O

OASIS

Serguei Mankovskii
CA Labs, CA Inc, Thornhill, ON, Canada

Synonyms

Organization for the advancement of structured information standards

Definition

OASIS is a non-for-profit consortium aiming at collaborative development and approval of open international, mainly XML-based, standards.

Key Points

OASIS was founded in 1993 under the name "SGML Open." The initial goal of the organization was to develop guidelines for interoperability among products using Standard Generalized Markup Language (SGML). In 1998 it changed name to OASIS to reflect on changing scope of its technical work.

OASIS consists of an open group of member organizations whose representatives work in committees developing standards, promoting standards adoption, product interoperability and standards conformance. In 2007 OASIS had 5,000 participants representing 600 organizations and individual members in 100 countries. OASIS is governed by a member-elected Board in an annual election process. The board membership is based on the personal merits of Board nominees.

OASIS process allows participants to influence standards that affect their business, contribute to standards advancement and start new standards. The process is designed to promote industry consensus. OASIS strategy values creativity and consensus over conformity and control. It relies on the market to determine the particular approach taken in the development of sometimes overlapping standards.

OASIS maintains collaborative relationships with the International Electrotechnical Commission (IEC), International Organization for Standardization (ISO), International Telecommunication Union (ITU) and United Nations Electronic Commission for Europe (UN/ECE), and National Institute of Standards and Technology (NIST).

Among major accomplishments of the OASIS are such influential of standards as a group of ebXML standards, SAML, XACML, WSRP, WSDM, BPEL, OpenDocument, DITA, DocBook, LegalXML and others.

Cross-references

- ▶ BPEL
- ▶ DITA
- ▶ DocBook
- ▶ ebXML
- ▶ eGovernment
- ▶ Emergency Management
- ▶ LegalXML
- ▶ oBIX
- ▶ Open CSA (SCA, SDO)
- ▶ OpenDocument
- ▶ SAML
- ▶ SOA-RM
- ▶ UDDI
- ▶ WS-Security
- ▶ WSDM
- ▶ WSRP

Recommended Reading

1. OASIS. Available at: http://www.oasis-open.org

Object Constraint Language

Martin Gogolla
University of Bremen, Bremen, Germany

Synonyms

OCL

Definition

The Unified Modeling Language (UML) includes a textual language called Object Constraint Language

(OCL). OCL allows users to navigate class diagrams, to formulate queries, and to restrict class diagrams with integrity constraints. From a practical perspective, the OCL may be viewed as an object-oriented version of the Structured Query Language (SQL) originally developed for the relational data model. From a theoretical perspective, OCL may be viewed as a variant of first-order predicate logic with quantifiers on finite domains only. OCL has a well-defined syntax [1,3] and semantics [2].

Key Points

The central language features in OCL are: navigation, logical connectives, collections and collection operations.

Navigation: The navigation features in OCL allow users to determine connected objects in the class diagram by using the dot operator ".". Starting with an expression `expr` of start class `C`, one can apply a property `propC` of class `C` returning, for example, a collection of objects of class `D` by using the dot operator: `expr.propC`. The expression `expr` could be a variable or a single object, for example. The navigation process can be repeated by writing `expr.propC.propD`, if `propD` is a property of class `D`.

Logical Connectives: OCL offers the usual logical connectives for conjunction (`and`), disjunction (`or`), and negation (`not`) as well as the implication (`implies`) and a binary exclusive (`xor`). An equality check (=) and a conditional (`if then else endif`) is provided on all types.

Collections: In OCL there are three kinds of collections: sets, bags, and sequences. A possible collection element can appear at most once in a set, and the insertion order in the set does not matter. An element can appear multiple times in a bag, and the order in the bag collection does not matter. An element can appear multiple times in a sequence in which the order is significant. Bags and sequences can be converted to sets with `->asSet()`, sets and sequences to bags with `->asBag()`, and sets and bags to sequences with `->asSequence()`. The conversion to sequences assumes an order on the elements. The arrow notation is explained in more detail below.

Collection Operations: There is a large number of operations on collections in OCL. A lot of convenience and expressibility is based upon them. The most important operations on all collection types are the following: `forAll` realizes universal quantification, `exists` is existential quantification, `select` filters elements with a predicate, `collect` applies a term to each collection element, `size` determines the number of collection elements, `isEmpty` tests for emptiness, `includes` checks whether a possible element is included in the collection, and `including` builds a new collection `including` a new element.

In addition to the central language features, OCL also has special operations available only on particular collection, e.g., the operation `at` on sequences for retrieving an element by its position. All collection operations are applied with the arrow notation mentioned above. Roughly speaking, the dot notation is used when a property follows, i.e., an attribute or a role follows, and the arrow notation is employed when a collection operation follows.

Variables in collection operations: Most collection operations allow variables to be declared (possibly including a type specification), but the variable may be dropped if it is not needed.

Retrieving all Current Instances of a Class: Another important possibility is a feature to retrieve the finite set of all current instances of a class by appending `.allInstances` to the class name. In order to guarantee finite results `.allInstances` cannot be applied to data types like `String` or `Integer`.

Return types in collection operations: If the collection operations are applied to an argument of type `Set/Bag/Sequence(T)`, they behave as follows: `forAll` and exists returns a `Boolean`, `select` yields `Set/Bag/Sequence(T)`, `collect` returns `Bag/Bag/Sequence(T')`, size gives back `Integer`, `isEmpty` yields `Boolean`, `includes` returns `Boolean`, and `including` gives back `Set/Bag/Sequence(T)`.

Most notably, the operation `collect(...)` changes the type of a `Set(T)` collection to a `Bag(T')` collection. The reason for this is that term inside the `collect` may evaluate to the same result for two different collection elements. In order to reflect that the result is captured for each collection element, the result appears as often as a respective collection element exists. This convention in OCL resembles the same approach in SQL: SQL queries with the additional keyword `distinct` return a set; plain SQL queries without `distinct` return a bag. In OCL, the convention is similar: OCL expressions using the additional conversion `asSet()` as in `collect(...)->asSet()` return a set; plain `collect(...)` expressions without `asSet()` return a bag.

Cross-references

► Unified Modeling Language

Recommended Reading

1. OMG (ed.). OMG Object Constraint Language Specification. OMG, 2007. www.omg.org.
2. Richters M. and Gogolla M. On Formalizing the UML Object Constraint Language OCL. In Proc. 17th Int. Conf. on Conceptual Modeling. 1998, pp. 449–464.
3. Warmer J. and Kleppe A. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley, Reading, MA, 2003.

Object Data Models

Susan D. Urban[1], Suzanne W. Dietrich[2]
[1]Texas Tech University, Lubbock, TX, USA
[2]Arizona State University, Phoenix, AZ, USA

Synonyms

ODB (object database); OODB (object-oriented database); ORDB (object-relational database)

Definition

An object data model provides support for objects as the basis for modeling in a database application. An object is an instance of a class, which is a complex type specification that defines both the state of its instance fields and the behavior provided by its methods. Object features also include a unique object identifier that can be used to refer to the object, as well as the organization of data into class hierarchies that support inheritance of state and behavior. The term object data model encompasses the data model for both object-oriented databases (OODBs) and object-relational databases (ORDBs). OODBs use an object-oriented programming language as the database language and provide inherent support for the persistence of objects with typical database functionality. ORDBs extend relational databases by providing additional support for objects.

Historical Background

The relational data model was developed in the 1970's, providing a way to organize data into tables with rows and columns [4]. Relationships between tables were defined by the concept of foreign keys, where a column (or multiple columns) in one table contained a reference to a primary key value (unique identifier) in another table. The simplicity of the relational data model was complemented by its formal foundation on set theory, thus providing powerful algebraic and calculus-based techniques for querying relational data.

Initially, relational data modeling concepts were used in business-oriented applications, where tables provided a natural structure for the organization of data. Users eventually began to experiment with the use of relational database concepts in new application domains, such as engineering design and geographic information systems. These new application areas required the use of complex data types that were not supported by the relational model. Furthermore, database designers were discovering that the process of normalizing data into table form was affecting performance for the retrieval of large, complex, and hierarchically structured data, requiring numerous join conditions to retrieve data from multiple tables. Around the same time, object-oriented programming languages (OOPLs) were also beginning to develop, defining the concept of user-defined classes, with instance fields, methods, and encapsulation for information hiding [14].

The OOPL approach of defining object structure together with object behavior eventually provided the basis for the development of Object-Oriented Database Systems (OODBs) in the mid-1980's. The *Object-Oriented Database System Manifesto*, written by leading researchers in the database field, was the first document to fully outline the characteristics of OODB technology [1]. OODBs provided a revolutionary concept for data modeling, with data objects organized as instances of user-defined classes. Classes were organized into class hierarchies, supporting inheritance of attributes and behavior. OODBs differed from relational technology through the use of internal object identifiers, rather than foreign keys, as a means for defining relationships between classes. OODBs also provided a more seamless integration of database and programming language technology, resolving the impedance mismatch problem that existed for relational database systems. The impedance mismatch problem refers to the disparity that exists between set-oriented relational database access and iterative one-record-at-a-time host language access. In the OODB paradigm, the OOPL provides a uniform, object-oriented view of data, with a single language for accessing the database and implementing the database application.

The relational database research community responded to the development of OODBs with the *Third Generation Database System Manifesto*, defining the manner in which relational technology can be extended to support object-oriented capabilities [13]. Rowe and Stonebraker developed Postgres as the first object-relational database system (ORDB), illustrating an evolutionary approach to integrating object-oriented and relational concepts [10]. ORDB concepts parallel those found in OODBs, with the notions of user-defined data types, object tables formed from user-defined types, hierarchies of user-defined types and object tables, rows of object tables with internal object identifiers, and relationships between object tables that use object identifiers as references.

Today, several OODB products exist in the market, and most relational database products provide some form of ORDB support. The following section elaborates on the common features of object data models and then differentiates between OODB and ORDB modeling concepts.

Foundations

Characteristics of Object Data Models

An object is one of the most fundamental concepts of the object data model, where an object represents an entity of interest in a specific application. An object has state, describing the specific structural properties of the object. An object also has behavior, defining the methods that are used to manipulate the object. Each method has a signature that includes the method name as well as the method parameters and types. The state and behavior of an object is expressed through an object type definition, which provides an interface for the object. Objects of the same interface are collected into a class, where each object is viewed as an instance of the class. A class definition supports the concept of encapsulation, separating the specification of a class from the implementation of its methods. The implementation of a method can therefore change without affecting the class interface and the way in which the interface is used in application code.

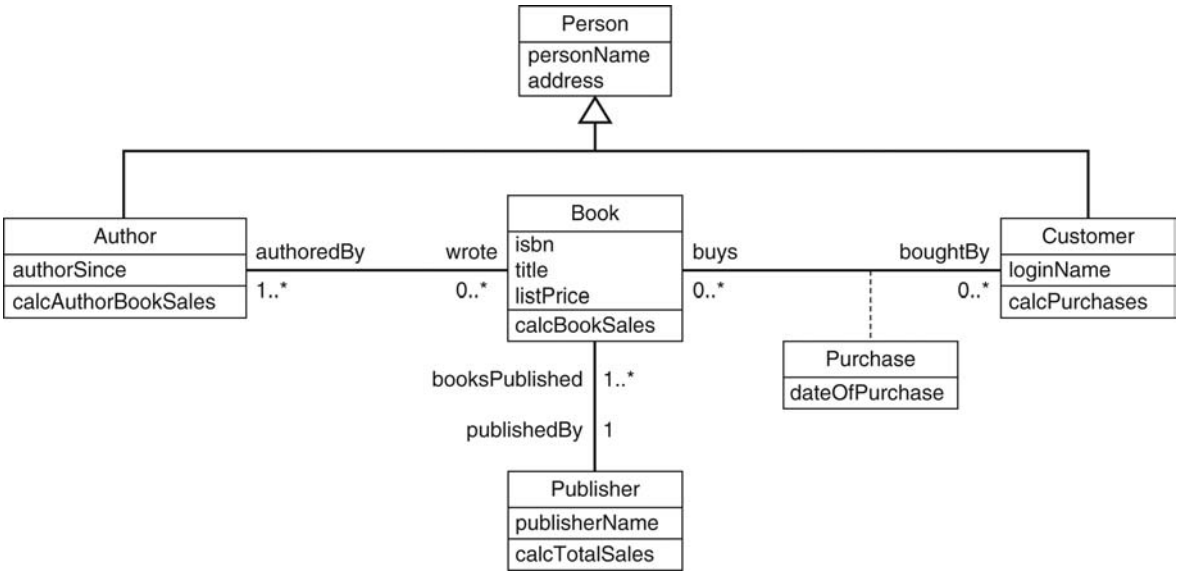
When an object of a class is instantiated, the object is assigned a unique, internal object identifier, or oid [6]. An oid is immutable, meaning that the value of the identifier cannot be changed. The state of an object, on the other hand, is mutable, meaning that the values of object properties can change. In an object data model, object identity is used as the basis for defining relationships between classes, instead of using object state, as in the relational model. As a result, the values of object properties can freely change without affecting the relationships that exist between objects. Object-based relationships between classes are referred to as object references.

