Lecture Notes in Computer Science 779

Matthias Jarke Janis Bubenko
Keith Jeffery (Eds.)

Advances in Database Technology — EDBT '94

4th International Conference
on Extending Database Technology
Cambridge, United Kingdom, March 1994
Proceedings

Springer-Verlag

Matthias Jarke Janis Bubenko
Keith Jeffery (Eds.)

Advances in Database Technology — EDBT '94

4th International Conference
on Extending Database Technology
Cambridge, United Kingdom, March 28-31, 1994
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Matthias Jarke
RWTH Aachen
Ahornstrasse 55, D-52056 Aachen, Germany

Janis Bubenko
SISU-ISE
Isafjordsgatan 26, S-1250 Kista, Sweden

Keith Jeffery
SERC Rutherford Appleton Laboratory
Chilton, Didcot, Oxfordshire OX11 0QX, United Kingdom

CR Subject Classification (1991): H.2, E.2, D.3.3, F.4.1, H.5.1, I.2.1, I.2.4

ISBN 3-540-57818-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57818-8 Springer-Verlag New York Berlin Heidelberg

CIP data applied for

Typesetting: Camera-ready by author
SPIN: 10131950 45/3140-543210 - Printed on acid-free paper

Foreword

The fourth international conference on Extending Data Base Technology (EDBT 94) was held in St. John's College (Cambridge, UK), March 28-31, 1994. After successful events in Venice (1988 and 1990) and Vienna (1992), the bi-annual EDBT has established itself as the premier European database conference. The conference provides an international forum for the presentation of new extensions to database technology through research, development, and application.

This proceedings volume contains the scientific papers of the conference. The programme committee faced the difficult task of selecting 31 papers among more than 180 submissions from 28 different countries. A major theme seemed to be the integration of database technology as a core technology in the larger context of large-scale and distributed computing systems. This problem was addressed from many different angles, ranging from novel view concepts to data access in mobile computing.

EDBT is the flagship conference of a broader initiative which strives to maintain the traditionally strong role of Europe in database research, and to improve the transition from research to practice. Related activities include summer schools organized by the EDBT Foundation, the European Network of Excellence in Databases and Information Retrieval (IDOMENEUS), and ongoing collaboration with the database journal *Information Systems.*

For the first time, EDBT 94 was held jointly with an industrial conference, the annual meeting of the British Data Management Specialist Group. The relationship between database research and practice was also the topic of a keynote address by Chris Stone, president of the Object Management Group, and of a panel introduced by Andrew Herbert, a pioneer in distributed computing environments. The conference programme was augmented by tutorials on interoperability, multimedia, object data models, and object-oriented databases by Michael Brodie, Stavros Christodoulakis, Moira Norrie and Hans Schek, and Roberto Zicari, respectively.

Many people deserve special thanks for their efforts to make EDBT 94 a success. Thanks are in particular due to the "British team" which admirably covered all aspects of conference organization, especially to the conference secretary Anna Duckworth, and to the members of the international programme committee, technically supported by Christoph Quix and Martin Staudt. Last not least, we are grateful to the sponsors for their enthusiasm despite difficult economic times.

Cambridge, March 1994 Matthias Jarke, Janis Bubenko, Keith Jeffery

Sponsorship

Promoted by EDBT Foundation, UK Database Foundation,
IDOMENEUS Network of Excellence

Sponsored by British Computer Society
In cooperation with IEEE, CEPIS

Organization

Conference Chairman: Janis Bubenko (SISU, Sweden)
Program Committee Chairman: Matthias Jarke (RWTH Aachen, Germany)

Organising Committee

K. G. Jeffery (SERC Rutherford Appleton Laboratory, Chairman)

A. Duckworth (BCS)
M. S. Jackson (Wolverhampton)
R. G. Johnson (Birkbeck)
J. Mabon (DMSG)
B. J. Read (SERC)
R. Williams (DMSG)
W. A. Gray (Cardiff)
A. J. Jenkins (DMSG)
J. Kennedy (Napier)
K. Moody (Cambridge)
G. Sharman (IBM)
R. F. Winterson (DMSG)

