

Open Active Services for Data-Intensive Distributed Applications

Christine Collet

Genoveva Vargas-Solar*
IMAG-LSR, University of Grenoble,
BP 72 38402 Saint-Martin d'Hères, France.
{Christine.Collet, Genoveva.Vargas}@imag.fr

Helena Grazziotin-Ribeiro[†]

Abstract

Applications are now highly distributed, heterogeneous and scalable. They comprise autonomous, loosely coupled components, some of them being database management systems, web servers, etc. Considering such a framework, database technology has to evolve towards cooperation and integration.

We propose open and distributed active services that can be used for specifying and generating event and rule managers that support cooperation and interaction between distributed database software (i.e., applications, systems). Event managers are brokers supporting anonymous event passing communication. Rule managers are able to support global (business) rules and execute them using different policies adapted to specific application needs. Both managers can reconfigure and adapt themselves with respect to applications requirements and to their environment.

1 Introduction

1.1 Motivation and Objectives

Future database software (i.e., applications, systems) will be built out of heterogeneous components or services within a fixed framework as proposed in [3, 8, 2, 15, 4]. A database system can be seen as an infrastructure together with a set of services tailored and combined according to specific needs of highly distributed, data-intensive applications. Services are cooperating components that must be supported by mechanisms for managing distributed synchronization, communication, information sharing, accessing, etc.

This paper focuses on a framework for specifying and building Open and Distributed Active Systems (ODAS)

*Supported by the CONACyT and the ECOS-ANUIES program M96E04 of the Mexican Government.

[†]Supported by the CAPES of the Brazilian Government.

comprised of independent event and rule managers. These systems can be used to support transparent integration, cooperation of distributed database and non database reactive components at a middleware level. For example to support electronic commerce applications [1], to manage business rules used for changing statically and dynamically views of Web sites [6]. In such contexts, rules can be processed with event managers at the client level and rule managers at the Web server level.

1.2 Related Work

Existing works for offering active functionalities as open services are recent. They intend to make active capabilities available for (non) database applications in distributed environments and have received a new impetus following the explosive development of Internet.

C²offein [20] proposes separately usable components for distributed systems supported by CORBA. The system consists of parametric wrappers enabling event processing and a rule engine whose characteristics are tailored to specific applications. FRAMBOISE [13] proposes a construction system of ECA-services decoupled from a particular DBMS. Event detection is performed by event detectors which can be specialized for specific DBMS. A rule service is responsible for the maintenance of the rule base and implements rule execution. CoopWARE [23] and TriggerMan [16] also propose execution mechanisms for distributed rules. The former has been developed for a workflow environment, and the latter proposes an asynchronous trigger processor as an extension module for an object-relational DBMS. [7] propose an architecture of a framework to support ECA rules suitable for distributed and heterogeneous systems.

Most approaches do not focus on an event service that interacts with “any” reactive component and rules are always executed as separate transactions, independent of the triggering transactions. Concerning event management our event service compiles techniques proposed by active databases, commercial products such as HP SoftBench [5],

DEC FUSE [17] and Sun ToolTalk [19] and distributed event based infrastructures [11, 24]. But different to these approaches, it provides an event processing layer to offer facilities (i.e., ordering, composing, storing) within a configurable environment. Finally, as us they define adaptable services (statically) but they do not stress on strategies to provide extensibility.

1.3 Main contributions

The originality of our work is that it groups together in a consistent way several functionalities coming from proposals of different areas from outside the classical active database context. The event service compiles event processing strategies that are spread out in existing products and proposals (subscription, filtering) from different domains such as active databases [26, 7], networks (communication protocols) [27, 5, 19], distributed systems [21, 24] and middle-ware [24, 11, 22]. As we already said, event processing with rich semantic can be tailored to different application needs.

The rule service compiles i) classical rule execution characteristics or dimensions that have been revisited for distribution [25, 12, 10]; and ii) new dimensions for characterizing the coordination between rule managers and other components such as event managers, other rule or transaction managers.

Another important aspect of our proposal is the extensibility and adaptability properties given to our services and generated (event and rule) managers.

Event managers enable event-based communication for flexible cooperation and integration of autonomous (database) components. Rule managers provide support for managing global or distributed views, schemas, integrity constraints and components cooperation.

Managers and services can adapt themselves (dynamically) according to specific clients needs and to their execution environment. Also, like proposals in the active database domain [20, 13, 7] our managers can be plugged, unplugged and tailored by database applications according to different internal and external factors.

1.4 Outline of the paper

The paper is organized as follows: Section 2 introduces the general architecture of the active services, the meta models they support and it discusses their properties. Section 3 describes how to build and use ODAS systems. Finally, Section 4 concludes this paper and introduces some on-going and future research directions.

2 Active services

Figure 1 depicts the event and rule services framework. It consists of application programming interfaces (API) for specifying schemas (Specification Interface) that describe the needs of a group of applications concerning event management and rule execution. An *em*-schema encompasses event types definitions and an event management model. Intuitively, it gives an “interpretation” – described by an event management model – to the set of event types it concerns. A *rm*-schema encompasses a set of rule definitions, a rule execution model and a cooperation model. Schemas are used to generate adaptable event/rule managers for a group of applications.

