The PADRES Event Processing Network: Uniform Querying of Past and Future Events

Das PADRES Ereignisverarbeitungsnetzwerk: Einheitliche Anfragen auf Ereignisse der Vergangenheit und Zukunft

Hans-Arno Jacobsen, Vinod Muthusamy, Guoli Li, University of Toronto

Summary This paper outlines requirements and sketches techniques for introducing to the publish/subscribe model the capability to uniformly access data produced in the past and future. The new model can filter, aggregate, correlate and project any combination of historic and future data. A flexible architecture is presented consisting of distributed and replicated data repositories that can be provisioned in ways to tradeoff availability, storage overhead, query overhead, query delay, load distribution, parallelism, redundancy and locality. ▸▸▸ **Zusammenfassung** Dieses Papier umreißt Anforderungen und Techniken, Publish/Subscribe um einheitliche Anfragen auf Daten zu erweitern, die sowohl in der Vergangenheit als auch in der Zukunft liegen können. Es wird eine flexible Architektur bestehend aus verteilten und replizierten Datenrepositorien vorgestellt, die verschiedene Auflösungen von Zielkonflikten bzgl. Verfügbarkeit, Speicheraufwand, Anfrageaufwand, Lastverteilung, Nebenläufigkeit, Redundanz sowie Ortsbezug ermöglichen.

Keywords H.3.3 [Information Systems: Information Storage and Retrieval: Information Search and Retrieval] Information filtering; H.4 [Information Systems: Information Systems Applications]; Publish/Subscribe, Content-based Routing, Historic Data Query, Event Processing Network, Event Processing, Event ▸▸▸ **Schlagwörter** Ereignisverarbeitung, Verarbeitungsnetzwerk, historische Daten

1 Introduction

The publish/subscribe (pub/sub) paradigm provides a simple and effective method for disseminating data while maintaining a clean decoupling of data sources and sinks [1; 6; 7; 9; 12; 18; 22]. This decoupling can enable the design of large, distributed, and loosely coupled systems that intercorporate through simple publish and subscribe invocations. A large variety of emerging applications benefit from the expressiveness, the filtering, the distributed event correlation, and the complex event processing capabilities of content-based publish/subscribe. These applications include RSS feed filtering [21; 22], system and network management, monitoring, and discovery [8; 17; 26], business process management and execution [15; 23], business activity monitoring [8], workflow management [6; 15], and automatic service composition [10; 11].

A limitation of the traditional publish/subscribe model is that it only delivers to subscribers those publications produced after the subscription was issued. Publish/Subscribe is a model to query the future. There are many application contexts, however, where access to publications from the past is necessary, such as for auditing purposes, replaying a business process execution to debug it, or tracing system events to perform root cause analysis. Even more interesting uses arise when

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

data from the past can be correlated with those in the future. For example, an algorithmic trader may wish to be notified when a stock behaves as it did during a recent economic downturn (incoming stock quotes are being correlated with those from the past). While such analysis can be performed periodically with traditional databases, the publish/subscribe model can detect these complex patterns in real-time and immediately notify subscribers as they occur.

The challenge is to support access to both historic and future publications[1] through a unified publish/subscribe interface while preserving the desirable properties of the publish/subscribe model. For example, in content-based publish/subscribe, addresses of participants are not available, so directly querying a database is not an option. The client should not even know which database to query. Also, *composite subscriptions* [12] that allow correlations, or joins, across publications should work with any combination of historic and future data.

2 Related Work

2.1 Publish/Subscribe

The publish/subscribe model supports push-based decoupled many-to-many interaction between information producers (*publishers*) and consumers (*subscribers*) [1; 6; 7; 9; 12; 18]. Data (*publications*) are pushed to subscribers who express their interests using *subscriptions*. Early systems were channel-based and delivered all publications on a particular channel to interested subscribers, whereas the more expressive content-based model allows subscriptions to express constraints on the content of publications. In all of the above systems, subscribers are only notified of publications issued after their subscriptions. None of the models proposed today can retrieve publications from before the registration of a subscription. In this paper, we motivate a unified way to access both future and historic publications.

To improve subscription expressiveness, Badrish et al. [2] separate publish/subscribe processing into *subscription processing* and *notification dissemination*. Publications are first inserted into a database where subscriptions are evaluated, and then matching results are disseminated over a publish/subscribe overlay. Processing subscriptions at a database facilitates aggregation and join operations, but subscriptions are still restricted to matching publications in the future. Furthermore, the databases can be distributed but not replicated.

In this paper, we elaborate on a model that supports subscriptions for both future and historic publications in a way that future publications are delivered directly without an intervening database, and aggregations and joins are processed within the distributed publish/subscribe overlay (for future publications), or at the databases (for historic data). We also exploit the opportunity to support projection capabilities missing in typical publish/subscribe systems.

[1] Publications issued before a subscription, are considered historic. The past and future are with respect to the time a subscription is issued.

2.2 Stream Processing and Continuous Queries

Continuous queries [3; 4] are issued once and run "continually" over the database. NiagaraCQ [4] uses an incremental query evaluation method but is not limited to append-only data sources. It uses a static query optimizer, and operators from different queries can be shared. PSoup [3] treats data and queries symmetrically as streams, allowing queries to access both data that arrived before the query registration and data in the future. However, PSoup uses a main-memory data structure limiting historic data to memory size. Also as with any centralized architecture, it presents a failure and performance bottleneck.

Content-based publish/subscribe differs from continuous queries and stream processing. First, a subscription may match publications from an anonymous and unknown number of data sources which may not conform to a predefined schema, whereas queries on streams typically explicitly identify the source streams and can rely on data conforming to known schemas, greatly simplifying query processing and routing decisions. Second, we allow clients to subscribe to both historic and future data. The historic data are stored in a distributed set of databases that are not limited by main memory.

3 Message Routing

3.1 Language Model

PADRES is a distributed publish/subscribe system, developed by the Middleware Systems Research Group at the University of Toronto. PADRES provides a SQL-like syntax PADRES SQL (PSQL) [13], which allows users to uniformly access data produced in the past and the future. The PSQL language includes the specification of notification semantics, and it can filter, aggregate, correlate and project any combination of historic and future data as described below. In PADRES, a message has a message header and a message body. The header includes the message identifier, which is unique throughout the system, the last hop and next hop information, which indicates where the message comes from and where it will be forwarded to, and the time stamp when the message is generated. There could be four types of objects in the message body: publications, advertisements, subscriptions, and notifications.