Classes in an object model can be organized into class hierarchies, defining superclass and subclass relationships between classes. A class hierarchy allows for the inheritance of the state and behavior of a class, allowing subclasses to inherit the properties and methods of its superclasses while extending the subclass with additional properties or behavior that is specific to the subclass. Inheritance hierarchies provide a powerful mechanism to represent generalization/specialization relationships between classes, which simplify the specification of an object schema, as well as queries over the schema.

As an example of the above concepts, consider the Publisher application described in Fig. 1 using a Unified Modeling Language (UML) class diagram [11]. A Book is a class that is based on an object type that defines the state of a book (isbn, title, and listPrice), as well as the behavior of a book (the method calcBookSales for calculating the total sales of a book based on customer purchases). Publisher, Person, Author, and Customer are additional classes, also having state and behavior. Since authors and customers are specific types of people, the Author and Customer classes are defined to be subclasses of Person. Since personName and address are common to authors and customers, these attributes are defined at the Person level and inherited by instances of the Author and Customer classes. Furthermore, Author introduces additional state and behavior that is specific to authors, defining the date (authorSince) when an author first wrote a book as well as a method (calcAuthorBookSales) for calculating the total sales of the author's books. The Customer class similarly introduces state and behavior that is specific to a customer.

Relationships are also defined between the classes of the application:

- A book is authored by one or more authors; an author writes many books
- A book is published by one publisher; a publisher publishes many books
- A book is bought by many customers; a customer buys many books, also recording the date of each purchase



Object Data Models. Figure 1. The publisher object data model.

For each relationship, specific instances of each class are related based on the object identity of each instance. For example, a book will establish a relationship to the publisher of the book using the oid of the publisher. In the relational model, the publisher name would be used as a foreign key to establish the relationship. If the publisher name changes, then the change in name must be propagated to the book that references the publisher. In the object data model, such changes in state do not affect relationships between objects since the relationship is based on an immutable, internal object identity.

A generic object model, such as the one shown in Fig. 1, can be mapped to either an object-oriented data model or an object-relational data model. The following subsections use the Publisher application in Fig. 1 to illustrate and explain OODB and ORDB approaches to object data modeling.

Object-Oriented Data Model

An object-oriented database (OODB) is a term typically used to refer to a database that uses objects as a building block and an object-oriented programming language as the database language. The database system supports the persistence of objects along with the features of concurrency and recovery control with efficient access and an ad hoc query language.

The Object Data Standard [2] developed as a standard to describe an object model, including a definition language for an object schema, and an ad-hoc query language. The object model supports the specification of classes having attributes and relationships between objects and the behavior of the class with methods. The Object Definition Language (ODL) provides a standard language for the specification of an object schema including properties and method signatures. A property is either an attribute, representing an instance field that describes the object, or a relationship, representing associations between objects. In ODL, relationships represent bidirectional associations with the database system being responsible for maintaining the integrity of the inverse association. An attribute can be used to define a unidirectional association. If needed, the association can be derived in the other direction using a method specification. The decision is based on trade-offs of storing and maintaining the association versus deriving the inverse direction on demand.

Fig. 2 provides an ODL specification of the Publisher application. Each class has a named extent, which represents the set of objects of that type. The Author and Customer classes inherit from the Person class, extending each subclass with specialized attributes. The Book class has the isbn attribute that forms a key, being a unique value across all books. The association between Author and Book is represented as an inverse relationship, and the cardinality of the association is many-to-many since an author can write many books and a book can be written by multiple authors. The set collection type models multiple

```
class Person
(extent people)
{attribute string personName,
attribute AddressType address
};

class Author extends Person
(extent authors)
{attribute Date authorSince,
 relationship set<Book> wrote
    inverse Book::authoredBy,
 public float calcAuthorBookSales();
};

class Customer extends Person
(extent customers)
{attribute string loginName,
relationship set<Purchase> buys
    inverse Purchase::purchasedBy,
public float calcPurchases();
};

class Purchase
(extent purchases)
{attribute Date dateOfPurchase,
relationship Book bookPurchased
    inverse Book::boughtBy,
relationship Customer purchasedBy
    inverse Customer::buys;
);
```

```
class Book
(extent books,
key isbn)
{attribute string isbn,
attribute string title,
attribute float listPrice,
relationship set<Author> authoredBy
    inverse Author::wrote,
relationship set<Purchase> boughtBy
    inverse Purchase::bookPurchased,
relationship Publisher publishedBy
    inverse Publisher::booksPublished,
public float calcBookSales();
};


class Publisher
(extent publishers)
{attribute string publisherName,
relationship set<Book> booksPublished
  inverse Book::publishedBy,
public float calcTotalSales();
};
```

Object Data Models. Figure 2. ODL schema of the publisher application.

books and authors. Since the Purchase association class from Fig. 1 has an attribute describing the association, Purchase is modeled in ODL using reification, which is the process of transforming an abstract concept, such as an association, into a class. As a result, the Purchase class in Fig. 2 represents the many-to-many association between Book and Customer. Each instance of the Purchase class represents the purchase of a book by a customer. The Purchase class has the dateOfPurchase instance field, as well as two relationships indicating which Book (bookPurchased) and which Customer (purchasedBy) is involved in the purchase. The inverse relationships in Book (boughtBy) and Customer (buys) are related to instances of the Purchase class.

This ODL specification forms the basis of the definition of the object schema within the particular OOPL used with the OODB, such as C++, Java, and Smalltalk. The specification of the schema and the method implementation using a given OOPL is known as a language binding. In some OODB products, the ODL specification of the properties of the class are used to automatically generate the definition of the schema for the OOPL being used.

The standard also includes a declarative query language known as the Object Query Language (OQL). The OQL is based on the familiar select-from-where syntax of SQL. The select clause defines the structure of the result of the query. The from clause specifies variables that range over collections within the schema, such as a class extent or a multivalued property. The where clause provides restrictions on the properties of the objects that are to be included in the result. Object references are traversed through the use of dot notation for single-valued properties and through the from clause for multivalued properties.

Consider a simple query that finds the name of a publisher of a book given its isbn:

```
select b.publishedBy.publisherName
from books b
where b.isbn = "0-13-042898-1";
```

This OQL query looks quite similar to SQL. In the from clause, the alias b ranges over the books extent.

The where clause locates the book of interest. The select clause provides a path expression that navigates through the publishedBy single-valued property to return the name of the publisher.

Consider another query that finds the title and sales for books published by Springer-Verlag:

```
select title: b.title, sales: b.calcBook-
Sales()
from p in publishers, b in p.booksPublished
where p.publisherName = ''Springer-Verlag''
```

This query illustrates the alternative syntax for the alias in the from clause, using the syntax "variable in collection". The alias p ranges over the publishers extent, whereas the alias b ranges over the multivalued relationship booksPublished of each publisher that satisfies the where condition. The select clause returns the name of each field and its value, where sales returns the results of a method call.

Object-Relational Data Model

An object-relational database (ORDB) refers to a relational database that has evolved by extending its data model to support user-defined types along with additional object features. An ORDB supports the traditional relational table in addition to introducing the concept of a typed table, which is similar to a class in an OODB. A typed table is created based on a user-defined type (UDT), which provides a way to define complex types with support for encapsulation. UDTs and their corresponding typed tables can be formed into class hierarchies with inheritance of state and behavior. The rows (or instances) of a typed table have object identifiers that are referred to as object references. Object references can be used to define relationships between tables that are based on object identity.

Figure 3 presents an ORDB schema of the Publisher application that is defined using the object-relational extensions to the SQL standard. The type personUdt is

```
create type personUdt as
(personName varchar(15),
address varchar(20))
instantiable not final ref is system generated;

create table person of personUdt
(primary key (personName),
ref is personID system generated);

create type authorUdt under personUdt as
(authorSince varchar(10),
wrote ref (bookUdt) scope book array [10]
    references are checked on delete no action)
instantiable not final
method calcAuthorBookSales() returns decimal;

create table author of authorUdt under person;

create type customerUdt under personUdt as
(loginName varchar(10),
buys ref (purchaseUdt) scope purchase array [50]
    references are checked on delete no action)
instantiable not final
method calcPurchases() returns decimal;

create table customer of customerUdt under person
(unique (loginName));

create type publisherUdt as
(publisherName varchar(30),
booksPublished ref (bookUdt) scope book array [1000]
    references are checked on delete set null)
instantiable not final ref is system generated
method calcTotalSales() returns decimal;

create table publisher of publisherUdt
(primary key (publisherName),
ref is publisherID system generated);

create type purchaseUdt as
(dateOfPurchase date,
purchasedBy ref (customerUdt) scope customer
    references are checked on delete cascade,
bookPurchased ref (bookUdt) scope book
    references are checked on delete cascade)
instantiable not final ref is system generated;

create table purchase of purchaseUdt
(ref is purchaseID system generated);

create type bookUdt as
(isbn varchar(30),
title varchar(50),
listPrice decimal,
authoredBy ref (authorUdt) scope author array[5]
    references are checked on delete no action,
boughtBy ref (purchaseUdt) scope purchase
array[1000]
    references are checked on delete set null,
publishedBy ref (publisherUdt) scope publisher
    references are checked on delete no action)
instantiable not final ref is system generated
 method calcBookSales() returns decimal;

create table book of bookUdt
(primary key (isbn),
ref is bookID system generated);
```

Object Data Models. Figure 3. ORDB schema of publisher application.

an example of specifying a UDT. The UDT defines the structure of the type by identifying attributes together with their type definitions. The phrase "instantiable not final ref is system generated" defines three properties of the type:

1. "Instantiable" indicates that the type supports a constructor function for the creation of instances of the type. The phrase "not instantiable" can be used in the case where the type has a subtype and instances can only be created at the subtype level.
2. "Not final" indicates that the type can be specialized into a subtype. The phrase "final" can be used to indicate that a type cannot be further specialized.
3. "Ref is system generated" indicates that the database system is responsible for automatically generating an internal object identifier. The SQL standard supports other options for the generation of object identifiers, which include user-specified object-identifiers as well as identifiers that are derived from other attributes.

Definition of the personUdt type is followed by the specification of the person typed table, which is based on the personUdt type. The person typed table automatically acquires columns for each of the attributes defined in personUdt. In addition, the person typed table has a column for an object identifier that is associated with every row in the table. The phrase "ref is personID" defines that the name of the object identifier column is personID. The definition of a typed table can add constraints to the columns that are defined in the type associated with the table. For example, personName is defined to be a primary key in the person typed table.

The authorUdt type is defined as a subtype of personUdt, as indicated by the "under personUdt" clause. In addition to defining the structure of the type, authorUdt also defines behavior with the definition of the calcAuthorBookSales method. Since authorUdt is a subtype of personUdt, authorUdt will inherit the object identifier (personID) defined in personUdt. For consistency, the author table is also defined to be a subtable of the person object table. The typed table hierarchy therefore parallels the UDT hierarchy. In a similar manner, customerUdt is defined to be a subtype of personUdt and the customer typed table, based on customerUdt, is defined to be a subtable of the person table. UDTs and typed tables are also defined for the Book and Publisher classes from Fig. 1, as well as the (reified implementation of the) Purchase association class.