For example, consider a financial application composed of stock exchange information servers, bank servers and desktop applications used by traders. Such components can exchange (produce/consume) events concerning the stock exchange market *actions upward trend, the stock exchange session is closed, etc.* Such events can be managed using to different policies suited for the financial application. Similarly, each component can have associated rules to manage internal integrity constraints (e.g., stock exchange information servers) and to execute operations automatically (e.g., buy, sell actions) at specific moments.

Managers process events/rules according to schema instances (internal functionalities) and offer mechanisms to control and execute adaptation operations. In our financial example, each component can have specific information needs and may react differently to events. For example, traders want fresh information in order to make timely decisions (i.e., buy or sell actions). Stock exchange information servers, can be updated at the end of stock exchange sessions. Components needs are specified through schema instances. This means that event and rule management policies are common to such a group of components, but at the same time each one chooses a policy (from the given ones) according to its own characteristics and needs. We develop these aspects in Section 3.

Applications and other services such as transaction service, persistence service, internet services, etc. use our managers to i) produce/consume events; ii) execute rules; and iii) to adapt managers (i.e., modify instances) according to their internal states and to the environment evolution¹. They can also interact with the specification interface to define and generate managers.

Both services control deployment, runtime and updates of event/rule managers. They can create new schema instances and signal changes to the corresponding managers that consequently adapt themselves. Adaptation depends on the type of changes and is executed at different points of

¹Each application can only modify its own (*em*-schema, *rm*-schema) instances.

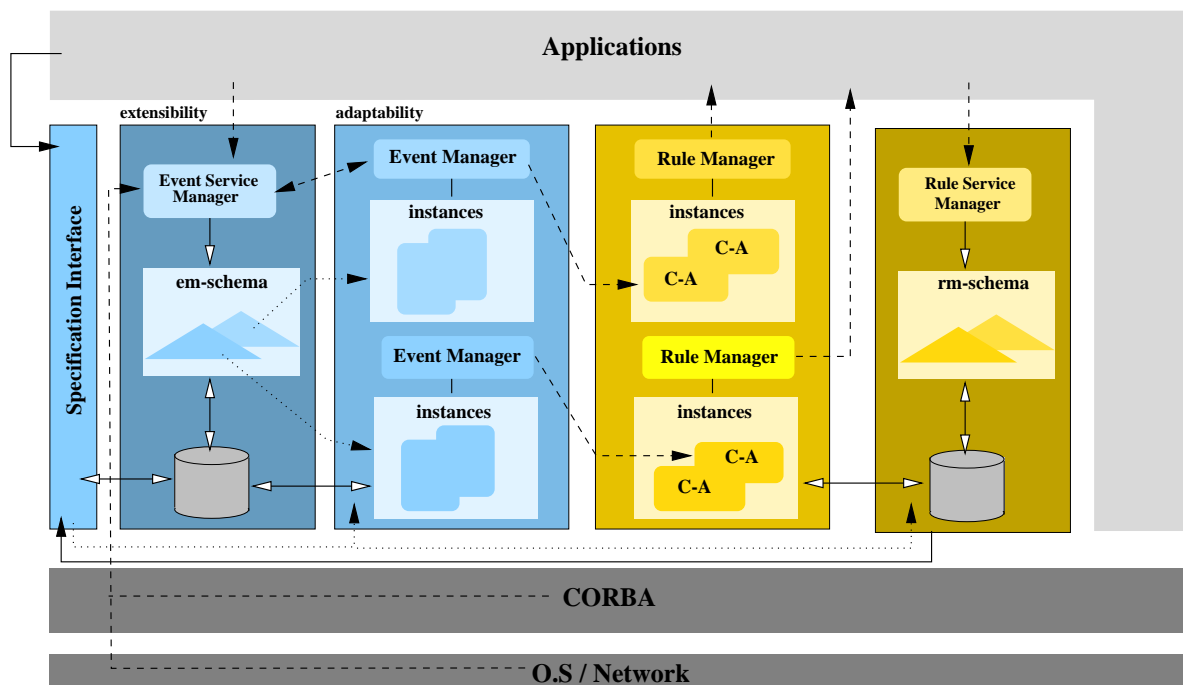


Figure 1. Event and Rule services

event processing or rule execution.

Finally, as shown in Figure 1 the CORBA [18] interface definition language has been used to specify interfaces between components (event managers, event producers/consumers, event and rule services, rule execution engines). CORBA Event Service is used as a “low level”, underlying communication infrastructure.

2.1 Event Service

The Event Service defines a framework for specifying and generating (implementing) event managers. It supports two meta models:

Event Type Meta Model provides (i) a general grammar for describing primitive event types (names and contexts) that concern specific application domains; (ii) concepts for specifying composition operators semantics.

For example in a financial application, the meta model can be used to define events that characterize financial market evolution. The situation *Actions upward trend* can be represented as the following event type :

(<instant> (ActionsUpwardTrend, [s_v, e_v])
[with delta] [where mask]))

<instant> represents the granularity of the observed situation. An event type can either represent a whole process or the fact that a process will be or has been executed. The validity time interval is denoted by [s_v, e_v] where s_v and

e_v represent respectively the starting and ending points in a logical time line. This interval can also be specified as a period (i.e., every two seconds). The delta contains context information about the situation represented by an event type. The mask is a filter expressed in terms of conditions including predicates and temporal expressions that events must satisfy.