Regional Co-ordinators

R. Andersen (Norway)
J. Fong (Hong Kong)
M. Kersten (Netherlands)
M. Leonard (Switzerland)
S. Nishio (Japan)
A. Pirotte (Belgium)
S. Sa (China)
G. Schlageter (Germany)
C. K. Tan (Singapore)
Y. Vassiliou (Greece)
R. Carapuca (Portugal)
J. B. Grimson (Ireland)
K.-C. Lee (Taiwan)
B. G. Lundberg (Sweden)
M. E. Orlowska (Australia)
F. Plasil (Czechoslovakia)
F. Saltor (Spain)
D. Shasha (USA)
L. Tucherman (Brazil)

EDBT Foundation Consultants

S. Ceri (Milan)
J. Schmidt (Hamburg)
M. Missikoff (Rome)

Program Committee

M. Jarke (Germany – chairman)
M. Agosti (Italy)
E. Bertino (Italy)
A. Borgida (USA)
M. Brodie (USA)
J. Clifford (USA)
H. P. Frei (Switzerland)
G. Gottlob (Austria)
V. Jagadish (USA)
L. Kalininchenko (Russia)
M. Lenzerini (Italy)
P. Loucopoulos (United Kingdom)
F. Matthes (USA)
J. Mylopoulos (Canada)
A. Olive (Spain)
M. Papazoglou (Australia)
A. Reuter (Germany)
T. Risch (Sweden)
T. Rose (Canada)
M. Scholl (Germany)
D. Shasha (USA)
A. Solvberg (Norway)
M. Stonebraker (USA)
B. Thalheim (Germany)
J. Widom (USA)

S. Abiteboul (France)
R. Bayer (Germany)
J. Bocca (Chile/United Kingdom)
J. Bubenko (Sweden)
M. Carey (USA)
M. Freeston (Germany)
H. Garcia-Molina (USA)
P. Gray (United Kingdom)
K. G. Jeffery (United Kingdom)
H. Kangassalo (Finland)
F. Lochovsky (Hong Kong)
L. Mark (Denmark/USA)
G. Moerkotte (Germany)
S. Nishio (Japan)
M. E. Orlowska (Australia)
A. Pirotte (Belgium)
R. van de Riet (Netherlands)
C. Rolland (France)
H. Schek (Switzerland)
T. Sellis (Greece)
E. Simon (France)
A. Stogny (Ukraine)
K. Subieta (Poland)
Y. Vassiliou (Greece)
J. Zlatuska (Czech Republic)

Additional Referees

S. Al-Naemi
P. Bayer
S. Blott
J. Brunet
R. Colomb
A. Deacon
T. Eiter
B. Freitag
F. Gire
T. Haplin
M. Hornick
M. Jeusfeld
H. Kaufmann
M. Kogalovsky
R. Kramorenko
S. Kuznetsov
L. Lin
J. Maier
M. Melucci
I. S. Mumick
W. Nejdl
E. Omodeo
I. Petrounias
B. Polyachenko
C. Rich
S. Salza
S. Schwer
A. Shrufi
W. W. Song
S. Sripada
B. Thedoulidis
A. Tuzhilin
H. Waechter
A. Wickler
B. Woerner
M. Yoshikawa
R. Zink

S. A. Ananevsky
W. Becker
A. Bouguettaya
F. Bry
G. Costa
A. Delis
M. Eitler
K. Froeschl
H. Grabner
K. Harumoto
R. Hull
I. Jurisica
J. Kempe
V. Kolinko
J. Krogstie
C. Laasch
A. Listl
G. Mamier
Z. Mikal
V. Murthy
M. C. Norrie
M. W. Orlowski
P. Pistor
N. Prakash
J.-P. Richter
G. Santucci
A. P. Sexton
C. O. Shum
C. Souveyet
M. Stumptner
T. Topaloglou
R. Unland
X. Y. Wang
S. Wiesener
M. F. Wyne
P. Zabback