Figure 3 also illustrates the use of object references to represent identity-based relationships between UDTs. Recall from the object data model in Fig. 1 that a book is published by one publisher; a publisher publishes many books. In an ORDB, this relationship is established through the use of reference types. In the bookUdt, the publishedBy attribute has the type ref (publisherUdt), indicating that the value of publishedBy is a reference to the object identifier (publisherID) of a publisher. In the inverse direction, the type of booksPublished in the publisherUdt is an array of ref(bookUdt), indicating that booksPublished is an array of object references to books. Each attribute definition includes a scope clause and a "references are checked" clause. Since a UDT can be used to define multiple tables, the scope clause defines the table of the object reference. The references clause specifies the same options for referential integrity of object references as originally defined for traditional relational tables.

To establish the fact that a book is published by a specific publisher, the object identifier of publisher is retrieved to create the relationship:

```
update book
set publishedBy = (select publisherID
                   from publisher
                   where publisherName =
                   ''Prentice Hall'')
where isbn = ''0-13-042898-1'';
```

A similar update statement can be used to establish the relationship in the inverse direction by adding the book oid to the array of object references of the publisher.

References can be traversed to query information about relationships. For example, to return the name of the publisher of a specific book, the following query can be used:

```
select publishedBy.publisherName
from book
where isbn = ''0-13-042898-1'';
```

The dot notation in the select clause performs an implicit join between the book table and the publisher table, returning the name of the publisher. The deref() function can also be used to retrieve the entire structured type associated with a reference value. For

example, the following query will return the full instance of the publisherUdt type, rather than just the publisherName:

```
select deref(publishedBy)
from book
where isbn = ''0-13-042898-1'';
```

In this case, the result of the query is a value of type publisherUdt, containing the publisher name and the array of references to books published by the publisher.

Key Applications

Computer-Aided Design, Geographic Information Systems, Computer-Aided Software Engineering, Embedded Systems, Real-time Control Systems.

Cross-references

- ▶ Conceptual Schema Design
- ▶ Database Design
- ▶ Extended Entity-Relationship Model
- ▶ OQL
- ▶ Relational Model
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading

1. Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D., and Zdonik S. The Object-Oriented Database System Manifesto. In Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, North Holland, 1990.
2. Cattell R.G.G., Barry D.K., Berler M., Eastman J., Jordan D., Russell C., Schadow O., Stanienda T., and Velez F. (eds.). The Object Data Standard: ODMG 3.0 Morgan Kaufmann, San Mateo, CA, 2000.
3. Chaudhri A. and Zicari R. (eds.). Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML. J. Wiley, New York, 2000.
4. Codd E.F. A relational model of data for large shared data banks. Comm. ACM, 13(6), 1970.
5. Dietrich S.W. and Urban S.D. An Advanced Course in Database Systems: Beyond Relational Databases. Prentice Hall, Upper Saddle River, NJ, 2005.
6. Koshafian S. and Copeland G. Object identity. ACM SIGPLAN Not., 20(11), 1986.
7. Loomis M.E.S. and Chaudhri A. (eds.). Object Databases in Practice: Prentice Hall, Upper Saddle River, NJ, 1997.
8. Melton J. Advanced SQL:1999: Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann, San Mateo, CA, 2002.
9. Object Database Management Systems: The Resource Portal for Education and Research, http://odbms.org/
10. Rowe L. and Stonebraker M. The Postgres Data Model. In Proc. 13th Int. Conf. on Very Large Data Bases. 1987.
11. Rumbaugh J., Jacobson I., and Booch G. The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, MA, 1991.
12. Stonebraker M. Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann, San Mateo, CA, 1995.
13. Stonebraker M., Rowe L., Lindsay B., Gray J., Carey M., Brodie M., Bernstein P., and Beech D. Third generation database system manifesto. ACM SIGMOD Rec., 19(3), 1990.
14. Stroustrup B. The C++ Programming Language, 3rd edn. Reading, MA. Addison-Wesley, Reading, MA, 1997.
15. Zdonik S.B. and Maier D. Readings in Object-Oriented Database Systems. Morgan Kaufmann, San Mateo, CA, 1990.

Object Detection and Recognition

▶ Automatic Image Annotation

Object Flow Diagrams

▶ Activity Diagrams

Object Identification

▶ Object Recognition

Object Identification

▶ Semantic Data Integration for Life Science Entities

Object Identifier

▶ Object Identity

Object Identity

Susan D. Urban[1], Suzanne W. Dietrich[2]
[1]Texas Tech University, Lubbock, TX, USA
[2]Arizona State University, Phoenix, AZ, USA

Synonyms

Object identifier; Oid; Object reference

Definition
Object identity is a property of data that is created in the context of an object data model, where an object is assigned a unique internal object identifier, or oid. The object identifier is used to define associations between objects and to support retrieval and comparison of object-oriented data based on the internal identifier rather than the attribute values of an object.

Key Points
In an object data model, an object is created as an instance of a class. An object has an object identifier as well as a state. An object identifier is immutable, meaning that the value of the object identifier cannot change over the lifetime of the object. The state, on the other hand, is mutable, representing the attributes that describe the object and the relationships that define associations among objects. Relationships in an object data model are defined using object references based on internal object identifiers rather than attribute values as in a relational data model. As a result, the attribute values of an object can freely change without affecting identity-based relationships.

Variables that contain object references can be compared using either object identity or object equality. Two object references are identical if they contain the same object identifiers. In contrast, two object references, that possibly contain different object identifiers, are equal if the values of attributes and relationships in each object state are identical. Shallow equality is the process of comparing the immediate values of attributes and relationships. Deep equality involves the traversal of object references in the comparison process. Query languages for objects must incorporate operators to distinguish between object identity and object equality in the specification of object queries.

Cross-references
- ▶ Conceptual Schema Design
- ▶ Extended Entity-Relationship Model
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading
1. Beeri C. and Thalheim B. Identification as a Primitive of Database Models. In Proc. 7th Int. Workshop on Foundations of Models and Languages for Data and Objects, 1999.
2. Koshafian S. and Copeland G. Object identity. ACM SIGPLAN Not., 20(11), 1986.

Object Labeling
▶ Object Recognition

Object Monitor
▶ Transactional Middleware

Object Query Language
▶ OQL

Object Recognition
Ming-Hsuan Yang
University of California at Merced, Merced, CA, USA

Synonyms
Object identification; Object labeling

Definition
Object recognition is concerned with determining the identity of an object being observed in the image from a set of known labels. Oftentimes, it is assumed that the object being observed has been detected or there is a single object in the image.

Historical Background
As the holy grail of computer vision research is to tell a story from a single image or a sequence of images, object recognition has been studied for more than four decades [9,22]. Significant efforts have been spent to develop representation schemes and algorithms aiming at recognizing generic objects in images

taken under different imaging conditions (e.g., viewpoint, illumination, and occlusion). Within a limited scope of distinct objects, such as handwritten digits, fingerprints, faces, and road signs, substantial success has been achieved. Object recognition is also related to content-based image retrieval and multimedia indexing as a number of generic objects can be recognized. In addition, significant progress towards object categorization from images has been made in the recent years [17]. Note that object recognition has also been studied extensively in psychology, computational neuroscience and cognitive science [4,9].

Foundations

Object recognition is one of the most fascinating abilities that humans easily possess since childhood. With a simple glance of an object, humans are able to tell its identity or category despite of the appearance variation due to change in pose, illumination, texture, deformation, and under occlusion. Furthermore, humans can easily generalize from observing a set of objects to recognizing objects that have never been seen before. For example, kids are able to generalize the concept of "chair" or "cup" after seeing just a few examples. Nevertheless, it is a daunting task to develop vision systems that match the cognitive capabilities of human beings, or systems that are able to tell the specific identity of an object being observed. The main reasons can be attributed to the following factors: relative pose of an object to a camera, lighting variation, and difficulty in generalizing across objects from a set of exemplar images. Central to object recognition systems are how the regularities of images, taken under different lighting and pose conditions, are extracted and recognized. In other words, all the algorithms adopt certain representations or models to capture these characteristics, thereby facilitating procedures to tell their identities. In addition, the representations can be either 2D or 3D geometric models. The recognition process, either generative or discriminative, is then carried out by matching the test image against the stored object representations or models.

Geometry-Based Approaches

Early attempts at object recognition were focused on using geometric models of objects to account for their appearance variation due to viewpoint and illumination change. The main idea is that the geometric description of a 3D object allows the projected shape to be accurately predicated in a 2D image under projective projection, thereby facilitating recognition process using edge or boundary information (which is invariant to certain illumination change). Much attention was made to extract geometric primitives (e.g., lines, circles, etc.) that are invariant to viewpoint change [13]. Nevertheless, it has been shown that such primitives can only be reliably extracted under limited conditions (controlled variation in lighting and viewpoint with certain occlusion). Mundy provides an excellent review on geometry-based object recognition research [12].

Appearance-Based Algorithms

In contrast to early efforts on geometry-based object recognition, most recent efforts have been centered on appearance-based techniques as advanced feature descriptors and pattern recognition algorithms are developed [8]. Most notably, the eigenface method has attracted much attention as it is one of the first face recognition systems that are computationally efficient and relatively accurate [21]. The underlying idea of this approach is to compute eigenvectors from a set of vectors where each one represents one face image as a raster scan vector of gray-scale pixel values. Each eigenvector, dubbed an eigenface, captures certain variance among all the vectors, and a small set of eigenvectors captures almost all the appearance variation of face images in the training set. Given a test image represented as a vector of gray-scale pixel values, its identity is determined by finding the nearest neighbor of this vector after being projected onto a subspace spanned by a set of eigenvectors. In other words, each face image can be represented by a linear combination of eigenfaces with minimum error (often in the L2 sense), and this linear combination constitutes a compact reorientation. The eigenface approach has been adopted in recognizing generic objects across different viewpoints [14] and modeling illumination variation [2].

As the goal of object recognition is to tell one object from the others, discriminative classifiers have been used to exploit the class specific information. Classifiers such as k-nearest neighbor, neural networks with radial basis function (RBF), dynamic link architecture, Fisher linear discriminant, support vector machines (SVM), sparse network of Winnows (SNoW), and boosting algorithms have been applied to recognize 3D objects from 2D images [16,6,1,18,19]. While

appearance-based methods have shown promising results in object recognition under viewpoint and illumination change, they are less effective in handling occlusion. In addition, a large set of exemplars needs to be segmented from images for generative or discriminative methods to learn the appearance characteristics. These problems are partially addressed with parts-based representation schemes.

Feature-Based Algorithms

The central idea of feature-based object recognition algorithms lies in finding interest points, often occurred at intensity discontinuity, that are invariant to change due to scale, illumination and affine transformation (a brief review on interest point operators can be found in [8]). The scale-invariant feature transform (SIFT) descriptor is arguably one of the most widely used feature representation schemes for vision applications [8]. The SIFT approach uses extrema in scale space for automatic scale selection with a pyramid of difference of Gaussian filters, and keypoints with low contrast or poorly localized on an edge are removed. Next, a consistent orientation is assigned to each keypoint and its magnitude is computed based on the local image gradient histogram, thereby achieving invariance to image rotation. At each keypoint descriptor, the contribution of local image gradients are sampled and weighted by a Gaussian, and then represented by orientation histograms. For example, the 16×16 sample image region and 4×4 array of histograms with 8 orientation bins are often used, thereby providing a 128-dimensional feature vector for each keypoint. Objects can be indexed and recognized using the histograms of keypoints in images. Numerous applications have been developed using the SIFT descriptors, including object retrieval [15,20], and object category discovery [5].

Although the SIFT approach is able to extract features that are insensitive to certain scale and illumination changes vision applications with large base line changes entail the need of affine invariant point and region operators [11]. A performance evaluation among various local descriptors can be found in [10], and a study on affine region detectors is presented in [11]. Finally, SIFT-based methods are expected to perform better for objects with rich texture information as sufficient number of keypoints can be extracted. On the other hand, they also require sophisticated indexing and matching algorithms for effective object recognition [8,17].

Key Applications

Biometric recognition, and optical character/digit/document recognition are arguably the most widely used applications. In particular, face recognition has been studied extensively for decades and with large scale ongoing efforts [23]. On the other hand, biometric recognition systems based on iris or fingerprint as well as as handwritten digit have become reliable technologies [3,7]. Other object recognition applications include surveillance, industrial inspection, content-based image retrieval (CBIR), robotics, medical imaging, human computer interaction, and intelligent vehicle systems, to name a few.