Note that event types are defined in a parametric way. Once adopted by applications (i.e., consumers, producers) the instant of detection, the validity time interval, the delta and the mask are instantiated according to an event management model.

Event Management Meta Model proposes a set of dimensions to characterize event processing: detection, production and notification. A domain gives the possible values for a dimension.

Detection dimensions describe the conditions in which events are detected and how they are recognized. They specify the observation granularity; properties such as scope (e.g., events concern the same object, they are produced within the same transaction); possible intervals during which events can be recognized. Production dimensions specify how to order and compute events production instants (with respect to a global reference, under which time representation); how to build events contexts using information contained in delta structures; how to combine events to produce composite events. Notification dimensions con-

cern information delivery, filtering (visibility) and event life-span. Events can be delivered to consumers at different instants and with different degrees of visibility.

Finally, detection and notification dimensions describe interaction protocols depending on producers and consumers characteristics. In general, events can be detected (notified) under pull and push protocols. Both operations can be executed with (a)synchronous modes. The synchronous (asynchronous) mode implies that producer (consumer) executions are (not) interrupted by the detection (notification) mechanism. Full description of these dimensions is out of the scope of this paper, further information can be found in [9].

2.2 Rule Service

The Rule Service can be used for specifying and generating rule managers able to cooperate with event managers (through specific events, detection and notification protocols) and to execute reactions (conditions and actions) upon specific execution models. Reactions are executed within application frameworks and may concern database systems. Specifying a rule manager mainly corresponds to define a schema that encompasses a set of E[C]A rules, an execution model and a cooperation model. The rule service supports three meta models:

Rule definition Meta Model provides concepts for specifying rules of the form :

<behavior> ON <EventType> [IF<Condition>] DO <Action>

<behavior> describes the way i) a rule is executed with respect to event notification; and ii) how to couple its execution within application execution (e.g. transactions). <EventType> describes operations executed by systems and applications (e.g. DBMS), users or situations that happen within the execution context (see Section 2.1). <Condition> is optional and it can be a predicate on database states expressed in a query language, or on information contained in the event context. <Action> is a sequence of operations that can concern multiple applications, DBMS, operating systems and services.

Notice that rules are defined in a parametric way, only E[C]A elements need to be specified. Applications can then adopt a rule and instantiate its behavior properties according to their characteristics and needs (see Section 3).

Rule execution Meta Model provides dimensions to characterize different aspects of rule execution such as event consumption, execution mode, transaction mode, multiple rule policies [14].

Rule triggering policies specify how to handle an event that has triggered a rule. For rule execution events are con-

sidered during their validity time interval. An event can be taken into account either for one execution of the rule or for several executions after its notification until the end of its validity time interval. Rules can be triggered every time their triggering event is notified or only once for a set of triggering events. Furthermore, one can specify whether the net effect must be taken into account or not, for the execution of each rule. The net effect is the result of executing a sequence of operations on the same data (or object).

Reaction execution has to be coupled with the underlying transaction model of a DBMS (local, federated), a specific transaction service or an application execution. Rule execution can start either immediately after the notification of its triggering event or it can be deferred, for example to the end of the global triggering transaction². A rule is executed either as a new transaction that belongs to the same global triggering transaction or as a separate global transaction. In addition, the global transaction in which a rule is executed can be dependent or independent of the global triggering transaction. Clearly, coupling aspects imply that the rule service has both sufficient access over global transactions to be informed of their execution status, and influence to abort, block and restart them according to rule behavior properties.

Rule Cooperation Meta model provides generic contracts describing the way a rule manager cooperates with other components such as event, a transaction or other rule managers.

Event managers cooperation properties given as production rules in Table 1, characterize information exchanged between event and rule managers when: i) rule and event managers establish or break connection; ii) rule managers (un)subscribe to event types; and iii) triggering events are notified.

Table 1. Cooperation with event managers

1	ActivateEM → connection(IdEM)
2	ActivateEvType(EvType) → subscription(EvType)
3	DisactivateType(EvType) → unsubscription(EvType)
4	reception(EvType(ev)) → trigger_rules(ev)
5	DisActivateEM → disconnection(IdEM)

In general, a rule manager has an associated set of event types that can be used to define rules. Events can be processed (i.e., detected and notified) by one or more event managers. Managers interaction is based on subscriptions.

²The triggering transaction is the transaction within which events are produced; a triggered transaction is the one that executes a triggered rule.

Once connections are established (1 in Table 1), a rule manager subscribes the set of event types (2) to be processed by event managers. Later, it receives event instances of subscribed event types (4). Rule managers unsubscribe a given event type (3) when there are no rules triggered by such an event because rules were either erased or deactivated.