Advertisements

Before each publisher issues its publications, it specifies a template that describes the publications it will publish. This is done by issuing an *advertisement* message. The advertisement is an indication of the data that the publisher is going to publish in the future. In this sense, an advertisement is like a database schema or a program-

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

ming language type. An advertisement is said to *induce* publications. That means the attribute set of an *induced* publication is a subset of attributes defined in the advertisement, and values of each attribute in an induced publication must satisfy the predicate constraint defined in the advertisement. Note that only publications induced from an advertisement of a publisher are allowed to be published by the publisher. We adapt the equivalent SQL table creation statement to express advertisements.

```
CREATE TABLE (attr op val[, attr op val]*)
```

PSQL's `CREATE TABLE` differs slightly from the same statement in SQL. Tables are unnamed since they need not be referred to by subscriptions or publications. Also, the range of values of each attribute (or column) can be specified. Moreover, regardless of the attribute value constraint, each attribute can implicitly be a null value. In the rest of this paper, we use an online retail workflow as example. The following statements setup two event sources. The 'invoke' event indicates the activation of a service with a particular service generation identifier and an additional item attribute. The 'result' event represents the results received from a service instance. It includes the service name, the item, the service id, and the state of the result, which is an integer value from 0 to 10 representing different result states. Values greater than 5 represents faulty states of the service.

```
CREATE TABLE (class = invoke, service = *, item = *, id = *)
CREATE TABLE (class = result, service = *, item = *, id = *,
              state = *)
```

In the above example, * is a wildcard that indicates that the corresponding attribute may have any value.

Publications

A publication is expressed using a construct similar to SQL's `INSERT` statement.

```
INSERT (attr[, attr]*) VALUES (val[, val]*)
```

The following publications are compatible with the advertisement schema defined above.

```
INSERT (class, service, id)
 VALUES (invoke, ItemView, a0012)
```

Notice that only a subset of attributes defined in the schema need to be specified. A publication may also contain a *payload*, which is an optional data value delivered to subscribers. The payload cannot be referenced by a subscription's constraints. In many applications, publications represent events. Often, both terms are used synonymously in the publish/subscribe literature.

Subscriptions

Subscribers express interests in receiving publications by issuing *subscriptions*. Subscriptions set constraints on matching publications. PADRES not only allows subscribers to subscribe to individual publications, but also allows correlations or joins across publications. In that sense, subscriptions can be classified into *atomic subscriptions* and *composite subscriptions*.

Subscribers issue `SELECT` statements to query both historic and future publications. With `SELECT`, a subscriber can specify a set of attributes or functions that she wants to receive once the subscription is matched. The `WHERE` clause indicates the predicate constraints applied to matching publications. The `FROM` and `HAVING` clauses are optional and are used to express joins and aggregations.

```
SELECT [ attr | function ], ...
 [FROM src, ...]
 WHERE attr op val, ...
[HAVING function, ...]
[GROUP BY attr, ...]
```

A traditional publish/subscribe subscription for future publications would look as follows in PSQL.

```
SELECT *
 WHERE class = invoke, service = 'ItemView'
```

Note that the above statement does not query a single table, so the results may have any number of attributes. The only guarantee is that all notifications will have the *class* and the *service* attributes with values constrained as specified.

Reserved attributes *start_time* and *end_time* specify time constraints, and are used to query for publications from the past, the future, or both. For example, when a new product is on-sale, the sales manager might want to monitor how many people are interested in this item. The following subscription queries data in a time window that begins one hour before the time the query is issued and extends into the future.

```
SELECT *
 WHERE class = invoke, service = 'ItemView',
       item = 'T-Shirt', start_time = NOW - 1h,
       end_time = NOW + 4h
```

The system internally splits the above subscription: one purely historic subscription that is evaluated once, and one ongoing future subscription. A subscription for both historic and future data is a *hybrid* subscription.

Publish/Subscribe composite subscriptions [12] can be expressed with simple join conditions. The event correlation is supported using the `FROM` clause, where the event pattern can be specified using Boolean expressions. For instance, to monitor the effect of shipping delay on customers interests, the below subscription is issued. It is a hybrid and composite subscription example. The subscription queries shipping information of items delayed over the past two months (state 2 in the example represents a delay in shipment), correlates these items with items browsed by customers in the future, and reports the invocation of the 'ItemView' service on the item as notification. A decrease in the number of matches of this subscription over time may indicate that customers loose interest in items whose shipment is delayed.

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

```
SELECT e2.service, e2.item, e2.id
  FROM e1 AND e2
 WHERE e1.class = result, e1.service = ItemShipped,
       e1.state = 2, e2.class = invoke,
       e2.service = ItemView,
       e1.start_time = NOW - 2 months,
       e1.item = e2.item
```

The identifer in the FROM clause specifies that two different publications are required to satisfy this query, and each publication must match the WHERE constraints. The two publications may come from different publishers, and may conform to different schemas, as long as they match the specified constraints.

Notice that a composite subscription can collect, correlate, and filter publications in the event processing network. Without this feature, a user must retrieve all future publications and then query databases for associated historic data. This would be expensive (both for the user and in terms of network traffic) in cases where the future publications are generated frequently.

Event aggregation is supported in PSQL as well. The HAVING clause can specify constraints across a set of matching publications. The functions AVG(a_i,N), MAX(a_i,N), and MIN(a_i,N) compute the appropriate aggregation across attribute a_i in a window of N matching publications. The window may either slide over matching publications, or be reset when the HAVING constraints are satisfied. For example, in our scenario services return result states in the 'state' attribute, where values greater than 5 indicate faults. Sales managers need to know the most severe problem customers experience. The following subscription matches if the maximum fault state reaches 5.

```
 SELECT *
  WHERE type = result
 HAVING MAX(state, 10) > 5
GROUP BY service
```

Any attributes specified by functions in the HAVING clause must appear in the publication. So, an implicit state = * condition is added to the WHERE clause above. Also, the GROUP BY clause has the same semantics as in SQL and serves to constrain the set of publications over which the HAVING clause operates.

Notifications

When a publication matches a subscription at a broker, a *notification* message is generated and further forwarded into the broker network until it is delivered to subscribers. Notification semantics do not constrain notification results, but transform them. Notifications may include a subset of attributes in matching publications indicated in the SELECT clause in PSQL. Most existing publish/subscribe systems use matching publication messages as notifications. PSQL supports projections and aggregations over matching publications to simplify notifications delivered to subscribers and reduce overhead by eliminating unnecessary information.

3.2 The PADRES Event Processing Network

The PADRES system consists of a set of brokers connected in an overlay network, as shown in Fig. 1. The overlay network forms the basis for message routing and event processing. Each PADRES broker acts as a content-based message router that routes and matches publish/subscribe messages. Each PADRES broker is essentially an event processing engine. The PADRES overlay constitutes an event processing network that can filter events published by many sources, distribute events to many subscribers, correlate and aggregate events from multiple sources to detect composite events, match composite subscriptions and match subscriptions for future events, historic events, and combinations of future and historic events [12; 13; 16].