S. Azarov
J. Besancenot
S. E. Bratsberg
C. Cauvet
P. Creasy
V. Dhar
G. Fahl
D. Georgakopoulos
J. A. Gulla
C. Hasse
G. Höfling
G. Kappel
A. Kemper
M. Koubarakis
W. Kuhn
R. A. Lehn
G. Lohman
R. Marti
A. Montesi
I. Narang
B. Novikov
J. Overbeck
D. Plexousakis
E. Rahm
S. Rizzo
U. Schmidt
E. Shekita
M. Sköld
G. Specht
A. Swami
R. Torlone
V. Vianu
B. Wangler
J. P. Wilks
B. Wüthrich
V. Zadorozhny

M. Baaz
C. Bettini
Y. Breitbart
V. K. Chaudhri
A. Croker
P. Drew
N. Frakash
U. Geuder
M. Haerdtner
L. Hermosilla
M. Iwamuro
K. Karlapalem
H. Knolle
W. Kowarschick
G. Kuper
Q. Li
P. J. Lupton
I. McLaren
K. Moody
B. Nebel
J. K. Obermaier
M. Pawlowski
R. Pollak
A. Reiser
M. A. Sakchnuk
F. Schwenkreis
S. Shimojo
R. Smith
P. A. Spruit
Z. Tari
M. Tresch
P. Vogel
M. Werner
G. Willumsen
V. Yaremenko
Y. Zhang

Contents

Invited Papers

Object Views

Intelligent User Interfaces

Distributed Information Servers

Transaction Management

Information Systems Design and Evolution

Semantics of Extended Data Models

Accessing New Media

Join Algorithms

Query Optimization

Multimedia Databases

The Object Management Group Standardization of Object Technology

Christopher M. Stone
President & CEO
Object Management Group, Inc.
Framingham, Massachusetts, USA

Abstract

Object technology (OT) which has been described by some as a remarkable paradigm shift in computing is in reality a technology that allows people to think, literally, in terms of nouns and verbs as they assemble software programs. Too often, contemporary software development is characterized as an artform with mystical heritage.

The Object Management Group, with over 330 members, has become the industry focal point for the development of interface standards in distributed computing. The OMG focuses its attention on creating specifications for distributed applications using object technology. The group's first specification, the *Object Request Broker*, has already been endorsed and committed to by over one hundred (100) companies. OMG is funded by the top computer, software, networking, and end user organizations in the information processing business.

Mr. Stone will discuss the rule of OT, the OMG, and the impact on the future of client server computing and distributed applications.

Databases in Distributed Systems: The New Frontier

– Extended Abstract –

Andrew Herbert

Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge CB3 ORD, United Kingdom

1 Introduction

Database technology is rooted in the 1960s mainframe culture. It has been very successful at enabling complex applications to share and manage enormous bodies of data. However, for many users the future resides with distributed computing as previously separate applications are linked together to provide new services in support of new business goals.

Within a distributed system there is still a role for databases per se, but not as the focus of the system. Processing will be shared with small personal computers providing the user interface and departmental machines providing integration services and local data caching. This is too complex a system to model as a single "database".

Research on distributed databases, multi-databases and federated databases will enable integration of databases to a greater or lesser extent, but does not contribute towards problems such as applications interoperability and distributed system management. For these functions "distributed objects" (see for example the work of the Object Management Group) show greatest promise.

Objects provide both modularity and abstraction; they encourage genericity thereby reducing special cases and complexity. In a distributed sysem object management can include functions such as object migration to balance loads and reduce latency, object replication for fault tolerance and enhanced availability.

Distributed objects have to be modelled, objects providing specific functions have to be located, objects have to deliver consistent results. These are all echos of database concepts (schema, query, transaction), but applied to a sea of distributed objects acting as a logical repository rather than a single physical one. The priority for database research must be to discover how to unbundle the ingredients of a successful database and make them work effectively in the distributed context. In a nutshell: data management - a successful past and a glorious future, but not in dumb old databases.

2 Distributed Systems

In the near future, most industrial computing will be based on a distributed computing architecture. A key notion is that of services. A service is some collection

of software and data that delivers a business function, for example billing for a telecommunication operator, customer account management for a bank or utility. To its clients, a service has a well defined interface which provides operations to inform the service of changes in the real world (the crediting of an account) and to obtain information from the service (e.g. a client's current balance). To the client the service is just an object which embodies a set of business functions. How those functions are implemented in terms of data and applications is a matter for the service provider.