Transaction managers cooperation (see production rules in Table 2) include (de)connection of managers (1,9); and transaction management for reaction execution. Rule

Table 2. Cooperation with transaction managers

1	ActivateTM \rightarrow connection(IdTM)
2	ImmRule(IdTT) \rightarrow newT(INDEP)
3	ImmRule(IdTT) \rightarrow newT(DEP_IMM)
4	Prepare(IdTT) \rightarrow newT(DEP_BEFORE)
5	Commit(IdTT) \rightarrow newT(DEP_AFTER)
6	Commit(IdTT) \rightarrow commitT(DEP_COMMIT)
7	Abortion(IdTT) \rightarrow abortT(DEP_ABORT_FAILURE)
8	(IdTR) \rightarrow newT(DEP_REDO)
9	DeactivateTM \rightarrow disconnection(IdTM)

and transaction managers exchange different information depending on rule behavior. For an immediate rule transaction managers start a new global transaction (in)dependent of the triggering one IdTT (see rules 2,3 in Table 2). Differed rules are executed before the prepare phase (4) or after global triggering transactions commit (5). Rules executed as dependent separate transactions, must be explicitly controlled by rule managers: a transaction manager does not commit triggered transactions until it receives an explicit demand from the corresponding rule manager (6). This happens when the triggering transaction has committed, otherwise, rule managers send an abort demand (7). Finally, rule managers must restart triggered transactions IdTT that fail (8).

Rule managers cooperation policies define the way two or more managers can be synchronized to execute rules (see Table 3). Managers exchange their associated priorities (3) and execution contexts (i.e., triggering event -2-, rules to be executed -4-, etc.) for establishing an execution order. They also inform each other about their (de)activation (1, 5).

Table 3. Cooperation with rule managers

1	ActivateRM \rightarrow connection(IdRM)
2	NotifiedEv \rightarrow send_priority(PrRM)
3	receive_priority \rightarrow verify_priorities
4	Send_Rules \rightarrow send_confirmation
5	DeactivateRM \rightarrow disconnection(IdTM)

2.3 Properties

Meta-modeling techniques provide two properties to our services: adaptability and extensibility.

Extensibility is based on the schema concept. It defines the space of possible management policies for a set of types or rules that can be executed by a manager.

Event Manager Schema (em-schema in Figure 1) associates a set of event types with a management model. Event type definitions ($\varepsilon\tau$) are instances of the Event Type Meta Model. An event management model ($\varepsilon\mu$) is an "instance" of the Event Management Meta Model that restricts domains of the proposed dimensions. **Rule Manager Schema** (rm-schema) in Figure 1) couples a set of rule definitions ($r\delta$) with a rule execution model, and the rule model with a cooperation one. Rule definitions concern EA or ECA rules. A rule execution model ($r\varepsilon\mu$) is an instance of the Rule Execution Meta Model that restricts domains of the proposed dimensions. A cooperation model ($coop\mu$) is an instance of the Cooperation Meta Model that provides cooperation contracts (i.e., one per event, transaction or rule manager).

Extensibility is supported through schema evolution, this means that one can i) extend or restrict the domains of a $\varepsilon\mu$ or $r\varepsilon\mu$ according to the "meta-domains"; ii) modify $\varepsilon\tau/coop\mu$ and $r\delta$ adding/deleting or updating event types/cooperation and E[C]A rules.

Adaptability specifies event/rule managers behavior with regard to a group of applications. Adaptability uses the schema instance concept that associates:

- An event type to a management model. For a given event type in $\varepsilon\tau$, an em-schema instance is defined simply by choosing, for each of the dimensions of $\varepsilon\mu$ one and only one value from its corresponding domain. The set of schema instances of an event manager specifies which event types are being processed and which policies are being used.
- A rule to a management model and cooperation model. A rm-schema instance is defined by i) associating each dimension of $r\varepsilon\mu$ with one and only one value of its domain, for a given rule in $r\delta$ ii) specifying a set of cooperating contracts: one for each couple rule-(event, transaction, rule) managers.

Adaptability, i.e., customization of event/rule processing policies (behavior properties), is achieved by updating schema instances with respect to a constant schema. These properties are provided dynamically at runtime considering factors such as event producers and consumers needs, external environment changes, portability, etc.


```

Rule RPurchase
ON before UpdateAccount, [GTbegin, GTend]
    with delta( accountid: string, amount: real)
IF select account
    from account in Accounts
    where accountid = accountid and
        account.credit < amount
DO abort

```

Figure 4. RPurchase

transactions ([GTbegin, GTend]).

Activities can consist of several sub-tasks performed by different actors. For example, a purchase operation concerns a trader that starts an activity, the stock exchange and bank servers. The server processes orders and communicates with the bank server to issue modifications on traders accounts. Finally the server modifies the actions repository and the purchase history. Each sub-task can modify data and consequently cause new actions from actors (e.g., new purchases).

3.1 Schemas specification

Event Manager schema specifies which values are valid to instantiate event type parameters: *instant*, *validity time interval*, *delta*, *mask* (see event type meta model in Section 2) and the way they will be processed (e.g., possible composition modes, notification instants, etc.). Valid values are those specified in the corresponding event management model domains.