A PADRES broker only knows its direct neighbors. The overlay information is stored in the *Overlay Routing Tables* (ORT) at each broker. Clients connect to brokers using various binding interfaces such as Java Remote Method Invocation (RMI) and Java Messaging Service (JMS). Publishers and subscribers are clients to the overlay. A publisher issues an advertisement before it publishes. Advertisements are effectively flooded to all brokers along the overlay network. A subscriber may subscribe at any time. The subscriptions are processed according to the *Subscription Routing Table* (SRT), which is built based on the advertisements. The SRT is essentially a list of *[advertisement, last hop]* tuples. If a subscription intersects an advertisement in the SRT, it will be forwarded to the last hop broker the advertisement came from. Subscriptions are routed hop by hop to the publisher, who advertises information of interest to the subscriber. Subscriptions are used to construct the *Publication Routing Table* (PRT). Like the SRT, the PRT is logically a list of *[subscription, last hop]* tuples, which is used to route publications. If a publication matches a subscription in the PRT, it is forwarded to the last hop broker of that subscription until it reaches the subscriber. A diagram showing the overlay network, SRT and PRT is provided in Fig. 1. In this figure in Step 1 an advertisement is published at B_1. In Step 2 a matching subscription enters from B_2. Since the subscription over-

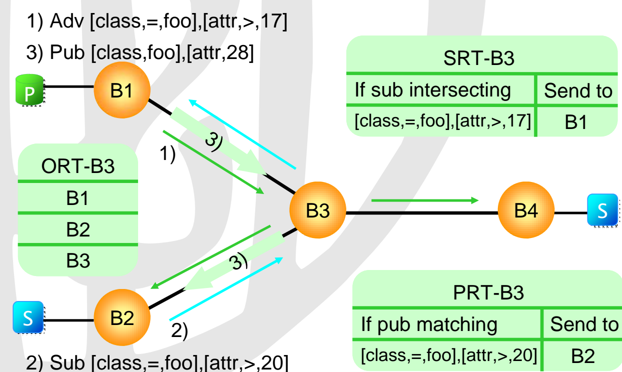


Figure 1 PADRES Broker Network.

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

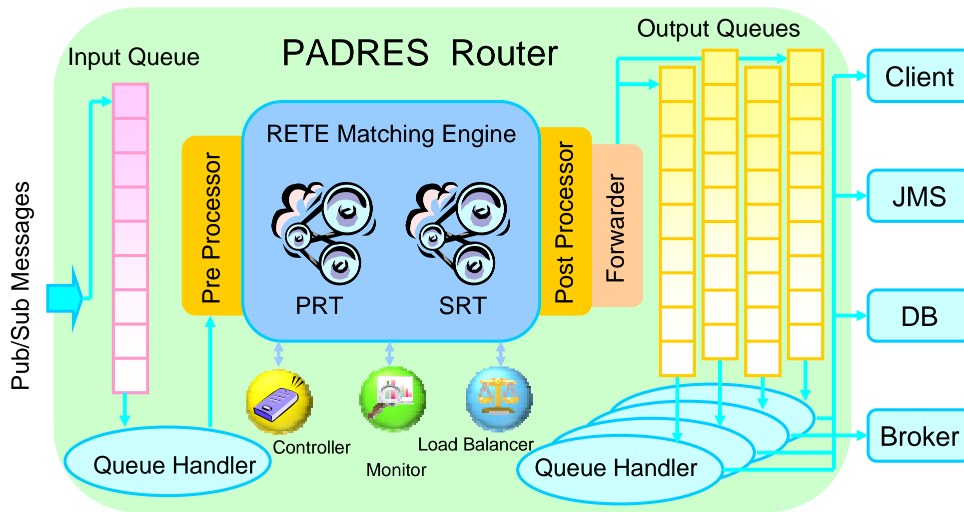


Figure 2 PADRES Broker Architecture.

laps the advertisement at broker B_3, it is sent to B_1. In Step 3 a publication is routed along the path established by the subscription to B_2.

Each broker consists of an input queue, a router, and a set of output queues, as shown in Fig. 2. A message arrives in the input queue. The router takes the message from the input queue, matches it against existing messages according to the message type, and puts it in the proper output queues representing different destinations. Other components are provided to support advanced features. For example, the controller component provides an interface for a system administrator to manipulate a broker (e. g., shut down a broker, inject a message into a broker, etc.); the monitor component collects operational statistics (e. g., incoming message rate, average queueing time, and matching time, etc.). If a broker is overloaded (e. g., the incoming message rate is above a certain threshold), a load balancer triggers offload algorithms [5] to balance the traffic among brokers. A failure detector monitors the broker network. If a failure is detected, a recovery procedure is triggered in order to guarantee message delivery in presence of failures [24]. PADRES also proposes a novel policy model and framework for content-based publish/subscribe systems that benefits from the scalability and expressiveness of existing content-based publish/subscribe matching algorithms [25].

3.3 Historic Data Access Architecture

Subscriptions for future publications are routed and handled as usual [1; 6; 7; 16; 19]. To support historic subscriptions, databases are attached to a subset of brokers as shown in Fig. 3. The databases are provisioned to sink a specified subset of publications, and to later respond to queries. The set of possible publications, as determined by the advertisements in the system, is partitioned and these partitions assigned to the databases. A partition may be assigned to multiple databases to achieve replication, and multiple partitions may be assigned to the same database if database consolidation is desired. Partition assignments can be modified at any time, and replicas will synchronize among themselves. The only constraint is that each partition be assigned to at least one database so no publications are lost. Partitioning algorithms, partition selection and partition assignment policies are described and evaluated in [13].

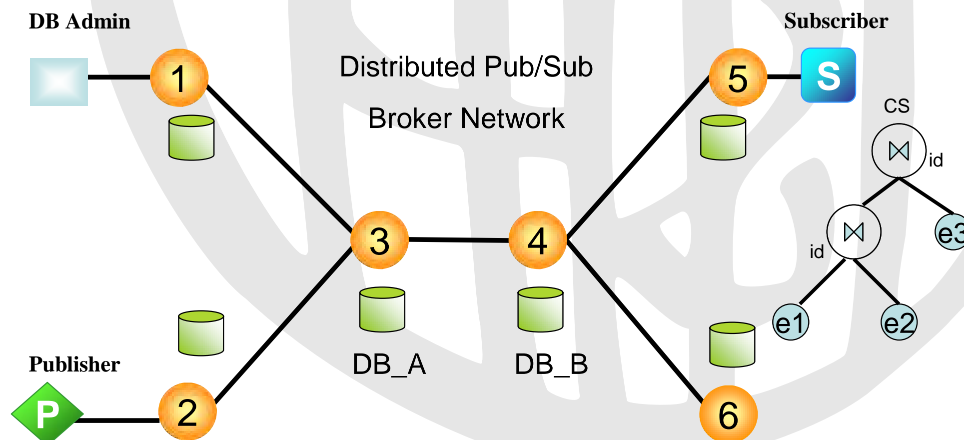


Figure 3 Historic Data Access.