Future Directions

With more reliable representation schemes and recognition algorithms being developed, tremendous progress has been made in the last decade towards recognizing objects under variation in viewpoint, illumination and under partial occlusion. Nevertheless, most working object recognition systems are still sensitive to large variation in illumination and heavy occlusion. In addition, most existing methods are developed to deal with rigid objects with limited intra-class variation. Future research will continue searching for robust representation schemes and recognition algorithms for recognizing generic objects.

Data Sets

Numerous face image sets are available on the web

- FERET face data set: http://www.itl.nist.gov/iad/humanid/feret/
- UMIST data set: http://images.ee.umist.ac.uk/danny/database.html
- Yale data set: http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html
- AR data set: http://cobweb.ecn.purdue.edu/%7Ealeix/aleix_face_DB.html
- CMU PIE data set: http://www.ri.cmu.edu/projects/project_418.html

There are several large data sets for object recognition experiments,

- COIL data set: http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php
- CalTech data sets: http://www.vision.caltech.edu/html-files/archive.html
- PASCAL visual object classes: http://www.pascal-network.org/challenges/VOC/

URL to Code

There are a few excellent short courses on object recognition in recent conferences available on the web.

- "Recognition and matching based on local invariant features" by Schmid and Lowe in IEEE Conference on Computer Vision and Pattern Recognition 2003: http://lear.inrialpes.fr/people/schmid/cvpr-tutorial03/
- "Learning and recognizing object categories" by Fei-Fei, Fergus and Torralba in IEEE International Conference on Computer Vision 2005:http://people.csail.mit.edu/torralba/shortCourseRLOC/
- "Recognizing and Learning Object Categories: Year 2007" by Fei-Fei, Fergus and Torralba in IEEE Conference on Computer Vision and Pattern Recognition 2005: http://people.csail.mit.edu/torralba/shortCourseRLOC/

Sample code for face recognition and SIFT descriptors:

- Face recognition: http://www.face-rec.org/
- Lowe's sample SIFT code: http://www.cs.ubc.ca/~lowe/keypoints/
- MATLAB implementation of SIFT descriptors by Vedaldi: http://vision.ucla.edu/~vedaldi/code/sift/sift.html
- `l`ibsift by Nowozin: http://user.cs.tu-berlin.de/~nowozin/libsift/

Grand challenge in object recognition:

- NIST face recognition grand challenge: http://www.frvt.org/FRGC/
- NIST multiple biometric grand challenge: http://face.nist.gov/mbgc/
- PASCAL visual object classes challenge 2007: http://www.pascal-network.org/challenges/VOC/voc2007/index.html

Cross-references

► Object Detection and Recognition

Recommended Reading

1. Belhumeur P., Hespanha J., and Kriegman D. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. IEEE Trans. Pattern Analy. Machine Intell., 19(7):711–720, 1997.
2. Belhumeur P. and Kriegman D. What is the Set of Images of an Object under All Possible Illumination Conditions. Int. J. Comput. Vision, 28(3):1–16, 1998.
3. Daugman J. Probing the uniqueness and randomness of iriscodes: Results from 200 billion iris pair comparisons. In Proc. IEEE, 94(11):1927–1935, 2006.
4. Edelman S. Representation and recognition in vision. MIT, Cambridge, MA, 1999.
5. Fergus R., Perona P., and Zisserman A. Object class recognition by unsupervised scale-invariant learning. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition. 2003, pp. 264–271.
6. Lades M., Vorbrüggen J.C., Buhmann J., Lange J., von der Malsburg C., Würtz R.P., and Konen W. Distortion Invariant Object Recognition in the Dynamic Link Architecture. IEEE Trans. Comput., 42:300–311, 1993.
7. Lecun Y., Bottou L., Bengio Y., and Haffner P. Gradient-based learning applied to document recognition. In Proc. IEEE, 86(11):2278–2324, 1998.
8. Lowe D. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91–110, 2004.
9. Marr D. Vision. W.H. Freeman and Company, San Francisco, CA, USA, 1982.
10. Mikolajczyk K. and Schmid C. A performance evaluation of local descriptors. IEEE Trans. Pattern Analy. Machine Intell., 27 (10):1615–1630, 2005.
11. Mikolajczyk K., Tuytelaars T., Schmid C., Zisserman A., Matas J., Schaffalitzky F., Kadir T., and Van Gool L. A comparison of affine region detectors. Int. J. Comput. Vision, 65(1/2):43–72, 2006.
12. Mundy J. Object recognition in the geometric era: a retrospective. In Toward category-level object recognition. J. Ponce, M. Hebert, C. Schmid, and A. Zisserman (eds.). Springer, Berlin, 2006, pp. 3–29.
13. Mundy J. and Zisserman A. Geometric invariance in computer vision. MIT, Cambridge, MA, 1992.
14. Murase H. and Nayar S.K. Visual learning and recognition of 3-D objects from appearance. Int. J. Comput. Vision, 14:5–24, 1995.
15. Nister D. and Stewenius H. Scalable recognition with a vocabulary tree. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition. 2006, pp. 2161–2168.
16. Poggio T. and Edelman S. A Network that Learns to Recognize 3D Objects. Nature, 343:263–266, 1990.
17. Ponce J., Hebert M., Schmid C., and Zisserman A. Toward category-level object recognition. Springer, Berlin, 2006.
18. Pontil M. and Verri A. Support Vector Machines for 3D Object Recognition. IEEE Trans. Pattern Analy. Machine Intell., 20(6):637–646, 1998.
19. Roth D., Yang M.-H., and Ahuja N. Learning to Recognize Objects. Neural Comput., 14(5):1071–1104, 2002.
20. Sivic J. and Zisserman A. Video Google: a text retrieval approach to object matching in videos. In Proc. 9th IEEE Conf. Computer Vision. 2003, pp. 1470–1477.
21. Turk M. and Pentland A. Eigenfaces for recognition. J. Cognitive Neurosci., 3(1):71–86, 1991.
22. Ullman S. High-level vision: Object recognition and visual recognition. MIT, Cambridge, MA, 1996.
23. Zhao W., Chellappa R., Rosenfeld A., and Phillips J.P. Face recognition: A literature survey. ACM Comput. Surv., 35(4):399–458, 2003.

Object Reference

► Object Identity

Object Relationship Attribute Data Model for Semi-structured Data

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, Auckland, New Zealand
[2]National University of Singapore, Singapore

Synonyms

ORA-SS data model; ORA-SS schema diagram

Definition

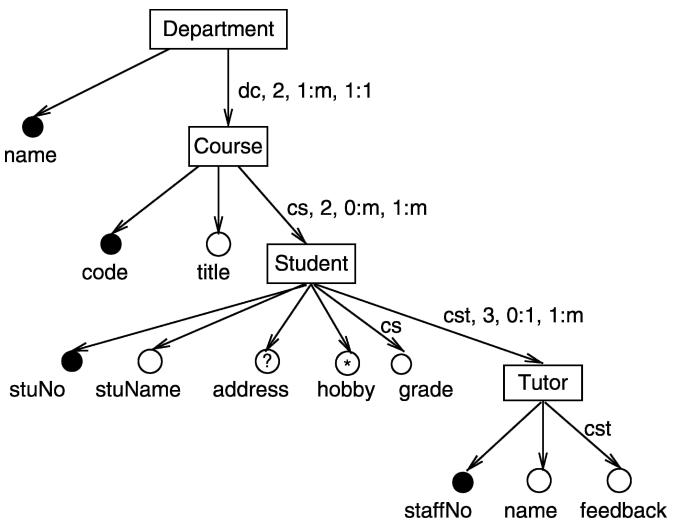
When a database schema is designed, a data model is initially used to model the real world constraints that are taken into account in the design of the schema. For semi-structured database design, it is necessary to capture the following constraints: object classes, n-ary relationship types, attributes of object classes, attributes of relationship types, cardinality, participation and uniqueness constraints, ordering, irregular and heterogeneous structures, for both data- and document-centric data.

Key Points

The ORA-SS (Object-Relationship-Attribute Data Model for Semi-structured Data) data model was designed [1] specifically to capture the constraints that are necessary for designing semi-structured databases, for normalization of schemas, and for defining views.

Figure 1 models the scenario where there is a department, with a name and many courses. A course has a unique code, a title, and many students, and a student has a unique student number, name, address, and many hobbies. For each course that a student takes, they have a grade. There is a tutor for each student in each course they take. A tutor has a unique staff number and a name, and each student can give feedback for the tutor they have in each course that they take.

A closer look is now taken at the notation used. Each of the rectangles represents an object class, the circles represent attributes and the labeled directed edges between object classes represent relationship types. A filled circle is an identifier, which is similar to a key in relational databases and an identifier of an object class. The "?" in the circle represents zero or one occurences of that attribute, while a "*" represents zero or more occurences. The default is one. An attribute in an ORA-SS diagram could be represented as an attribute or an element in an XML document. The label on the edge has *name, n, a:b, c:d*, where *name* is the name of the relationship type, *n* is the degree, *a:b* is the participation constraint on the parent and *c:d* is the participation constraint on the child. The participation constraint *a:b* indicates that the parent object participates in a minimum of *a* and a maximum of *b* relationships. Whereas, the participation constraint *c:d* indicates that the child object participates in a minimum of *c* and a maximum of *d* relationships. A label on the edge between an object class and an attribute, *name*, indicates that the attribute belongs to relationship type *name*.



Object Relationship Attribute Data Model for Semi-structured Data. Figure 1. An ORA-SS schema diagram.

Consider the example in Fig. 1. There are object classes *Department*, *Course*, *Student* and *Tutor*. Object class *Department* has an identifier *name*, and is the parent object class in the relationship type between *Department* and *Course*. The relationship type is a binary relationship with name *dc*, and the participation constraint *1:m* indicates that a department has a minimum of *one* course and a maximum of *m* courses, where *m* means many, i.e., any number of courses. The participation constraint *1:1* indicates that each course must belong to *one* department and can belong to a maximum of *one* department. Each course has an identifier *code*, a required attribute *title*, and is the parent object class in the relationship type, *cs*, between *course* and *student*. Object class *Student* has an identifier *stuNo*, a required attribute *stuName*, an optional attribute *address*, and zero or more *hobby*. There is a binary relationship type, *cs*, between *Course* and *Student*, where a *Course* can have zero or more *Student*s, and a *Student* takes one or more *Courses*. The attribute *grade* belongs to the relationship type, *cs*, that is it represents the grade a student scored in a particular course. There is a ternary relationship type among object classes *Course*, *Student* and *Tutor*. Each course-student pair can have zero to one tutor, and each tutor belongs to one or more course-student pairs. Each tutor has an identifier *staffNo*, a required attribute *name*, and there is an attribute *feedback* for each tutor from a particular student in a particular course.

Cross-references

- ▶ Entity Relationship Model
- ▶ Hierarchical Data Model
- ▶ Object Data Models
- ▶ Semi-structured Data Model
- ▶ Semi-structured Database Design
- ▶ XML Integrity Constraints
- ▶ XML Schema

Recommended Reading

1. Ling T.W., Lee M.L., and Dobbie G. Semi-structured Database Design. Springer, Berlin Heidelberg New York, 2005.

Object Request Broker

- ▶ CORBA
- ▶ Request Broker

Object-based Storage Device

- ▶ Network Attached Secure Device

Object-Role Modeling

Terry Halpin
Neumont University, South Jordan, UT, USA

Synonyms

Fact-oriented modeling; NIAM

Definition

Object-Role Modeling (ORM), also known as *fact-oriented modeling*, is a conceptual approach to modeling and querying the information semantics of business domains in terms of the underlying facts of interest, where all facts and rules may be verbalized in language readily understood by non-technical users of those business domains. Unlike Entity-Relationship (ER) modeling and Unified Modeling Language (UML) class diagrams, ORM treats all facts as relationships (unary, binary, ternary etc.). How facts are grouped into structures (e.g., attribute-based entity types, classes, relation schemes, XML schemas) is considered a design level, implementation issue that is irrelevant to the capturing of essential business semantics.