For our example, we can define the em-schema $\mathcal{S}_{evt} : \varepsilon\tau_1 \rightarrow \varepsilon\mu_1$. The set of event types $\varepsilon\tau_1 = \{E_1: \text{Actions upward trend}, E_2: \text{the Dow Jones changed by 20\%}, E_3: \text{The stock exchange session is closed}, E_4: \text{a purchase transaction has been completed}, E_5: \text{Update of an account}\}$. $\varepsilon\mu_1$ defines different management policies taking into account sources and traders characteristics and needs. For example events can be detected and notified (a)synchronously using push and pull protocols. Information is notified at different instants, with various reliability degrees. For instance, traders need to be notified automatically as soon as stock exchange events are detected while information sources can wait until the end of a session. Similarly, the trader can receive purchase and bank account events. An event manager can retrieve events from the bank server and notify them once atomic operations are completed. According to \mathcal{S}_{evt} , a valid definition of E_1 is:

```

(after, ActionsUpwardTrend, [9:00, 17:00]
    with delta(company : string, price : real, country : string)

```

E_1 represents upward trend of Telecom actions of a company in country after an update that occurred at an instant in [9:00, 17:00].

Rule Manager schema specifies the way a set of rules can be executed by a rule manager and how it can cooperate with event, transaction and other rule managers. Concerning rules, it specifies which rule behavior can be associated to each rule, by instantiating *behavior* parameters (see rule definition meta model in Section 2). As for events, valid values are those specified in the corresponding execution model domains.

For our financial example we can define a rm-schema $\mathcal{S}_{rule} : r\delta_1 \rightarrow r\varepsilon\mu_1 ; r\varepsilon\mu_1 \rightarrow coop\mu_1$. Where $r\delta_1 = \{RTelecom ; RPurchase\}$ (see Figures 3, 4). The $r\varepsilon\mu_1$ specifies different execution policies taking into account application needs. For instance, since purchases are in general timely operations, RTelecom (triggered by instances of type ActionsUpwardTrend) must be executed immediately as separate independent operations. In contrast, RPurchase (triggered by instances of type UpdateAccount) execution is coupled with global transactions execution performed by the bank federation mediator. Since the rule implements an integrity constraint concerning accounts credit it must be executed as a dependent transaction within the same global triggering transaction (GTT). Also its execution can be differed with respect to the prepare phase of GTT and it is executed for a set of events (event processing *set oriented*). Finally, cooperation policies are specified by $coop\mu_1 = \{em-coop \text{ contract}, tm-coop \text{ contract}, rm-coop \text{ contract}\}$. According to \mathcal{S}_{rule} , a valid instance of RTelecom is shown in Figure 5.

```

Rule RTelecom

CONSUMPTION consume
E_PROCESSING instance
NET EFFECT off
COUPLING immediate, independent, separate

ON after ActionsUpwardTrend, [9:00, 17:00]
    with delta(company: string, domain: string,
        price: real, country: string)
    where domain = 'Telecom'

DO /** get_object purchase object **/
    purchase.buy(company, price, myaccount.number)

```

Figure 5. RTelecom instance

Considering \mathcal{S}_{rule} and \mathcal{S}_{evt} a valid instance of *em-coop contract* is shown in Table 4. As we said before an instance is created for every couple rule-event managers. Note that for event subscription and reception a production rule is created for every event type associated to the rule manager. In our example, $r\delta_1$ contains two rules triggered by E_1 and E_5 . So, we generate rules only for those events (see Table 4).

According to our application a *tm-coop contract* instance must be defined for rules executed in the bank federation. Table 5 shows the corresponding instance that con-

Table 4. Event manager cooperation instance

1	ActivateEM \rightarrow connection(<i>IdEM</i>)
2	(Dis)ActivateEvType(E_1) \rightarrow subscription(E_1)
3	(Dis)ActivateEvType(E_5) \rightarrow subscription(E_5)
4	reception(E_1 (ev)) \rightarrow trigger_rules(ev)
5	reception(E_5 (ev)) \rightarrow trigger_rules(ev)
6	DisActivateEM \rightarrow disconnection(<i>IdEM</i>)

tains production rules needed to execute the set of rules defined in $r\delta_1$. If new rules with different behaviors are considered, new production rules are inserted into **tm-coop contract**. For ECA rules that are not executed within transactional contexts such as **RTelecom** no **tm-coop** contract instances are needed. Similarly, **rm-coop** contract instances are required only when two or more rule managers are specialized to execute reactions over a common set of components. For our application we assume that generated ODAS systems are independent, therefore no **rm-coop** instance is required.

Table 5. Cooperation with transaction managers

1	ActivateTM \rightarrow connection(<i>IdTM</i>)
3	ImmRule(<i>IdTT</i>) \rightarrow newT(DEP_IMM)
6	Commit(<i>IdTT</i>) \rightarrow commitT(DEP_COMMIT)
7	Abortion(<i>IdTT</i>) \rightarrow abortT(DEP_ABORT_FAILURE)
8	(<i>IdTR</i>) \rightarrow newT(DEP_REDO)
9	Deactivate(<i>IdTM</i>) \rightarrow disconnection(<i>IdTM</i>)

3.2 Architecture

General architecture Figure 6 depicts the general architecture of an ODAS system that integrates at least a couple of event and rule services (Event/Rule service managers). Services generate and control event and rule managers that cooperate to execute active rules.

The generation process The generation process of ODAS systems (static subscription in Figure 6) needs a schema and at least one instance to generate an event/rule manager. Application developers can define schemas and instances directly using the specification Interface of the ODAS manager. They can also reuse existing ones by retrieving them from the persistence support. Managers are generated according to application developers specifications.