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

Each database subscribes to DB_CONTROL publications addressed to it, and the administrator assigns partitions to databases by sending publications with STORE commands to the appropriate database. For example, the following publications assigns a partition to database *DB_A*.

```
INSERT (class, command, db, partition_spec)
VALUES (DB_CONTROL, STORE, DB_A,
       'SELECT * WHERE shipID = *, level< 10')
```

The partition specification is itself a subscription with the selection formula expressed in the WHERE clause. A database that receives the STORE command will extract the partition specification and issue it as an ordinary future subscription. Matching publications will then be delivered to the database which will store them. A database assigned a partition also issues an advertisement that defines its partition.

When the first broker receives a historic subscription issued by a subscriber, it assigns it a unique query identifier and then routes the subscription as usual towards publishers whose advertisements intersect the subscription. This ensures that the subscription will arrive at databases whose partitions intersect the subscription. The database(s) convert(s) the subscription into a SQL query, retrieve(s) matching publications from the database, and publish(es) the results. These "historic" publications are annotated with the subscription's unique query identifier so they are only delivered to the requesting subscriber. After the result set has been published, the database will issue an END publication, which is used to *unsubscribe* the historic subscription.

The interaction with the databases fully leverages the content-based publish/subscribe model, and the databases are never addressed directly. In fact, it is impossible for publishers to discover where their publications are being stored, or for subscribers to know which databases process their queries. This simplifies management since databases can be moved, added or removed, and partitions reassigned at will.

To improve availability, fault-tolerance and query performance, a partition may be replicated. Partition assignment strategies include partitioning, partial replication and full replication.

With partitioning, a database may be assigned several partitions, but each partition is assigned to only one database. That is, there is only one replica per partition. With partial replication, a given partition may be replicated by assigning it to multiple databases. With full replication, every database maintains replicas of all partitions. That is, each database stores all publications.

The various strategies have tradeoffs and are appropriate under different circumstances. The partitioning policy is simple and avoids replica consistency issues, but is sensitive to failures. Partial replication can tolerate failures of all but one replica, but requires logic to ensure the historic subscription is answered by only one of the replicas. Full replication is even more robust, and historic subscriptions can always be answered fully by the nearest database, minimizing network traffic. However, the high degree of replication imposes greater overall traffic and storage costs, as well as larger synchronization overhead. The partition assignment policies allow an administrator to tradeoff storage space, routing complexity, query delay, network traffic, parallelism of queries, and robustness. Detailed discussion, algorithms, and evaluations of these strategies can be found in [13].

3.4 Subscription Routing

Subscriptions in PADRES can be *atomic* expressing constraints on single publications, or *composite* expressing correlation constraints over multiple publications.

Atomic Subscription Routing

When a broker receives an atomic subscription, it checks the *start_time* and *end_time* attributes. A future subscription is forwarded to potential publishers using standard publish/subscribe routing [1; 6; 7; 16; 19]. A hybrid subscription is split into future and historic parts, with the historic subscription routed to potential databases as described next.

For historic subscriptions, a broker determines the set of advertisements that *overlap* the given subscription, and for each partition, selects the database with the minimum routing delay. The subscription is forwarded to only one database per partition to avoid duplicate results. When a database receives a historic subscription, it evaluates it as a database query, and publishes the results as publications to be routed back to the subscriber. Upon receiving an END publication, after the final result is published, the subscriber's host broker unsubscribes the historic subscription. This broker also unsubscribes future subscriptions whose *end_time* has expired.

Composite Subscription Routing

Topology-based composite subscription routing evaluates correlation constraints in the network where the paths from the publishers to the subscribers merge [12]. If a composite subscription correlates with a historic data source and with a publisher, where the former produces more publications, correlation detection would save network traffic if moved closer to the database, thereby filtering potentially unnecessary historic publications earlier in the network. We propose the adaptive composite subscription routing protocol in [13], which determines the locations of event correlation, refereed to as the *join points*, based on a routing cost model. The cost model minimizes the network traffic and the notification routing delay.

When network conditions change, join points may no longer be optimal and should be recomputed. A join point broker periodically evaluates the cost model, and upon finding a broker able to perform detection cheaper than itself, initiates a join point movement. The state transfer from the original join point to the new one in-

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

cludes routing path information and partial matching states. Each part of the composite subscription should be routed to the proper destinations so routing information is consistent. Publications that partially match composite subscriptions stored at the join point broker must be delivered to the new join point.

4 Applications

Business process management (BPM) is an important domain where content-based publish/subscribe systems are extremely useful [15]. One of the key aspects of BPM is workflow processing. Figure 4 shows a simplified workflow of an online retailer. A workflow specifies the interactions among different enterprise applications, processes, services, and users. Workflow interactions are usually causal, temporal, and inherently event-driven. The completion of one stage of the workflow triggers the execution of the next stage, where stages are applications, processes, services, and users. A content-based publish/subscribe system is ideally suited to model this kind of event-driven orchestration and choreography.

For example, Fig. 4 shows the workflow of an online sales application where the availability of an item is checked and the shipping charge is calculated before a detailed item view is enabled. To model this interaction, the 'ItemView' module subscribes as follows:

```
SELECT *
 FROM e1 AND e2 AND e3
WHERE e1.class = invoke, e1.service = ItemView,
      e2.class = result, e2.service = AvailCheck,
      e3.class = result, e3.service = CalcShipping,
      e1.id = e2.id, e2.id = e3.id
```

Note that by issuing this composite subscription, the 'ItemView' module performs as follows: it subscribes to the activation command of the 'ItemView' module, it initiates an instance of the service until the results from the relevant 'AvailCheck' and 'CalcShipping' services are received. These events are correlated based on a variable over the id attribute. Only when there is a full match of the composite subscription, that is, all events are available, the 'ItemView' module will execute customers' invocation requests.

In a workflow, the output of a process can vary depending on the incoming event. For example, in Fig. 4 the online retailer might decide to give a post-order discount, if the shipment is delayed more than a specified duration. When a purchase order is placed, a shipment monitor is instantiated, which waits for the shipment event. When the shipment event is received, the shipment monitor will check the purchase agreement and if the shipment failed to match the agreed shipment date, a post-order discount is sent out. This business rule can be expressed by issuing the following subscription and publication:

```
SELECT *
 FROM e1 AND e2
WHERE e1.class = invoke, e1.service = MonitorShipment,
      e2.class = result, e2.service = ItemShipped,
      e1.id = e2.id, e2.time = e1.time + 10


  INSERT (class, service, id)
   VALUES (invoke, PostOrderDiscount, aaaaa)
```

The subscription is used to detect the condition where a post-order discount is to be issued and the publication is used to activate the post-order delivery module. The threshold that triggers a post order discount is 10 days from the order date.