Avoiding attributes in the base model enhances semantic stability, populatability, and natural verbalization, facilitating communication with all stakeholders. For information modeling, fact-oriented graphical notations are typically far more expressive than those provided by other notations. Fact-oriented textual languages are based on formal subsets of native languages, so are easier to understand by business people than technical languages like UML's Object Constraint Language (OCL). Fact-oriented modeling includes procedures for mapping to attribute-based structures, so may also be used to front-end other approaches.

The fact-oriented modeling approach comprises a family of closely related "dialects", known variously as Object-Role Modeling (ORM), Natural Language Information Analysis Method (NIAM), and Fully-Communication Oriented Information Modeling (FCO-IM). While not adopting the ORM graphical notation, the Object-oriented Systems Model (OSM) [4] and the Semantics of Business Vocabulary and

business Rules (SBVR) [13] initiative within the Object Management Group (OMG) are close relatives, with their attribute-free philosophy.

Historical Background

In 1973, Falkenberg generalized work by Abrial and Senko on binary relationships to *n*-ary relationships, and excluded attributes at the conceptual level to avoid "fuzzy" distinctions and to simplify schema evolution. Later, Falkenberg proposed the fundamental ORM framework, which he called the "object-role model" [5]. This framework allowed *n*-ary and nested relationships, but depicted roles with arrowed lines. Nijssen adapted this framework by introducing a circle-box notation for objects and roles, and adding a linguistic orientation and design procedure to provide a modeling method called ENALIM (Evolving NAtural Language Information Model) [12]. Nijssen's team of researchers at Control Data in Belgium developed the method further, including van Assche who classified object types into lexical object types (LOTs) and non-lexical object types (NOLOTs). Today, LOTs are commonly called "entity types" and NOLOTs are called "value types". Meersman added subtyping to the approach, and made major contributions to the RIDL query language [11] with Falkenberg and Nijssen. The method was renamed "aN Information Analysis Method" (NIAM). Later, the acronym "NIAM" was given different expansions, and is now known as "Natural language Information Analysis Method".

In the 1980s, Nijssen and Falkenberg worked on the design procedure and moved to the University of Queensland, where the method was further enhanced by Halpin, who provided the first full formalization, including schema equivalence proofs, and made several refinements and extensions. In 1989, Halpin and Nijssen co-authored a book on the approach, followed a year later by Wintraecken's book [16]. Today several books, including major works by Halpin [10], and Bakema et al. [1] expound on the approach.

Many researchers contributed to the fact-oriented approach over the years, and there is no space here to list them all. Today various versions exist, but all adhere to the fundamental object-role framework. Habrias developed an object-oriented version called MOON (Normalized Object-Oriented Method). The Predicator Set Model (PSM), developed mainly by ter Hofstede et al. [7], includes complex object constructors. De Troyer and Meersman developed a version with constructors called Natural Object-Relationship Model (NORM). Halpin developed an extended version simply called ORM, and with Bloesch and others developed an associated query language called ConQuer [2]. Bakema et al. [1] recast all entity types as nested relationships, to produce Fully Communication Oriented NIAM, which they later modified to Fully Communication Oriented Information Modeling (FCO-IM).

More recently, Meersman and others adapted ORM for ontology modeling, using a framework called DOGMA (Developing Ontology-Grounded Methodology and Applications) (http://www.starlab.vub.ac.be/website/). Nijssen and others extended NIAM to a version called NIAM2007. Halpin and others developed a second generation ORM (ORM 2), whose graphical notation is used in this article.

Foundations

ORM includes graphical and textual *languages* for modeling and querying information at the conceptual level, as well as *procedures* for designing conceptual models, transforming between different conceptual representations, forward engineering ORM schemas to implementation schemas (e.g., relational database schemas, object-oriented schemas, XML schemas, and external schemas) and reverse engineering implementation schemas to ORM schemas.

Attributes are not used as a base construct. Instead, all fact structures are expressed as *fact types* (relationship types). These may be unary (e.g., Person smokes), binary (e.g., Person was born on Date), ternary (e.g., Person visited Country in Year), and so on. This attribute-free nature has several advantages: *semantic stability* (minimize the impact of change caused by the need to record something about an attribute); *natural verbalization* (all facts and rules may be easily verbalized in sentences understandable to the domain expert); *populatability* (sample fact populations may be conveniently provided in fact tables); *null avoidance* (no nulls occur in populations of base fact types, which must be elementary or existential). Although attribute-free diagrams typically consume more space, this apparent disadvantage is easily overcome by using an ORM tool to automatically create attribute-based structures (e.g., ER, UML class, or relational schemas) as views of an ORM schema.

ORM's graphical language is far more expressive for data modeling purposes than that of UML or industrial versions of ER, as illustrated later. The *rich graphical notation* makes it easier to detect and express

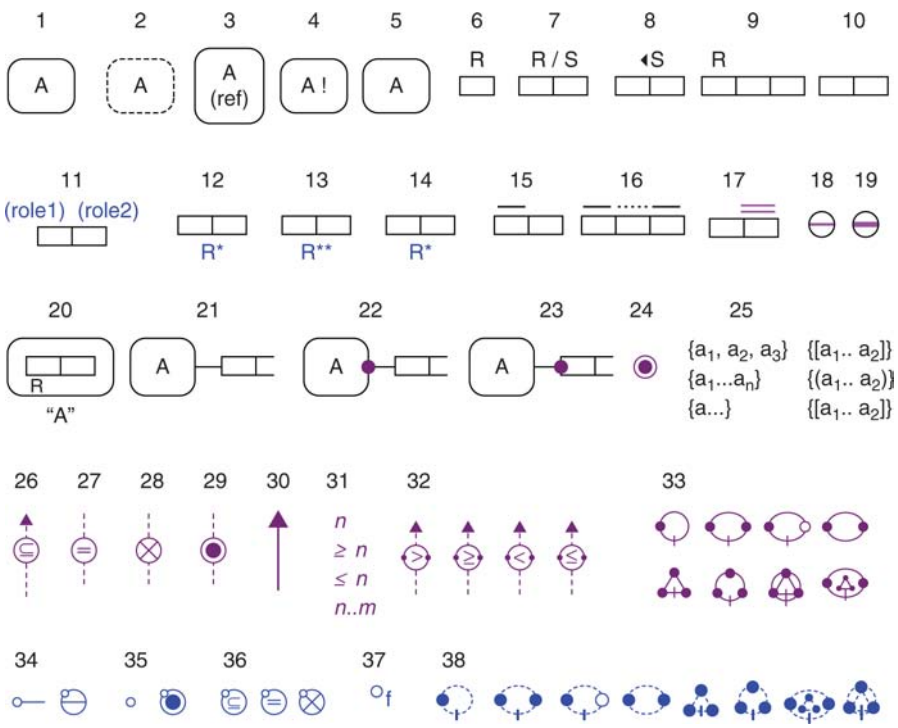
constraints, and to visually transform schemas into equivalent alternatives.

ORM includes *effective modeling procedures* for constructing and validating models. In step 1a of the Conceptual Schema Design Procedure (CSDP), the domain expert informally verbalizes facts of interest. In step 1b, the modeler formally rephrases the facts in natural yet unambiguous language, using standard reference patterns to ensure that entities are well identified. Verbalized fact instances are abstracted to fact types, which are then populated with sample instances. The constraints on the fact types are verbalized formally, a process that may be automated [8], and these verbalizations are checked with the domain expert, using positive populations to illustrate satisfaction of the constraints as well as counterexamples to illustrate what it means to violate a constraint. This approach to model *validation by verbalization and population* has proved extremely effective in industrial practice, with correct models typically obtained from the outset rather than going through unreliable iterative procedures.

Figure 1 lists the main graphical symbols in the ORM 2 notation [8], numbered for easy reference. An *entity type* (e.g., Person) is depicted as a named, soft rectangle (symbol 1), or alternatively an ellipse or hard rectangle. *Value type* (e.g., Person Name) shapes have dashed lines (symbol 2). Each entity type has a *reference scheme*, indicating how each instance may be mapped via predicates to a combination of one or more values. Injective (1:1 into) reference schemes mapping entities (e.g., countries) to single values (e.g., country codes) may be abbreviated as in symbol 3 by displaying the *reference mode* in parentheses, e.g., Country (.code). The reference mode indicates how values relate to the entities. Values are constants with a known denotation, so require no reference scheme.

Relationships used for *preferred reference* are called *existential facts* (e.g., there exists a country that has the country code 'US'). The other relationships are *elementary facts* (e.g., The country with country code 'US' has a population of 301,000,000). The exclamation mark in symbol 4 declares that an object type is *independent* (instances may exist without participating in any elementary facts). Object types displayed in multiple places are shadowed (symbol 5).

A fact type results from applying a logical *predicate* to a sequence of one or more object types. Each predicate comprises a named sequence of one or more *roles* (parts played in the relationship). A predicate is



Object-Role Modeling. Figure 1. Main ORM graphic symbols.

sentence with object holes, one for each role, with each role depicted as a box and played by exactly one object type. Symbol 6 shows a unary predicate (e.g., ... smokes), symbols 7 and 8 depict binary predicates (e.g., ... loves ...), and symbol 9 shows a ternary predicate. Predicates of higher *arity* (number of roles) are allowed. Each predicate has at least one *predicate reading*. ORM uses *mixfix* predicates, so objects may be placed at any position in the predicate (e.g., the fact type Person introduced Person to Person involves the predicate "... introduced ... to ..."). Mixfix predicates allow natural verbalization of *n*-ary relationships, as well as binary relationships where the verb is not in the infix position (e.g., in Japanese, verbs come at the end). By default, forward readings traverse the predicate from left to right (if displayed horizontally) or top to bottom (if displayed vertically). Other reading directions may be indicated by an arrow-tip (symbol 8). For binary predicates, forward and inverse readings may be separated by a slash (symbol 7). Duplicate predicate shapes are shadowed (symbol 10).

Roles may be given *role names*, displayed in square brackets (symbol 11). An asterisk indicates that the fact type is *derived* from one or more other fact types (symbol 12). If the fact type is derived and stored, a double asterisk is used (symbol 13). Fact types that are semi-derived are marked "+" (symbol 14). *Internal uniqueness constraints*, depicted as bars over one or more roles in a predicate, declare that instances for that role (combination) in the fact type population must be unique (e.g., symbols 15, 16). For example, a uniqueness constraint on the first role of Person was born in Country verbalizes as: **Each** person was born in **at most one** Country. If the constrained roles are not contiguous, a dotted line separates the constrained roles (symbol 16). A predicate may have many uniqueness constraints, at most one of which may be declared *preferred* by a double-bar (symbol 17). An *external uniqueness constraint* shown as a circled uniqueness bar (symbol 18) may be applied to two or more roles from different predicates by connecting to them with dotted lines. This indicates that instances of the role combination in the join of those predicates are unique. For example, if a state is identified by combining its state code and country, an external uniqueness constraint is added to the roles played by Statecode and Country in: State has Statecode; State is in Country. Preferred external uniqueness constraints are depicted by a circled double-bar (symbol 19).

To talk about a relationship, one may *objectify* it (i.e., make an object out of it) so that it can play roles. Graphically, the objectified predicate (a.k.a. *nested* predicate) is enclosed in a soft rectangle, with its name in quotes (symbol 20). Roles are connected to their players by a line segment (symbol 21). A *mandatory role constraint* declares that every instance in the population of the role's object type must play that role. This is shown as a large dot placed at the object type end (symbol 22) or the role end (symbol 23). An *inclusive-or* (*disjunctive mandatory)* constraint applied to two or more roles indicates that all instances of the object type population must play at least one of those roles. This is shown by connecting the roles by dotted lines to a circled dot (symbol 24).

To restrict the population of an object type or role, the relevant values may be listed in braces (symbol 25). An ordered range may be declared separating end values by "..". For continuous ranges, a square/ round bracket indicates an end value is included/excluded. For example, "(0..10)" denotes the positive real numbers up to 10. These constraints are called *value constraints*.

Symbols 26–28 denote *set comparison constraints*, which apply only between compatible role sequences. A dotted arrow with a circled subset symbol depicts a *subset constraint*, restricting the population of the first sequence to be a subset of the second (symbol 26). A dotted line with a circled "=" symbol depicts an *equality constraint*, indicating the populations must be equal (symbol 27). A circled "X" (symbol 28) depicts an *exclusion constraint*, indicating the populations are mutually exclusive. Exclusion and equality constraints may be applied between two or more sequences. Combining an inclusive-or and exclusion constraint yields an *exclusive-or constraint* (symbol 29).