The interface concept is crucial - it enables the service implementation to evolve without disrupting clients, even to be replaced totally by an alternative technology. It gives a view of an information technology system in terms of how it supports the operations of the enterprise that owns it.

Services can be large, as illustrated in the preceding examples, or very small: each window on a screen can be a service, each element within a window can be a service. From a service point of view granularity is of no issue. From an implementation point of view it is important to choose an appropriate infrastructure (i.e. Object Request Broker) to the kind of object being used. However the programming interfaces to each infrastructure for managing objects and invoking operations can be the same, and infrastructures at different granularities can be made to interwork with one another providing a uniform sea of distributed, remotely accessible objects.

3 Objects Need Management

Databases arose when programmers recognized that ad hoc schemes for data management were out of control and that carefully designed mechanisms and programming abstractions were called for. Distributed objects are running into a comparable problem.

3.1 Trading for Services

How do you find a service? Perhaps you know its type - i.e. the operations you want to invoke. Then you'd like to be able to query the distributed system to find objects that provide an interface of the right type (or a suitable subtype). To exercise some control over selecting from the available objects we should assign attributes to them such as location, owner, cost of use. Immediately we have asked for a schema function (so that types can be described and compared), and a query function. If you are new to a system you might want to find out what operations are available in terms of more abstract descriptions of the kind of behaviour required of a service. This implies that schemas should include behavioural specifications as well as structural informaion such as operation signatures.

3.2 Schemas for Distributed Objects

The schema function is complicated by the fact that whilst we may be able to superimpose the same conceptual object model on our services there are many implementations to choose from (e.g various flavours of OMG CORBA, OSF DCE RPC, ISO GDMO notation, various OODBMS notations) and as part of deciding if a client can interact with a candidate server we have to decide if appropriate adaptors can be put in the path to make all necessary conversions. At the simplest level our types can be just names and we rely on administrators to define type relationships. We can automate comparison of operation signatures (i.e. function prototypes) using type checkers. Maybe we can automate some aspects of checking semantics (e.g. by comparing pre and post-conditions as part of operation specifications). One of the challenges of interface schema design is to capture as much information as possible about a service, how to use it and its infrastructure requirements. The schema representation should be part of the operational system and universally accessible (i.e. schemas are also "services").

3.3 Traders and Distributed Query

Giving objects attributes and querying those attributes suggests some sort of repository of attributes and object references will suffice (called a "trader" in ANSA). As systems grow we will want to combine traders to build up a composite trading space. Therefore we want a notion of querying that expresses not only what to look for in a particular trader, but also how to include information from other traders in the resolution of the query. This capability is likely to be the responsibility of the traders rather than their clients so that programmers see a single interface to a local trader as their window on the system, and that local trader maps the information available to it into a form suitable for its users. Different users might have quite different views of the same system.

3.4 Generalized Query

Since objects provide services, they can be asked questions. Therefore it would seem useful to extend the simple trading concept to include a wider notion of not only including attributed held in one or more repositories, but also data dynamically obtained from the object. This raises interesting questions about consistency and caching - whilst an object might satisfy a query now, it might not tomorrow. How do we represent (protect) the temporal validity of the result of a query?

3.5 Concurrency Control for Objects

Significant applications will invoke many services as they perform their function. They must be able to recover from failures of individual objects en route or to back out of activities that turn out to be redundant or impossible to complete. All this has to be managed in a concurrent world. A transaction function

would provide the application programmer with a tool for managing concurrency. However this is a transaction function that works on any object in a distributed system, not just those that choose to live in a database. It must therefore be capable of fine grained locking: it must support some form of nesting so that services can call other services within a transaction. Since objects can be very small it must be transaction function that can protect longer pieces of activity as well as individual operations, bringing in notions of workflow. Not all objects can be rolled back - once a message has been sent it cannot be recalled, instead a compensation must be issued. Therefore forward as well as backward recovery must be provided. It is an interesting question to debate how much application semantics (as captured in a interface specification) can be used to generate locking disciplines.

