270	92	1989	2040
ED11	**2007**	1910	2190	78	1986	2029
RM1	**470**	420	520	26	463	477
RM2	**48**	40	60	6	46	50
RE1	**500**	460	570	33	491	509
RE2	**523**	470	590	30	515	531
RE3	**462**	410	560	34	452	471
RE4	**975**	910	1050	34	965	984

Table 24: BEAST Results for Samos (small rulebase)

Remarks: the composite event detector is not initialized with (useless) comonent events before the test series started.

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**506**	460	560	25	499	513
ED3	**540**	500	620	31	531	549
ED6	**2206**	2090	2460	90	2182	2230
ED7	**2367**	2250	2550	86	2344	2390
ED8	**6548**	6420	6760	93	6521	6575
ED9	**6018**	5770	6370	162	5973	6064
ED10	**2010**	1890	2360	100	1982	2037
ED11	**2041**	1920	2180	77	2021	2062
RM1	**497**	440	570	33	488	506
RM2	**42**	30	60	6	41	44
RE1	**531**	500	590	27	524	539
RE2	**513**	450	570	30	505	521
RE3	**473**	420	540	35	464	482
RE4	**999**	940	1070	40	988	1010

Table 25: BEAST Results for Samos (medium rulebase)

Remarks: the composite event detector is not initialized with (useless) comonent events before the test series started.

Test	Statistic measures					
	mean	min	max	st. dev.	90% conf. interval	
ED2	**502**	460	600	36	492	512
ED3	**569**	510	650	36	559	579
ED6	**2105**	1990	2270	80	2082	2128
ED7	**2246**	2130	2490	91	2221	2270
ED8	**6979**	6710	7590	227	6911	7046
ED9	**5893**	5720	6130	122	5857	5929
ED10	**1986**	1870	2150	67	1968	2004
ED11	**2002**	1920	2230	72	1982	2023
RM1	**510**	460	580	34	500	520
RM2	**42**	30	70	9	40	44
RE1	**536**	490	590	27	529	544
RE2	**533**	490	610	32	524	541
RE3	**491**	420	560	37	480	502
RE4	**988**	920	1090	38	978	998

Table 26: BEAST Results for Samos (large rulebase)

Remarks: the composite event detector is not initialized with (useless) comonent events before the test series started.