A solid arrow (symbol 30) from one object type to another indicates that the first is a (proper) *subtype* of the other (e.g., Woman is a subtype of Person). Mandatory (circled dot) and exclusion (circled "X") constraints may be displayed between subtypes, but are implied by other constraints if the subtypes have formal definitions. Symbol 31 shows four kinds of *frequency constraint*. Applied to a role sequence, these indicate that instances that play those roles must do so *exactly n* times, *at least n* times, *at most n* times, or *at least n and at most m* times. Symbol 32 shows four varieties of *value-comparison constraint*. The arrow shows the direction in which to apply the circled

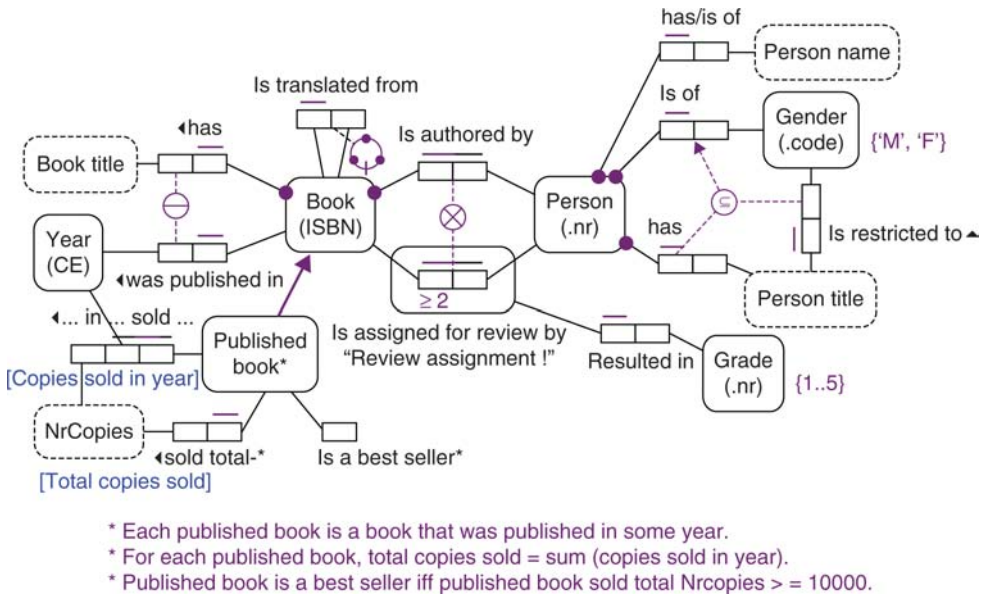
operator between two instances of the same type (e.g., **For each** Employee, hiredate > birthdate).

Symbol 33 shows the main kinds of *ring constraint* that may apply to a pair of compatible roles. Read left to right and top row first, these indicate that the binary relation formed by the role population must respectively be irreflexive, asymmetric, antisymmetric, reflexive, intransitive, acyclic, intransitive and acyclic, or intransitive and asymmetric.

The previous constraints are *alethic* (necessary, so can't be violated) and are colored violet. ORM 2 also supports *deontic* rules (obligatory, but can be violated). These are colored blue, and either add an "o" for obligatory, or soften lines to dashed lines. Displayed here are the deontic symbols for uniqueness (symbol 34), mandatory (symbol 35), set-comparison (symbol 36), frequency (symbol 37) and ring (symbol 38) constraints.

Figure 2 shows a sample ORM schema for a book publishing domain. A detailed discussion using the CSDP to develop this schema may be found elsewhere [9]. Each book is identified by an International Standard Book Number (ISBN), each person is identified by a person number, each grade is identified by a grade number in the range 1 through 5, each gender is identified by a code ('M' for male and 'F' for Female), and each year is identified by its common era (CE) number. Published Book is a derived subtype determined by the subtype definition shown at the bottom of the figure. Review Assignment objectifies the relationship Book is assigned for review by Person, and is independent since an instance of it may exist without playing any other role (one can known about a review assignment before knowing what grade will result from that assignment).

The internal uniqueness constraints (depicted as bars) and mandatory role constraints (solid dots) verbalize as follows: **Each** Book is translated from **at most one** Book; **Each** Book has **exactly one** Book Title; **Each** Book was published in **at most one** Year; **For each** Published Book **and** Year, **that** Published Book in **that** Year sold **at most one** NrCopies; **Each** Published Book sold **at most one** total NrCopies; **It is possible that the same** Book is authored by **more than one** Person **and that more than one** Book is authored by **the same** Person; **Each** Book is authored by **some** Person; **It is possible that the same** Book is assigned for review by **more than one** Person **and that more than one** Book is assigned for review by **the same** Person; **Each** Review Assignment resulted in **at most one** Grade; **Each** Person has **exactly one** Person Name; **Each** Person has **at most one** Gender; **Each** Person has **at most one** Person Title; **Each** Person Title is restricted to **at most one** Gender.



Object-Role Modeling. Figure 2. An ORM schema for a book publishing domain.

The external uniqueness constraint (circled bar) indicates that the combination of BookTitle and Year applies to at most one Book. The acyclic ring constraint (circle with three dots and a bar) on the book translation predicate indicates that no book can be a translation of itself or any of its ancestor translation sources. The exclusion constraint (circled cross) indicates that no book can be assigned for review by one of its authors. The frequency constraint (≥ 2) indicates that each book that is assigned for review is assigned for review by at least two persons. The subset constraint (circled subset symbol) means that if a person has a title that is restricted to some gender, then the person must be of that gender. The first argument of this subset constraint is a person-gender role pair projected from a join path that performs a conceptual join on PersonTitle. The last two lines at the bottom of the schema declare two derivation rules, one specified in attribute-style using role names and the other in relational style using predicate readings.

Key Applications

ORM has been used productively in industry for over 30 years, in all kinds of business domains. *Commercial tools* supporting the fact-oriented approach include Microsoft's Visio for Enterprise Architects, and the FCO-IM tool CaseTalk (www.casetalk.com). CogNIAM, a tool supporting NIAM2007 is under development at PNA Active Media (http://cogniam.com/). *Free ORM tools* include VisioModeler and Infagon (www.mattic.com). Dogma Modeler (www.starlab.vub.ac.be) and T-Lex [15] are academic ORM-based tools for specifying ontologies. NORMA (http://sourceforge.net/projects/orm), an open-source plug-in to Microsoft® Visual Studio, is under development to provide deep support for ORM 2 [3].

Future Directions

Research in many countries is actively extending ORM in many areas (e.g., dynamic rules, ontology extensions, language extensions, process modeling). A detailed overview of this research may be found in [9]. General information about ORM, and links to other relevant sites, may be found at www.ORMFoundation.org and www.orm.net.

Cross-references

- ▶ Conceptual Schema Design
- ▶ Data Model
- ▶ Entity Relationship Model
- ▶ UML

Recommended Reading

1. Bakema G., Zwart J., and van der Lek H. Fully Communication Oriented Information Modelling. Ten Hagen Stam, The Netherlands, 2000.
2. Bloesch A. and Halpin T. Conceptual queries using ConQuer-II. In Proc. 16th Int. Conf. on Conceptual Modeling, 1997, pp. 113–126.
3. Curland M. and Halpin T. Model Driven Development with NORMA. In Proc. 40th Annual Hawaii Int. Conf. on System Sciences, 2007.
4. Embley D., Kurtz B., and Woodfield S. Object-Oriented Systems Analysis: A Model-Driven Approach. Prentice Hall, Englewood Cliffs, 1992.
5. Falkenberg E. Concepts for modeling information. In Proc. IFIP Working Conference on Modelling in Data Base Management Systems, 1976, pp. 95–109.
6. Halpin T. Comparing metamodels for ER, ORM and UML data models. In Advanced Topics in Database Research, vol. 3, K. Siau (ed.). Idea Publishing Group, Hershey, 2004, pp. 23–44.
7. Halpin T. Fact-oriented modeling: past, present and future. In Conceptual Modelling in Information Systems Engineering, J. Krogstie A. Opdahl S. Brinkkemper (eds.). Springer, Berlin Heidelberg New York, 2007, pp. 19–38.
8. Halpin T. and Curland M. Automated verbalization for ORM 2. In On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, LNCS vol. 4278, 2006, pp. 1181–1190.
9. Halpin T., Evans K., Hallock P., and MacLean W. Database Modeling with Microsoft® Visio for Enterprise Architects. Morgan Kaufmann, San Francisco, CA, 2003.
10. Halpin T. and Morgan T. Information Modeling and Relational Databases, 2nd Edition. Morgan Kaufmann, San Francisco, CA, 2008.
11. Meersman R. (1982) The RIDL conceptual language, Research report. International Centre for Information Analysis Services, Control Data Belgium, Brussels, 1982.
12. Nijssen G.M. Current issues in conceptual schema concepts. In Proc. IFIP Working Conference on Modelling in Data Base Management Systems, 1977, pp. 31–66.
13. OMG 2007, Semantics of Business Vocabulary and Business Rules (SBVR). URL: http://www.omg.org/cgi-bin/doc?dtc/2006-08-05.
14. ter Hofstede A.H.M., Proper H.A., and Weide th.P. van der. Formal definition of a conceptual language for the description and manipulation of information models. Inf. Syst., 18(7):489–523, 1993.
15. Trog D., Vereecken J., Christiaens S., De Leenheer P., and Meersman R. T-Lex: a role-based ontology engineering tool. In On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, LNCS vol. 4278, 2006, pp. 1191–1200.
16. Wintraecken J. (1990) The NIAM Information Analysis Method: Theory and Practice, Kluwer, Deventer, The Netherlands, 1990.

OCL

▶ Object Constraint Language

ODB (Object Database)

▶ Object Data Models

ODBC

▶ Open Database Connectivity

Office Automation

▶ Enterprise Content Management

Oid

▶ Object Identity

OKAPI Retrieval Function

▶ BM25

OLAP

▶ On-Line Analytical Processing

On-Disk Security

▶ Storage Security

One-Copy-Serializability

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Transactional consistency in a replicated database

Definition

While transactions typically specify their read and write operations on logical data items, a replicated database has to execute them over the physical data copies. When transactions run concurrently in the system, their executions may interfere. The replicated database system has to isolate these transactions. The strongest, and most well-known correctness criterion for replicated databases is one-copy-serializability. A concurrent execution of transactions in a replicated database is one-copy-serializable if it is equivalent to a serial execution of these transactions over a single logical copy of the database.

Key Points

A transaction is a sequence of read and write operations on the data items of the database. A read operation of transaction T_i on data item x is denoted as $r_i(x)$, a write operation on x as $w_i(x)$. A transaction T_i either ends with a commit c_i (all operations succeed) or with an abort a_i (whereby all effects on the data are undone before the termination).

A replicated database consists of a set of database servers A, B, ... and each logical data item x of the database has a set of physical copies x^A, x^B, ... where the index refers to the database server on which the copy resides. *Replica Control* translates each operation $o_i(x), o_i \in \{r,w\}$ of a transaction T_i on data item x into operations $o_i(x^A), o_i(x^B)$ on physical data copies. Given a set of transactions $\mathcal{T}$, a replicated history RH describes the execution of these transactions in the replicated database. For simplicity the following discussion only considers histories where all transactions commit. A database server A executes the subset of operations of the transactions in $\mathcal{T}$ performed on copies residing on A. The local history RH^A describes the order in which these operations occur. For simplicity a local history is assumed to be a total order. RH is the union of all local histories with some additional ordering. In particular, if a transaction T_i executes $o_i(x)$ before $o_i(y)$, and RH^A contains $o_i(x^A)$ and RH^B contains $o_i(y^B)$, then $o_i(x^A) <_{RH} o_i(y^B)$.

As an example, given $T_1 = w_1(y)w_1(x)$ and $T_2 = r_2(y)w_2(x)$, and database servers A and B, both having a copy of both x and y, the local histories could be:

$$RH^A : w_1(y^A)r_2(y^A)w_1(x^A)w_2(x^A)c_1c_2$$

$$RH^B : w_1(y^B)w_1(x^B)w_2(x^B)c_2c_1$$

The replicated history *RH* is the union of these two local histories plus the ordering of $r_2(y^A) <_{RH} w_2(x^B)$.