3.6 Persistence

Finally not all of the objects in a system need be active at once, and can migrate out from main memory to secondary storage - this is the persistence function. Here the challenge is transparency. Ideally the client of a persistent object should not be aware of it movement to and from disc. The object's manager however will probably want to exercise a great deal of control over its placement. Persistence might include replication (for fault tolerance). Persistence and transactions must interact correctly in the sense that only consistent versions of objects should be made stable.

4 Wither Databases

The DBMS community has made many contributions to computing. In fact, they are so good that they are of greater value outside DBMSs than inside them. Just one of the examples above is transactions.

Transactions should be a general purpose service available to all programs (database applications or not) to update any computing resource (database or not). This is all already happening. There are a number of research systems providing distributed transactions for objects (e.g. Argus at MIT, Arjuna at Newcastle University), in products like Transarc's ENCINA and in standards (e.g. the Transaction Service RFP process underway in the OMG).

Just about all of the capabilities associated hitherto with databases have significance to distributed systems, but ripped apart into a number of independently usable parts rather than as a monolithic whole

- general purpose queries over all computing resources
- query optimization
- workload optimizations (data and application migration)
- run time accessible schemas (for trading, on the fly "adaptor generation") - schema and systems evolution - transactions - persistence.

DBMSs qua DBMSs will continue to provide useful functionality, but lurking, together with their intimate applications, behind a service interface.

The challenge facing the DBMS community is that databases, as we know and love them, simply do not address all of the needs of a general purpose, distributed computing environment. They need to be generalized and broken down into smaller functions. This is what is being done under the name of distributed object management. If the DBMS folks don't do this work, others will, and are (e.g. the vendors and ISVs supporting the OMG).

The message is: don't be parochial. Topics such as query optimization, query language design, object modelling, concurrency control are all interesting in the database world, but not of wider significance. Consider the greater challenge of: distributed computing query optimization, distributed processing language design, distributed object modelling, distributed concurrency control. More challenging but vastly more profitable when the solution is reached since it then applies to all of computing.

Acknowledgements
The author has had valuable discussions with Mike Brodie of GTE, Gomer Thomas of Bellcore and Rob van der Linden of ANSA in preparing this paper.

Type Derivation Using the Projection Operation
(Extended Abstract)

Rakesh Agrawal Linda G. DeMichiel

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract. We present techniques for deriving types from existing object-oriented types using the relational algebraic projection operation and for inferring the methods that are applicable to these types. Such type derivation occurs, for example, as a result of defining algebraic views over object types. We refactor the type hierarchy and place the derived types in the type hierarchy in such a way that the state and behavior of existing types remain exactly as before. Our results have applicability to relational databases extended with object-oriented type systems and to object-oriented systems that support algebraic operations.

1 Introduction

In relational database systems, it is often useful to define *views* over sets of related data items for purposes of abstraction or encapsulation. Views are specified by using the standard algebraic query operations. Views are defined over relations, and the "type" of a view, like that of a relation, is implicitly given by the types of its attributes. The instantiation, or materialization, of a view is determined by the contents of those relations over which the view is defined.

Because of their usefulness in relational databases, views have also attracted considerable attention in object-oriented database systems. However, unlike in relational systems, types and type extents are often decoupled in object-oriented type systems. Thus it becomes important to separate two aspects of view operations: (1) the derivation of new types as a result of the view operation; (2) the manipulation of instances of the source types of the view to obtain the instances of the type derived by the view operation. It is the first of these that we shall address in this paper.

In object-oriented type systems, there are two aspects to such type derivation that must be considered: (1) The behavior of the derived type must be inferred—that is, it must be determined which of the methods that are applicable to the source types of the derived type are applicable to the new type itself. (2) The new type must be correctly inserted into the existing type hierarchy in such a way that existing types have both the same state and the same behavior as before the creation of the derived type.

We consider type derivation using the relational algebraic operations, focussing here only on the projection operation. Of the relational operations, pro-