A content-based publish/subscribe system not only efficiently implements a workflow, but it also simplifies reorganizing the workflow when required. For example, in Fig. 4, the retailer may decide to invoke a new service to verify the age of the consumers against the approved limit before activating the item view. The modification, shown with the dotted lines in the figure, can be readily

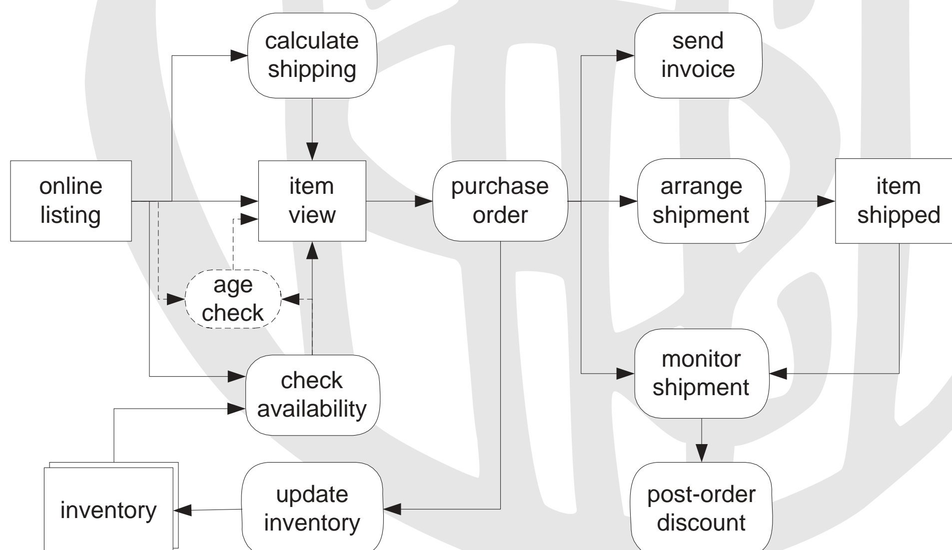


Figure 4 Example Workflow of an Online Sales Application.

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

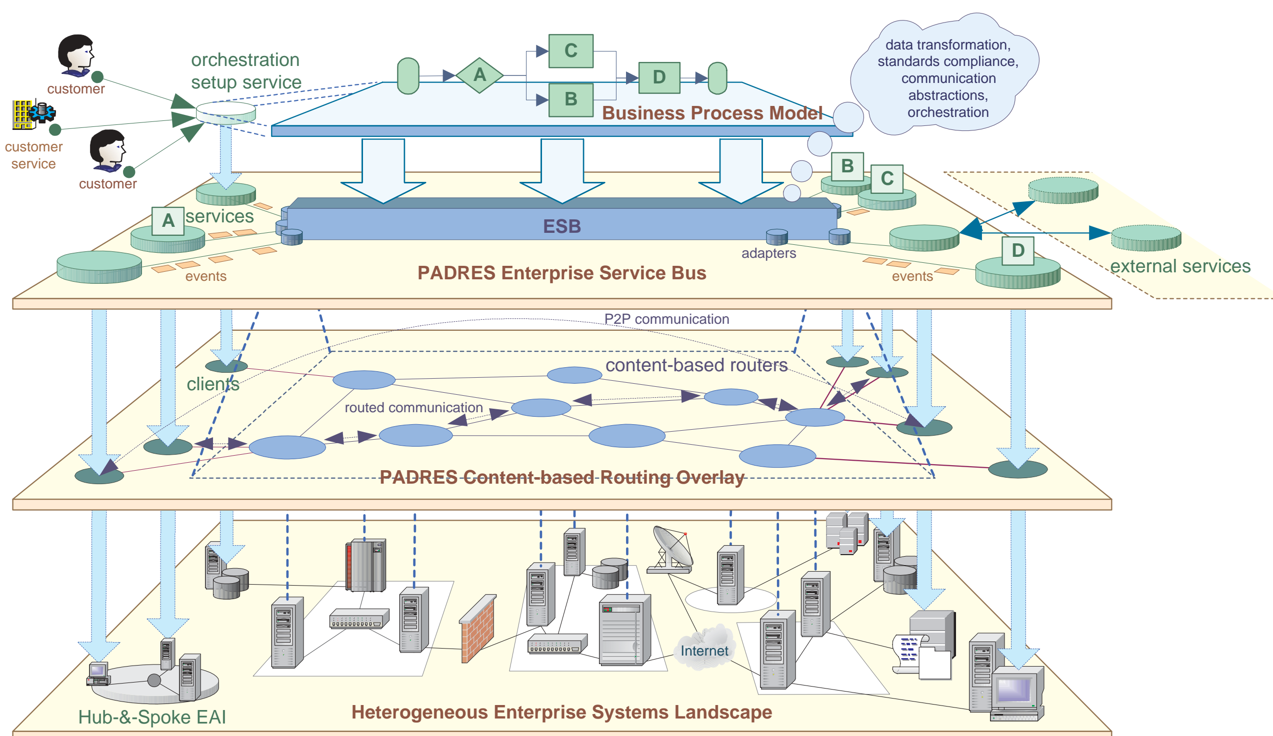


Figure 5 Building an Enterprise Service Bus with PADRES.

implemented by unsubscribing the previous subscription and invoking a new subscription as follows:

```
SELECT *
 FROM e1 AND e2 AND e3 AND e4
WHERE e1.class = invoke, e1.service = ItemView,
      e2.class = result, e2.service = AvailCheck,
      e3.class = result, e3.service = CalcShipping,
      e4.calss = result, e4.service = AgeCheck,
      e4.approve = YES, e1.id = e2.id,
      e2.id = e3.id, e3.id = e4.id
```

More generally speaking, Fig. 5 shows how PADRES can be used to implement an Enterprise Service Bus (ESB) that acts as an event processing network. PADRES enables a highly distributed event-driven architecture that supports many advanced ESB features. The distributed broker network connects applications across a large enterprise, even across the Internet. The content-based publish/subscribe system offers an event-driven approach that supports expressive workflow processing. Service orchestration is supported by content-based resource discovery and event-based monitoring, and the situation detection mechanism helps enforcing governance and compliance [10; 11; 15; 26].

5 Evaluation

We comprehensively evaluated the implementation of the PSQL language and the historic data access architecture [13]. We construct a 30 broker publish/subscribe overlay (with one database attached to each broker) with 10 publishers and 20 subscribers deployed across a 20 node cluster of 1.86 GHz machines with 4 GB of RAM. In the experiments, the publications are derived from stock traces from Yahoo! Finance [27]. Lacking real subscription traces, we generated subscriptions with predicates following a Zipf distribution in order to model locality among subscribers. Detailed deployment description and more results are given in [13].