Dynamic subscription At startup, a dynamic subscription is made for each schema instance concerning a client (DBMS, application, rule manager). When clients signal their presence, event/rule managers activate their associated subscriptions. For each active subscription, a proxy identification (handling object) is supplied, as an interface for managers interaction. Once subscriptions are done, managers start event processing/rule execution of according to subscribed instances. Simultaneously, rules associated to an application are loaded to wait for the arrival of their triggering events.

Event processing is done according to a set of **em-schema** instances that specify contracts between a manager and its clients (producers and consumers). A client receives/produces events according to its associated schema instances. Similarly, rule execution is done according to a set of **rm-schema** instances that specify rule execution policies and interaction contracts among managers that ensure that such policies are verified.

Events are received by the Detection module from proxies supplied at subscription time. Only at the recognition of a subscribed event (i.e., when it is time stamped and filtered), it is sent to a Production module (i.e., for event composition). Once produced (ordered and composed) events are sent to a Notification module. The notification module filters and delivers events. Events are notified to their corresponding rules at different instants using **listeners** (see Figure 6).

Once triggered, rules are scheduled and then executed (see Scheduler and Executor in Figure 6). Rule managers interact with applications (DBMS, federation mediators, applications, services) to evaluate conditions and execute actions.

3.3 Examples

The electronic trader application is built on top of two ODAS systems specified and generated using the above schemas:

Business-ODAS for executing business rules to refresh trader views; and for executing remote purchase operations on the stock exchange server. For such rules event managers are specialized to manage three kinds of event types: i) financial market evolution, ii) purchase operations, iii) banking operations.

Events are detected during the stock exchange session ([9:00,17:00]). Stock exchange events are explicitly signaled using a push synchronous protocol. Banking events (e.g., *an account credit was modified*) are detected using an asynchronous push protocol (i.e., they are signaled by active rules managed by DB-ODAS). Event are notified to the rule

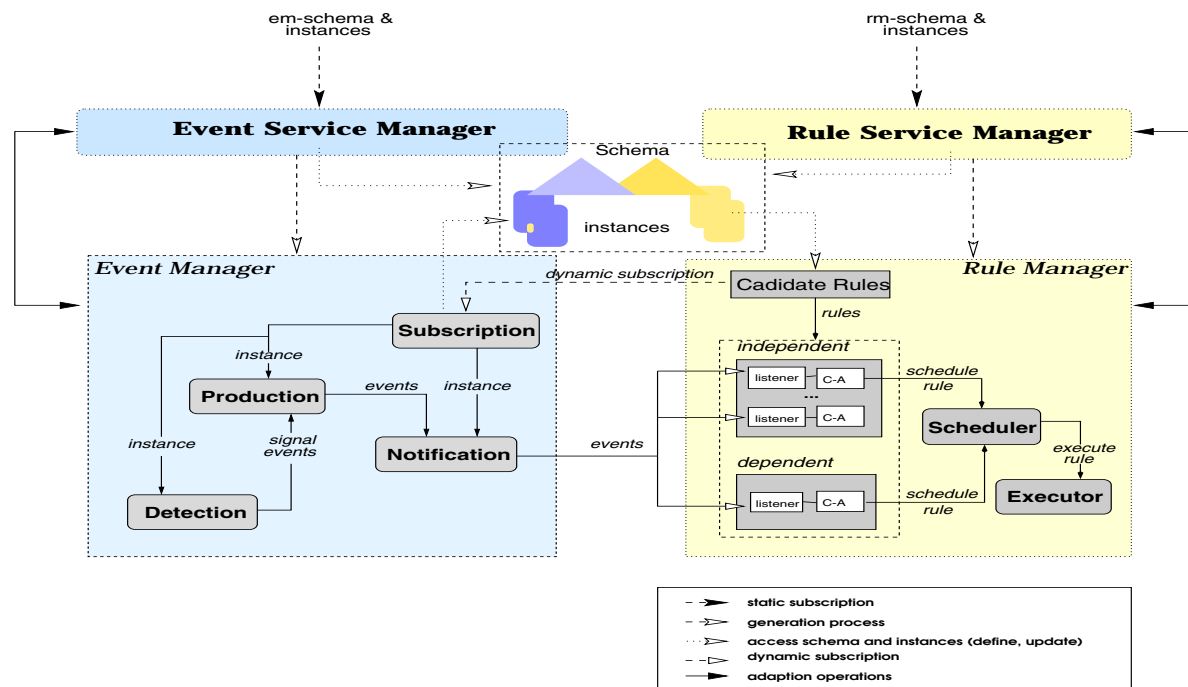


Figure 6. ODAS systems general architecture

manager at different instants. Stock exchange events are notified as soon as detected while purchase ones are notified every hour. In any case they are notified using an asynchronous push protocol.

Business rules are executed immediately after their triggering event notification, as an independent, separate operation with respect to the e-trader application execution. Rules that refresh the financial market view (RTelecom in Figure 5) are executed every time a triggering event is notified. In contrast, rules that control banking information views refreshment are executed only once for a set of events produced during an hour.