Using this notation, the following defines one-copy-serializability for the case that replica control uses ROWA (read-one-write-all-approach), i.e., where each read operation is performed on one copy while write operations are performed on all copies of the data item. Failures are ignored. In this restricted case *conflict-equivalence* can be exploited. Two operations o_i and o_j conflict, if they are from two different transactions, access the same data copy, and at least one is a write operation.

Definition *A replicated history RH over a set of transactions $\mathcal{T}$ in a replicated system with servers A, B,...is one-copy-serializable if it is conflict-equivalent to a serial history H over $\mathcal{T}$ in a single-copy system with one logical server L. This means that if $o_i(x^A)$, $o_j(x^A) \in RH$ and the operations conflict, then $o_i(x^L) <_H o_j(x^L) \in H$ if and only if $o_i(x^A) <_{RH^A} o_j(x^A) \in RH$.*

Using conflict-equivalence, one can easily determine whether *RH* is one-copy-serializable. For each local history RH^A the serialization graph $SG(RH^A)$ has each committed transaction as node, and contains an edge from T_i to T_j if $o_i(x^A) <_{RH^A} o_j(x^A)$ and the two operations conflict. The serialization graph $SG(RH)$ is then the union of the local serialization graphs.

Theorem *A replicated history RH over a set of transactions $\mathcal{T}$ and database servers A, B,...following the ROWA strategy is one-copy-serializable if and only if its serialization graph SG(RH) is acyclic.*

The example history above is one-copy-serializable because its serialization graph contains only an edge from T_1 to T_2, i.e., in all local histories, and for any conflict between T_1 and T_2, T_1's operations are ordered before T_2's operation.

As soon as node failures are considered or both read and write operations only access a subset of copies, conflict-equivalence is not appropriate anymore because it might miss catching conflicts at the logical level. For that purpose, one can define one-copy-serializability based on view-equivalence which observes which data versions a read operation accesses and in which order write operations occur.

Cross-references

- ► Replica Control
- ► Replicated Database Concurrency Control
- ► Strong Consistency Models for Replicated Data
- ► Traditional Concurrency Control for Replicated Databases

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, USA, 1987.

One-Pass Algorithm

Nicole Schweikardt
Johann Wolfgang Goethe-University, Frankfurt am Main, Frankfurt, Germany

Synonyms

One-pass algorithm; Streaming algorithm; Data stream algorithm

Definition

A one-pass algorithm receives as input a list of data items x_1, x_2, x_3,.... It can read these data items only once, from left to right, i.e., in increasing order of the indices $i = 1, 2, 3,$.... Critical parameters of a one-pass algorithm are (1) the size of the memory used by the algorithm, and (2) the processing time per data item x_i. Typically, a one-pass algorithm is designed for answering one particular query against the input data. To this end, the algorithm stores and maintains a suitable data structure which, for each i, is updated when reading data item x_i.

The two parameters *processing time per data item* and *memory size* are usually measured as functions depending on the size N of the input (different measures of the *input size* are considered in the literature, among them, e.g., the number of data items occurring in the input, as well as the total number of bits needed for storing the entire input). The ultimate goal when designing a one-pass algorithm is to keep the processing time per data item and the memory size *sublinear*, preferably polylogarithmic, in N. In particular, one typically aims at algorithms whose memory size is far smaller than the size of the input.

Key Points

The design and study of one-pass algorithms has a long tradition in many areas of computer science. For

example, they are used in the area of *data stream processing*, where streams of huge amounts of data have to be monitored *on-the-fly* without first storing the entire data. A deterministic finite automaton on words can be viewed as a (very simple) example of a one-pass algorithm whose memory size and processing time per data item is constant, i.e., does not depend on the input size. For most computational problems, however, the amount of memory necessary for solving the problem grows with increasing input size. *Lower bounds* on the memory size needed for solving a problem by a one-pass algorithm are usually obtained by applying methods from *communication complexity* (see, e.g., [1,2] for typical examples).

For many concrete problems it is even known that the memory needed for solving the problem by a deterministic one-pass algorithm is at least linear in the size N of the input. For some of these problems, however, *randomized* one-pass algorithms can still compute good *approximate* answers while using memory of size sublinear in N (cf. [1,2,3]). Typically, such algorithms are based on *sampling*, i.e., only a "representative" portion of the data is taken into account, and *random projections*, i.e., only a rough "sketch" of the data is stored in memory (see [3] for a comprehensive survey of according algorithmic techniques).

In the context of database systems these techniques are relevant, for example, for maintaining information needed for cost-based query optimization, e.g., estimates for the number of distinct values of an attribute, or the self-join size of a database relation. Efficient one-pass algorithms for incrementally updating these estimates can be found in [1].

In some application areas, rather than just a single pass, a small number P of sequential passes over the data may be available; the resulting algorithms are called *multi-pass algorithms* (see e.g., [2] for an analysis of the trade-off between the memory size and the number of passes necessary for solving particular problems).

Cross-references

- ▶ Approximate Query Processing
- ▶ Clustering on Streams
- ▶ Data Stream
- ▶ Data Sketch/Synopsis
- ▶ Event and Pattern Detection over Streams
- ▶ Stream Processing
- ▶ XML Stream Processing

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. J. Comput. Syst. Sci., 58:137–147, 1999.
2. Henzinger M., Raghavan P., and Rajagopalan S. Computing on data streams. In External Memory Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 50. American Mathematical Society, Boston, MA, USA, 1999, pp. 107–118.
3. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):117–236, 2005.

One-Way Hash Functions

▶ Hash Functions

Online Advertising

▶ Web Advertising

On-Line Analytical Processing

Alberto Abelló, Oscar Romero
Polytechnic University of Catalonia, Barcelona, Spain

Synonyms

OLAP

Definition

On-line analytical processing (OLAP) describes an approach to decision support, which aims to extract knowledge from a data warehouse, or more specifically, from data marts. Its main idea is providing navigation through data to non-expert users, so that they are able to interactively generate ad hoc queries without the intervention of IT professionals. This name was introduced in contrast to on-line transactional processing (OLTP), so that it reflected the different requirements and characteristics between these classes of uses. The concept falls in the area of business intelligence.

Historical Background

From the beginning of computerized data management, the possibility of using computers in data analysis has been evident for companies. However, early

analysis tools needed the involvement of the IT department to help decision makers to query data. They were not interactive at all and demanded specific knowledge in computer science. By the mid-1980s, executive information systems appeared introducing new graphical, keyboard-free interfaces (like touch screens). However, executives were still tied to IT professionals for the definition of ad hoc queries, and prices of software and hardware requirements where prohibitive for small companies. Eventually, cheaper and easy-to-use spreadsheets became very popular among decision makers, but soon it was clear that they were not appropriate for using and sharing huge amounts of data. Thus, it was in 1993 that Codd et al. [2], coined the term OLAP. In that report, the authors defined 12 rules for a tool to be considered OLAP. These rules caused heated controversy, and they did not succeed as Codd's earlier proposal for relational database management systems (RDBMS). Nevertheless, the name OLAP became very popular and is broadly used.

Although the name OLAP comes from 1993 and the idea behind them goes back to the 1980s, there is not a formal definition for this concept, yet. As proposed by Nigel Pendse [6], OLAP tools should pass the FASMI (fast analysis of shared multidimensional information) test. Thus, they should be fast enough to allow interactive queries; they should help analysis task by providing flexibility in the usage of statistical tools and what–if studies; they should provide security (both in the sense of confidentiality and integrity) mechanisms to allow sharing data; they should provide a multidimensional view so that the data cube metaphor can be used by users; and, finally, they should also be able to manage large volumes of data (gigabytes can be considered a lower bound for volumes of data in decision support) and metadata. However, there are not measures and thresholds for all these characteristics in order to be able to establish whether one of them is fulfilled or not, and therefore it is always arguable that a given tool fulfills them. Nevertheless, it is generally agreed that in order to be considered an OLAP tool, it must offer a multidimensional view of data.

Since their first days, OLAP tools have been losing weight and lowering prices, while at the same time, offering more functionality, better user interfaces and easier administration. Thus, time has come for small companies to use OLAP. They can afford it and they are willing to use it in their decision processes. Part of OLAP industry was associated into the OLAP Council (created in January 1995), whose aim was the promotion and standardization of OLAP terminology and technology. However, some major vendors never became members of this council, so eventually it disappeared (last news date from 1999). Nowadays, there is no standardization institution specifically devoted to OLAP. Therefore, it seems difficult to have a standard data model and query language in the near future, despite the fact that it is clearly desirable.

Foundations

OLAP environments have completely different requirements, compared to OLTP. Figure 1 summarizes the main differences. Firstly, their usage is different. While OLTP systems are conceived to solve a concrete problem and are used in the daily work of companies, OLAP systems are used in decision support. Thus, in the first case, since the addressed problem can be completely specified, the workload of the system is clearly predefined. Conversely, a decision support system aims to solve new problems every day. Therefore, ad hoc queries are executed. OLTP systems read as well as write data, while OLAP systems are considered read-only, because decision makers do not directly modify data. Nevertheless, the queries in a decision support system are much more complex, since they usually include big volumes of information processed by joining several tables, grouping data and calculating functions. Queries in OLTP systems do not usually involve volumes of data of the same magnitude, neither as many tables, nor groupings or calculations. The

	OLTP	OLAP
Usage	Application specific	Decision support
Workload	Predefined	Unforeseeable
Access	Read/Write	Read-only
Query structure	Simple	Complex
Records per operation	Tens/Hundreds	Thousands/Millions
Number of users	Thousands/Millions	Tens/Hundreds

On-Line Analytical Processing. Figure 1. Comparing OLTP Versus OLAP.

number of records in OLTP operations can be estimated as tens or hundreds at most, while OLAP queries usually involve thousands or even millions of records. Finally, the number of users is also different in both kinds of systems. OLTP systems can have thousands or millions of users (like in the case of cash machines), while OLAP systems have tens or maybe hundreds of users.

The main characteristic of OLAP is multidimensionality. The data cube metaphor is used to make user interaction easier and closer to decision makers' way of thinking, who would probably find SQL or any other text-based query language hard to understand and error prone. Thus, it is much easier for them to think in terms of the multidimensional model, where a Fact is a subject of analysis and its Dimensions are the different points of view that analysts could use to study the Fact. In this way, the instances of a Fact are shown in an *n*-dimensional space usually called Cube or Hypercube.

In order to show *n*-dimensional Cubes in two-dimensional interfaces, Cross-tabs or Statistical Tables such as the one in Fig. 2 (its data is entirely fictitious) are used. While in relational tables it is found that fixed columns and different instances are shown in each row, in Cross-tabs both columns and rows are fixed and interchangeable. In this example, you see three dimensions (i.e., Product, Place, and Year) that show the different points of view to analyze the OLAP tools market.

Multidimensionality is based on this fact-dimension dichotomy. A Dimension is considered to contain a hierarchy of aggregation levels representing different granularities (or levels of detail) to study data, and an aggregation level to contain descriptive attributes. On the other hand, a Fact contains quantitative attributes that are called measures. Dimensions of analysis arrange the multidimensional space where the Fact of study is depicted. Each instance of data is identified (i.e., placed in the multidimensional space) by a point in each of its analysis dimensions. Two different instances of data cannot be spotted in the same point of the multidimensional space. Therefore, given a point in each of the analysis dimensions they only determine one, and just one, instance of factual data. Moreover, data summarization that is performed must be correct, i.e., aggregated categories must be a partition (complementary and disjoint) and the kind of measure, aggregation function, and the dimension along which data is aggregated must be compatible. For example, stock, sum and time are not compatible, since stock measures cannot be added along temporal dimensions.