Publication Storage

To store a publication in a database, the publication is converted into a SQL INSERT statement and then executed. The main factor here is the number of attributes in the publication. Figure 6 shows that the storage time (averaged over 5000 publications for each data point) increases linearly with the number of predicates.

Subscription Processing

A historic subscription arriving at a database is converted into an SQL SELECT statement and executed. Then, each query result is transformed into a publication message and published. Figure 7 quantifies this time up to but excluding sending the publications. We see that the processing time increases roughly linearly with the number of subscription predicates and the result set size.

Composite Subscription Routing

We evaluate the composite subscription routing protocols: simple, topology-based, and adaptive routing. A single composite subscription is issued that correlates data from eight sources: seven publishers publishing at

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

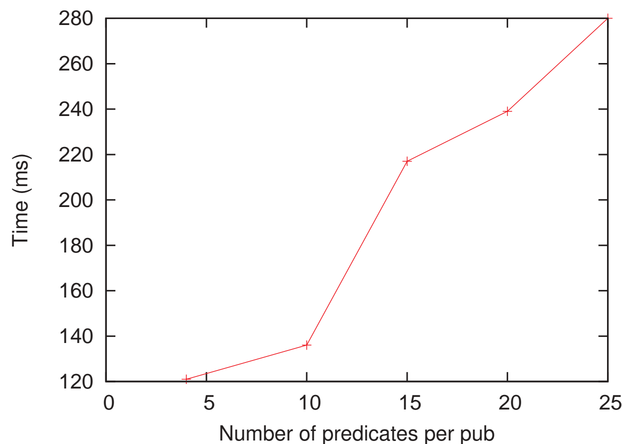


Figure 6 Publication Stortage Time.

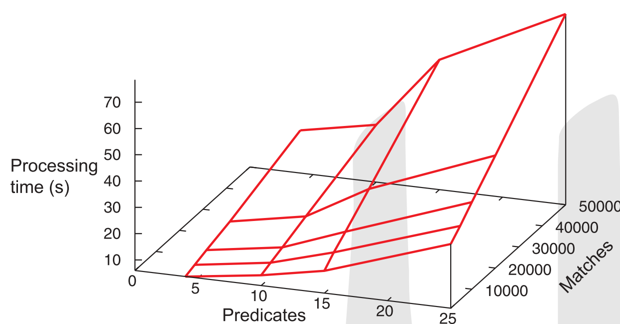


Figure 7 Query Processing Time.

rates from 50 to 500 msg/min, and one database. We measure the network traffic under different routing policies.

Figure 8 shows the outgoing traffic at each broker in the overlay. In simple routing, a composite subscription is split into atomic subscriptions at the broker that first receives the composite subscription from a subscriber. In this case, broker B0 evaluates the composite subscription and filters out unmatched publications. Since filtering does not occur until the last broker along the paths from publishers to subscriber, network traffic is high.

In topology-based routing, a composite subscription is forwarded through the broker network as a whole until it reaches a *join point broker* where its atomic subscriptions are forwarded in different directions in the topology. Brokers B6, B7, B8 and B10 are the join point brokers, and we observe in Fig. 8 that filtering occurs at those brokers.

The adaptive routing algorithm determines the composite detection location based on potential publication traffic. In this scenario, publishers with high publication rates connect to brokers B24, B27, and B20, and hence it is desirable to detect composite subscriptions near them. The results in Fig. 8 show that the adaptive algorithm reduces traffic by an average of about 66% and 43% compared to simple and topology-based routing, respectively. These savings are also enjoyed by all brokers downstream of the join points. The average notification delay in topology-based routing is about 0.1 s, which the adaptive algorithm manages to reduce by about 48% by filtering out messages early in the network and hence reduce queuing and routing delays.

We further evaluate adaptive routing in a scenario where conditions change over time. Publication rates are modified during the experiment: heavy publishers reduce their rates from 400 to 50 msg/min, and others increase their rates from 100 to 500 msg/min. Under the new workload, the original join point brokers initially chosen by the adaptive approach may no longer be optimal.

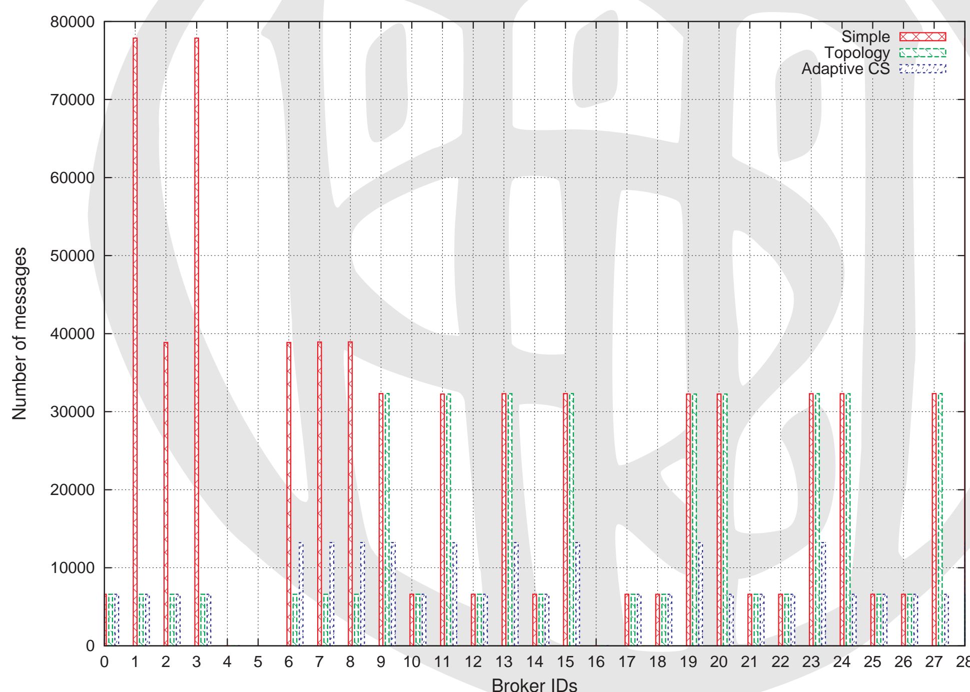


Figure 8 Composite Subscription.