DB-ODAS is adapted for active rules in database federations for supporting checking and validation of global integrity constraints, global schema updates and notification of global operations.

Supported event types represent global and local database operations, transactions, user defined types, as well as database and non database applications. Two managers have been specialized respectively for independent and dependent rules execution. Managers detect events with different protocols and modes depending on database management systems (DBMS) and applications specificities. Event composition is done across transactions and applications boundaries. Different notification policies are supported according to rule execution performance requirements.

Active rules execution is coupled with the federation underlying transaction model (global transactions). Rule execution can start either immediately after triggering events notification or it can be deferred, for example to the end of global triggering transactions. A rule is executed either as a new transaction that belongs to the same global triggering transaction or as a separate global transaction. In addition, the global transaction in which a rule is executed can be or not dependent from the global triggering transaction.

4 Conclusions and Future Work

This paper presented a flexible infrastructure for supporting component-based distributed applications construction using rules. It is based on event and rule services used to specify, generate and execute adaptable and extensible managers that may cooperate to detect events and execute rules in different contexts – i.e. (non) transactional, centralized, distributed, etc.

We first overviewed our active services characteristics mainly meta models proposed to describe policies for event type specification; event management; reaction execution. Then we presented our generation approach to specify and implement ODAS systems comprised of event and rule managers. We also showed how to use our services for building ODAS systems dedicated to an e-commerce application.

In conclusion, our main contribution is to provide both

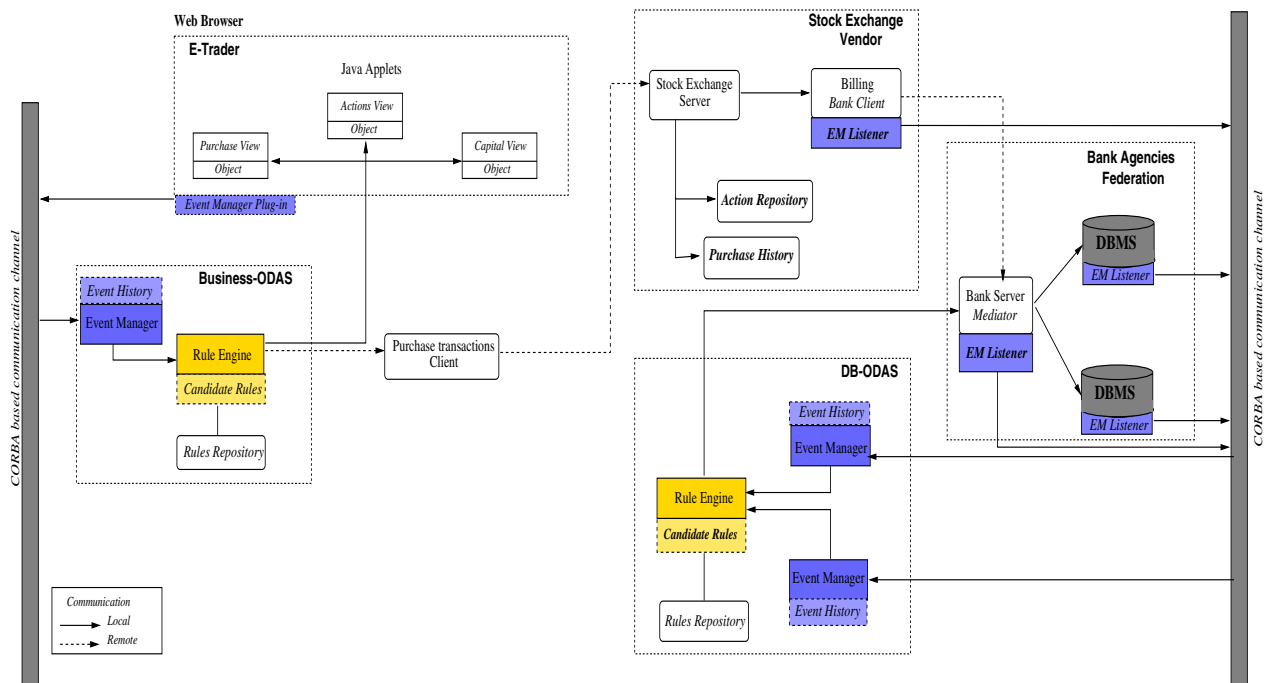


Figure 7. ODAS systems for the distributed application

services and managers as software components customizable according to (database) applications requirements, and to their environment (i.e., other services, the network, etc). Besides the generation approach that provides adaptability, the originality of our proposal is to: i) group together, in a flexible way (meta modeling), functionalities proposed in non classical (active) database domains. ii) use such solutions for improving communication and integration between database services and other components.

On-going work includes i) pursuing the implementation of our services infrastructure; and ii) evaluating it with respect to performance requirements of a pilot application (Web based workflow e-commerce systems). This experience is being held in the NODS project[8] which aims at providing advanced distributed database services such as replication and persistence management of large quantities of heterogeneous data, semantic caching, change management, etc. In this context, the event service can be used to integrate a set of database services that offer adaptable DBMS functionalities. The rule service should be used for consistency maintenance and cooperation management between these services.