This article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

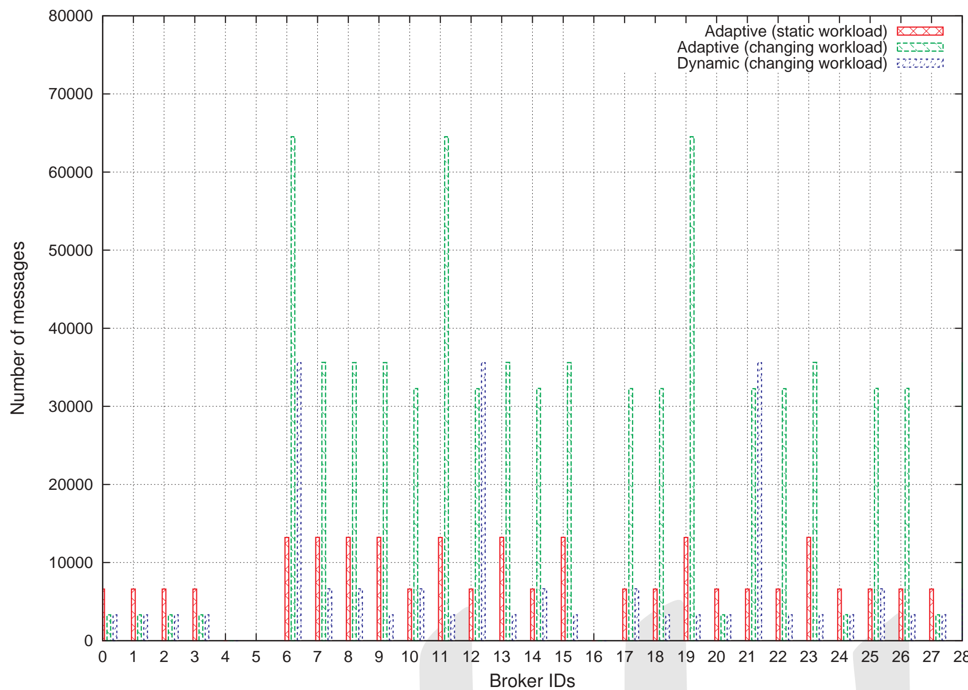


Figure 9 Dynamic Workload.

Figure 9 shows traffic with the changing workload when the join points remain at their optimal locations as determined during subscription routing (the adaptive case), and when they are allowed to react to the changing workload (the dynamic case). For comparison, the figure also replots from Fig. 8 the adaptive results under a static workload. In all cases, we only measure the traffic after the workload has changed.

We see that when the join points do not move (the adaptive cases), a change in the publication workload increases the overall traffic by about 220% since the join points are no longer optimal. However, by moving the join points to brokers B26, B18 and B22, where more publications are being generated, the dynamic algorithm reduces the total network traffic by about 72% compared to the case when the join points remain fixed despite changing conditions. Again, the traffic reductions are also enjoyed by all brokers downstream of the new join points.

6 Conclusions

This paper gave an overview of the PADRES content-based publish/subscribe system and highlighted its event processing network capabilities. The PSQL language provides a SQL-like interface to subscribe to historic and future publications. The language fully retains content-based publish/subscribe semantics and features, and can express filtering constraints, aggregation functions, projections, and correlations (joins) across any combination of future and historic data in a manner that preserves publish/subscribe decoupling and anonymity. To exemplify how content-based publish/subscribe can be used in practice for developing event-based applications, we presented a detailed discussion of example applications that benefit from the content-based nature of the paradigm and also take advantage of its scalability and robustness features. This paper also illustrated how publish/subscribe and PADRES serve the development of event-based applications.

References

[1] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. *Design and evaluation of a wide-area event notification service.* In: ACM Trans. on Computer Systems 19(3), 2001.

[2] B. Chandramouli, J. Xie, and J. Yang. *On the database/network interface in large-scale publish/subscribe systems.* In: Proc. of the 28rd Int’l Conf. on Very Large Data Bases, 2002.

[3] S. Chandrasekaran, and M. J. Franklin. *Streaming queries over streaming data.* In: Proc. of the ACM SIGMOD Int’l Conf. on Management of Data, 2006.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: *A scalable continuous query system for Internet databases.* In: SIGMOD Record 29(2), 2000.

[5] A. Cheung and H.-A. Jacobsen. *Dynamic Load balancing in distributed content-based Publish/Subscribe.* In: Proc. of the Int’l Middleware Conf., Melbourne, Australia, 2006.

[6] G. Cugola, E. D. Nitto, and A. Fuggetta. *The JEDI event-based infrastructure and its application to the development of the OPSS WFMS.* In: IEEE Trans. on Software Engineering 27(9), 2001.

[7] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. *Filtering algorithms and implementation for very fast publish/subscribe systems.* In: Proc. of the ACM SIGMOD Int’l Conf. on Management of Data 30(2), 2001.

[8] T. Fawcett and F. Provost. *Activity monitoring: Noticing interesting changes in behavior.* In: Proc. on the Fifth ACM SIGKDD Int’l Conf. on Knowledge Discovery and Data Mining, San Diego, CA, 1999.

[9] L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. *Engineering event-based systems with scopes.* In: Proc. of the 16th European Conf. on Object-Oriented Programming, 2002.

This article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

[10] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. *Distributed automatic service composition in large-scale systems.* In: Distributed Event-Based Systems (DEBS), 2008.
[11] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. *Transactional mobility in distributed content-based publish/subscribe systems.* In: Int'l Conf. on Distributed Computing Systems, Montreal, Canada, 2009.
[12] G. Li and H.-A. Jacobsen. *Composite subscriptions in content-based publish/subscribe systems.* In: Proc. of the Int'l Middleware Conf., Grenoble, France, 2005.
[13] G. Li, V. Muthusamy, and H.-A. Jacobsen. *Subscribing to the past in content-based publish/subscribe.* Technique Report, CSRG-585, University of Toronto, 2008.
[14] G. Li, S. Hou, and H.-A. Jacobsen. *A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams.* In: Int'l Conf. on Distributed Computing Systems, Columbus, Ohio, 2005.
[15] G. Li, V. Muthusamy, and H.-A. Jacobsen. *A distributed service oriented architecture for business process execution.* Technique Report, CSRG-584, University of Toronto. 2008.
[16] G. Li, V. Muthusamy, and H.-A. Jacobsen. *Adaptive content-based routing in general overlay topologies.* In: Proc. of the Int'l Middleware Conf., Levuen, Belgium, 2008.
[17] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. *Network intrusion detection.* In: IEEE Network 8(3):26–41, 1994.
[18] G. Mühl. *Generic constraints for content-based publish/subscribe systems.* In: Proc. of the 6th Int'l Conf. on Cooperative Information Systems, Trento, Italy, 2001.
[19] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. *Exploiting IP multicast in content-based publish-subscribe systems.* In: Proc. of the Int'l Middleware Conf., 2000.
[20] PADRES. http://padres.msrg.toronto.edu.
[21] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS: *Fast filtering of graph-based metadata.* In: Proc. of the 14th Int'l Conf. on World Wide Web, Chiba, Japan, 2005.
[22] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: *Content-based filtering and aggregation of blogs and RSS feeds.* In: Proc. of the 4th USENIX Symp. on Networked Systems Design & Implementation, 2007.
[23] C. Schuler, H. Schuldt, and H. J. Schek. *Supporting reliable transactional business processes by publish/subscribe techniques.* In: TES, 2001.
[24] R. Sherafat Kazemzadeh and H.-A. Jacobsen. *Reliable and highly available distributed publish/subscribe service.* In: IEEE Symp. on Reliable Distributed Systems, 2009.
[25] A. Wun and H.-A. Jacobsen. *A policy management framework for content-based publish/subscribe middleware.* In: Proc. of the Int'l Middleware Conf., Newport Beach, CA, 2007.
[26] W. Yan, S. Hu, V. Muthusamy, H.-A. Jacobsen,and L. Zha. *Efficient event-based resource discovery.* In: Distributed Event-Based Systems (DEBS), 2009.
[27] http://research.msrg.utoronto.ca/Padres/DataSets.

Received: May 15, 2009

Hans-Arno Jacobsen holds the Bell University Laboratories Chair in Software. He is a Professor of Electrical and Computer Engineering and Computer Science at the University of Toronto, where he leads the Middleware Systems Research Group. His principal areas of research include the design and the development of middleware systems, distributed systems, and information systems. His current research focus lies on publish/subscribe, content-based routing, event processing, and aspect-orientation.

Address: University of Toronto, 10 King's Colleg Road, M5S 3G4 Toronto, Canada,
e-mail: jacobsen@eecg.toronto.edu

Vinod Muthusamy is a Ph. D. candidate in the Middleware Systems Research Group in the Department of Electrical and Computer Engineering at the University of Toronto. His research interests include publish/subscribe systems and distributed workflow processing. Vinod holds a Bachelor degree from the University of Waterloo and a Master degree from the University of Toronto.

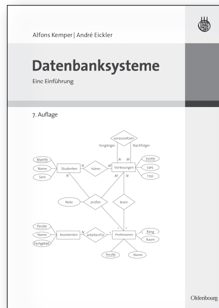
Address: University of Toronto, 10 King's Colleg Road, M5S 3G4 Toronto, Canada,
e-mail: vinod@eecg.toronto.edu

Guoli Li is a Ph. D. candidate in the Department of Computer Science at the University of Toronto. Her research focuses on historic data access in publish/subscribe. Guoli received her Master degree in Computer Science from the University of Toronto in 2005. She received a Master degree in Electronic Engineering from Xi'an Jiaotong University in 2002 and a Bachelor's degree in Information Techniques from the same university in 1999.

Address: University of Toronto, 10 King's Colleg Road, M5S 3G4 Toronto, Canada,
e-mail: gli@cs.toronto.edu

article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.

Einführung in moderne Datenbanksysteme



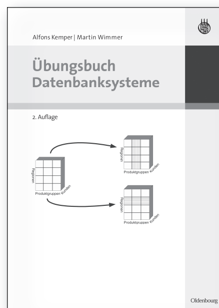
Alfons Kemper,
André Eickler

Datenbanksysteme
Einführung

7., aktualisierte und
erweiterte Auflage 2009
718 S. | Flexcover

€ 39,80
ISBN 978-3-486-59018-0

Dieses Buch vermittelt eine systematische und umfassende Einführung in moderne Datenbanksysteme. Einen Schwerpunkt der nunmehr 7. Auflage bilden die fortschrittlichen Anwendungen von Datenbanken im Internet sowie im betriebswirtschaftlichen Data Warehouse für Decision Support-Anfragen und das Data Mining. Neuere Entwicklungen wie XML und Web-Services werden ausführlich behandelt. Alle Konzepte werden an einer durchgehenden Beispielanwendung veranschaulicht. Jedes Kapitel enthält zahlreiche Übungsaufgaben.



Alfons Kemper,
Martin Wimmer

Übungsbuch Datenbanksysteme

2., aktualisierte und
erweiterte Auflage 2009
445 S. | Broschur mit DVD

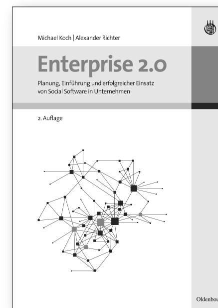
€ 29,80
ISBN 978-3-486-59001-2

Das Übungsbuch zu den „Datenbanksystemen“, in dem die Lösungen zu den im Buch vorgestellten Aufgaben gezeigt und erklärt werden. Umfangreiche Materialien auf DVD ergänzen die Buchinhalte und geben dem Leser die Möglichkeit, den Stoff zu verinnerlichen.

Oldenbourg

Bestellen Sie in Ihrer Fachbuchhandlung oder direkt bei uns:
Tel: 089/45051-248 · Fax: 089/45051-333 · verkauf@oldenbourg.de
www.oldenbourg.de

Erfolgreiche Praxis von Web 2.0



Michael Koch,
Alexander Richter

Enterprise 2.0

2., aktualisierte und
erweiterte Auflage
2009
275 S. | Flexcover

€ 39,80
ISBN 978-3-486-59054-8

Planung, Einführung und erfolgreicher Einsatz von *Social Software* in Unternehmen.

Die Verwendung von Web 2.0-Techniken und entsprechenden Werkzeugen birgt großes Potential für ein Unternehmen. Dieses Potential aufzuzeigen und nutzbar zu machen ist das Ziel des vorliegenden Buches.

Nach einer Einführung in die Thematik werden die wichtigsten Softwaregattungen und deren Anwendungsfelder im betrieblichen Umfeld vorgestellt. Die Beschreibungen sind dabei jeweils mit Beispielen und Handlungsleitfäden illustriert. Fallstudien aus 15 Organisationen unterstützen den Leser dabei, sich einen Überblick zu verschaffen oder sich Anregungen zu holen, wie man ganz konkrete Szenarien - z. B. Teamarbeit oder Informationsmanagement - im Unternehmen unterstützen kann.
Nach einer ausführlichen Diskussion der wichtigsten Herausforderungen beim Einsatz von *Social Software* wird das Buch mit einer Diskussion neuer Konzepte wie *Semantic Web*, *Virtuelle Welten* und *Ubiquitäre Benutzungsschnittstellen* abgerundet.

Oldenbourg

Bestellen Sie in Ihrer Fachbuchhandlung oder direkt bei uns:
Tel: 089/45051-248 · Fax: 089/45051-333 · verkauf@oldenbourg.de
www.oldenbourg.de

This article is protected by German copyright law. You may copy and distribute this article for your personal use only. Other use is only allowed with written permission by the copyright holder.