References

[1] Serge Abiteboul, Bernd Amann, Sophie Cluet, Adi Eyal, Laurent Mignet, and Tova Milo. Active views for electronic commerce. In *VLDB'99, Proceedings*

of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK. Morgan Kaufmann, 1999.

- [2] Roger Bamford, Rafiul Ahad, and Angelo Pruscino. A scalable and highly available networked database architecture. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK.* Morgan Kaufmann, 1999.
- [3] Phil Bernstein, Michael Brodie, Stefano Ceri, David DeWitt, Mike Franklin, Hector Garcia-Molina, Jim Gray, Jerry Held, Joe Hellerstein, H. V. Jagadish, Michael Lesk, Dave Maier, Jeff Naughton, Hamid Pirahesh, Mike Stonebraker, and Jeff Ullman. The Asilomar Report on Database Research. *SIGMOD Record*, 18(1), september 1998. Asilomar meeting of 1998.
- [4] Michael L. Brodie and Surajit Chaudhuri. Issues in network management in the next millennium. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK.* Morgan Kaufmann, 1999.
- [5] M.R. Cagan. The HP SoftBench environment: an architecture for a new generation of software tools. *Hewlett-Packard Journal: technical information from*

- the laboratories of Hewlett-Packard Company*, 3(41), June 1990.
- [6] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. Data-driven, one-to-one web site generation for data-intensive applications. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*. Morgan Kaufmann, 1999.
- [7] S. Chakravarthy, R. Le, and R. Dasari. ECA Rule Processing in Distributed and Heterogeneous Environments. In *Proceedings of the Fourteenth International Conference on Data Engineering*, Florida, USA, February 1998. IEEE Computer Society Press.
- [8] C. Collet. The NODS Project: Networked Open Database Services. In *Proc. of the 14th European Conference on Object-Oriented Programming (ECOOP 2000)- Symposium on Objects and Databases*, Cannes, France, June 2000. <http://www-lsr.imag.fr/Les.Groupes/storm/NODS/index.html>.
- [9] C. Collet and G. Vargas-Solar. A flexible event service for database co-operating components. Technical report, LSR-IMAG, Grenoble, France, 2000.
- [10] T. Coupaye and C. Collet. Semantics based implementation of flexible execution models for active database systems. In *les actes des 14ièmes Journées Bases de Données Avancées*, Hammamet- Tunisie, october 1998.
- [11] Flanagan, editor. *JAVA in a nutshell*. O'Reilly, 1997. O'Reilly andssociates, Inc.
- [12] P. Fraternali and L. Tanca. A Structured Approach for the Definition of the Semantics of Active Databases. *ACM Transactions on Database Systems*, 20(4), dec 1995.
- [13] H. Frithschi, S. Gatzju, and K.R. Dittrich. Framboise – an approach to construct active database mechanisms. Technical Report 97.04, Department of Computer Science, University of Zurich, Zurich, April 1997.
- [14] H. Grazziotin-Ribeiro and C. Collet. Behavior of active rules within multi-database systems. In *Proceedings of the XIV Symposium on Databases*, Florianopolis-Brazil, october 1999.
- [15] James Hamilton. Networked data management design points. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*. Morgan Kaufmann, 1999.
- [16] E.N. Hanson and S. Khosla. An introduction to the triggerman asynchronous trigger processor. Technical Report TR-97-007, CISE Department, University of Florida, United States, April 1997.
- [17] R.O. Hart and G. Lupton. Dec fuse: Building a graphical software development environment from unix tools. *Digital Technical Journal of Digital Equipment Corporation*, 7(2), Srping 1995.
- [18] Object-Oriented Concepts Inc. ORBacus 3.1.3. <http://www.ooc.com/index.html>, 1999.
- [19] A.M. Julienne and B. Holtz. *ToolTalk and open protocols, inter-application communication*. Prentice-Hall, New Jersey, 1994.
- [20] A. Koschel, R. Kramer, G. Bültzingslöwen, T. Bleibel, P. Krumlinde, S. Schmuck, and C. Wein. Configuration Active Functionality for CORBA. In *Proceedings of the ECOOP97 Workshop*, Jyväskylä, Finland, June 1997.
- [21] E. Lenormand. *Communication par événements dans les modèles à objets*. PhD thesis, University of Joseph Fourier, novembre 1996.
- [22] MOMA Educational Information: A Middleware Taxonomy. <http://www.moma-inc.org/>, 1998. White paper.
- [23] J. Mylopoulos, A. Gal, K. Kontogiannis, and M. Stanley. A generic integration architecture for cooperative information systems. In *Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 208–217, Brussels, Belgium, June 1996. IEEE.
- [24] OMG, editor. *The Common Object Request Broker Architecture and Specification*. Object Management Group, 1997.
- [25] N. Paton, O. Díaz, M.H. Williams, J. Campin, A. Dinn, and A. Jaime. Dimensions of Active Behaviour. In N. Paton et M. Williams, editor, *Proceedings of 1st Int. Workshop on Rules in Database Systems (RIDS'93)*, Edinburgh - Scotland, September 1993. Springer Verlag.
- [26] N. W. Paton. *Active Rules for Databases*. Springer Verlag, 1998.
- [27] S. Reiss. Connecting tools using message passing in the field environment. *IEEE Software*, July 1990.