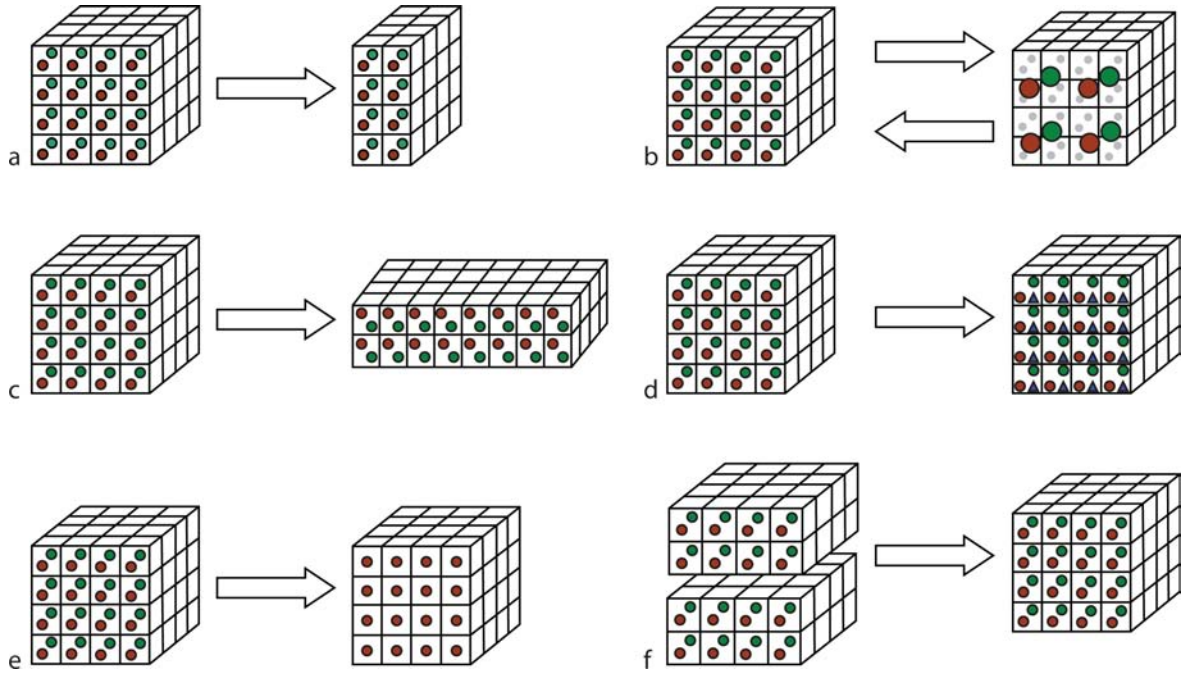
Market (billion US$)	ROLAP tools		MOLAP tools	
	Europe	USA	Europe	USA
2005	1	2	0.75	0.25
2006	1.5	2.5	1	0.5

On-Line Analytical Processing. Figure 2. Example of cross-tab or statistical table representation of a $2 \times 2 \times 2$ data cube.

Operations

Unfortunately, there is no consensus on the set of multidimensional operations and how to name them. However, [10] provides a comparison of algebraic proposals in the academic literature, as well as a set of operations subsuming all of them. A sequence of these operations is known as an OLAP session. An OLAP session allows transformation of a starting query into a new query. Figure 3 draws the transitions generated by each one of these operations (circles and triangles represent different attributes for Fact instances):

1. *Selection or dice.* By means of a logic predicate over the dimension attributes, this operation allows users to choose the subset of points of interest out of the whole *n*-dimensional space (Fig. 3a).
2. *Roll-up.* Also called "Drill-up", it groups cells in a Cube based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relationship which relates instances of two aggregation levels in the same Dimension, corresponding to a part-whole relationship (Fig. 3b from left to right). For example, it is possible to roll-up monthly sales into yearly sales moving from "Month" to "Year" aggregation level along the temporal dimension.
3. *Drill-down.* This is the counterpart of Roll-up. Thus, it removes the effect of that operation by going down through an aggregation hierarchy, and showing more detailed data (Fig. 3b from right to left).
4. *ChangeBase.* This operation reallocates exactly the same instances of a Cube into a new *n*-dimensional space with exactly the same number of points (Fig. 3c). Actually, it allows two different kinds of changes in the space: rearranging the multidimensional space by reordering the Dimensions, interchanging rows and columns in the Cross-tab (this



On-Line Analytical Processing. Figure 3. Schema of operations on cubes.

is also known as Pivoting), or adding/removing dimensions to/from the space.

5. *Drill-across*. This operation changes the subject of analysis of the Cube, by showing measures regarding a new Fact. The *n*-dimensional space remains exactly the same, only the data placed in it change so that new measures can be analyzed (Fig. 3d). For example, if the Cube contains data about sales, this operation can be used to analyze data regarding production using the same Dimensions.
6. *Projection*. It selects a subset of measures from those available in the Cube (Fig. 3e).
7. *Set operations*. These operations allow users to operate two Cubes defined over the same *n*-dimensional space. Usually, Union (Fig. 3f), Difference and Intersection are considered.

This set of algebraic operations is minimal in the sense that none of the operations can be expressed in terms of others, nor can any operation be dropped without affecting functionality (some tools consider that the set of measures of a Fact conform to an artificial analysis dimension, as well; if so, Projection should be removed from the set of operations in order to be considered minimal, since it would be done by Selection over this artificial Dimension). Thus, other operations can be derived by sequences of these. It is the case of Slice (which reduces the dimensionality of the original Cube by fixing a point in a Dimension) by means of Selection and ChangeBase operations. It is also common that OLAP implementations use the term Slice&Dice to refer to the selection of fact instances, and some also introduce Drill-through to refer to directly accessing the data sources in order to lower the aggregation level below that in the OLAP repository or data mart.

Declarative Languages

There are some research proposals of declarative query languages for OLAP. Cabibbo and Torlone [1] propose a graphical query language, while Gyssens and Lakshmanan [3] propose a calculus. From the industry point of view, MDX (standing for multidimensional expressions) [5] is the de facto standard. It was introduced in 1997, and in spite of the specification being owned by Microsoft, it has been widely adopted. Its syntax resembles that of SQL:

```
[WITH <MeasureDefinition>+]
SELECT <DimensionSpecification>+
FROM <CubeName>
[WHERE <SlicerClause>]
```

However, its semantics are completely different. Roughly speaking, an MDX query gets the instances of a given Cube stated in the FROM clause and places them in the space defined by the SELECT clause. Moreover, complex calculations can be defined in the WITH clause, and the dimensions not used in the SELECT clause can be sliced in the WHERE clause (if not explicitly sliced, it is assumed that dimensions that do not appear in the SELECT are sliced at the higher aggregation level: All).

```
WITH MEMBER [Measures].[pending] AS
'[Measures].[Units Ordered]-[Mea-
sures].[Units Shipped]' SELECT
 {[Time].[2006].children} ON COLUMNS,
 {[Warehouse].[Warehouse Name].mem-
bers} ON ROWS
FROM Inventory
WHERE ([Measures].[pending],[Trade-
mark].[Acme]);
```

In the previous MDX query, an ad hoc measure "pending" is first defined as the difference between units ordered and shipped. Then, the children of the instance representing year 2006 (i.e., the 12 months of that year) are placed on columns, and the different members of the aggregation level "Warehouse Name" on rows. Now, this matrix is filled with the data in "Inventory" cube, showing the previously defined measure "pending" and slicing "Acme" trademark.

Key Applications

Managers are usually not trained to query databases by means of SQL. Moreover, if the query is relatively complex (several joins and subqueries, grouping, and functions) and the database schema is not small (with maybe hundreds of tables), using interactive SQL could be a nightmare even for SQL experts. Thus, OLAP is used to ease the tasks of these managers in extracting knowledge from the data warehouse by means of Drag&Drop, instead of typing SQL queries by hand.

OLAP market is estimated around US$ 6 billion in 2006, which is mainly devoted to decision making. However, this paradigm can also be used in any other field with non-expert users, where schemas and queries are relatively complex. For example, its usage is under investigation in bioinformatics [8], and the semantic web [9].

Future Directions

OLAP is used to extract knowledge from the data warehouse. Data mining tools can also be used for this purpose. Until now, both research communities have been evolving separately. The former must be interactive, while the latter presents computational complexity problems. However, it seems promising to integrate both kinds of tools so that one can benefit from the other [4]. Some tools like Microsoft Analysis Services already integrate them in some way. Nevertheless, there is still much work to do in this field.

On the other hand, security is usually a flaw in data warehousing projects. Reference [7] contains a survey of OLAP security problems. In the past, OLAP tools used to have just a few users and all of them had high responsibilities in the organization, so this was not really a concern in the sense of confidentiality. Nowadays, with the increase in potential users of OLAP systems inside as well as outside the organization, security has emerged as a priority in these projects. Moreover, personal data (like those of customers) are usually analyzed in almost all companies. Thus, inference control mechanisms need to be studied in data mining as well as OLAP tools.

Other research directions in OLAP can be the improvement of user interaction and flexibility in the calculation of statistics, and the integration of what–if analysis (see What–if Analysis definitional entry).

Url to Code

Some OLAP vendors:

1. Microsoft Analysis Services: http://www.microsoft.com/sql/technologies/analysis/default.mspx
2. Hyperion Solutions: http://www.hyperion.com
3. Cognos PowerPlay: http://www.cognos.com/products/business_intelligence/analysis/index.html
4. Business Objects: http://www.businessobjects.com/products/queryanalysis/olapaccess/businessobjects.asp
5. MicroStrategy: http://www.microstrategy.com/Solutions/5Styles/olap_analysis.asp

Some open source OLAP tools:

1. Mondrian: http://mondrian.pentaho.org
2. Palo: http://www.palo.net

Cross-references

- ▶ Business Intelligence
- ▶ Cube Implementations
- ▶ Database Management System
- ▶ Data Mart
- ▶ Data Mining
- ▶ Data Warehouse
- ▶ Dimension
- ▶ Hierarchy
- ▶ Hierarchical Data Summarization
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ Star Schema
- ▶ Summarizability
- ▶ Visual On-Line Analytical Processing (OLAP)

Recommended Reading

1. Cabibbo L. and Torlone R. From a procedural to a visual query language for OLAP. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management. 1998, pp. 74–83.
2. Codd E.F., Codd S.B., and Salley C.T. Providing OLAP to user-analysts: An IT mandate. Technical Report, E. F. Codd & Associates, 1993.
3. Gyssens M. and Lakshmanan L.V.S. A foundation for multi-dimensional databases. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 106–115.
4. Han J. OLAP Mining: Integration of OLAP with Data Mining. In Proc. IFIP TC2/WG2.6 Seventh Conf. Database Semantics, 1997, pp. 3–20.
5. Microsoft. Multidimensional Expressions (MDX) Reference. Available at http://msdn2.microsoft.com/en-us/library/ms145506.aspx, 2007. SQL Server books online.
6. Pendse N. The OLAP Report – What is OLAP? Available at http://www.olapreport.com/fasmi.html, 2007. Business Application Research Center.
7. Priebe T. and Pernul G. Towards OLAP Security Design – Survey and Research Issues. In Proc. ACM Int. Workshop on Data Warehousing and OLAP, 2000, pp. 33–40.
8. Rahm E., Kirsten T., and Lange J. The GeWare data warehouse platform for the analysis of molecular-biological and clinical data. J. Integr. Bioinformat., 1(4):47, 2007.
9. Romero O. and Abelló A. Automating Multidimensional design from ontologies. In Proc. ACM Int. Workshop on Data Warehousing and OLAP, 2007, pp. 1–8.
10. Romero O. and Abelló A. On the need of a reference algebra for OLAP. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 2007, pp. 99–110.

Online Handwriting

- ▶ Electronic Ink Indexing

Online Recovery

- ▶ Crash Recovery

Online Recovery in Parallel Database Systems

Ricardo Jimenez-Peris
Universidad Politecnica de Madrid, Madrid, Spain

Synonyms

High availability; Continuous availability; 24x7 operation

Definition

Replication (also known as clustering) is a technique to provide high availability in parallel and distributed databases. High availability aims to provide continuous service operation. High availability has two faces. On one hand, it provides fault-tolerance by introducing redundancy in the form of replication, that is, having multiple copies or replicas of the data at different sites. On the other hand, since sites holding the replicas may crash and/or fail, in order to keep a given degree of availability, failed or new replicas should be reintroduced into the system. Introducing new replicas requires transferring to them the current state in a consistent fashion (known as *recovery*). A simple solution to this problem is *offline recovery*, that is, in order to obtain a quiescent state, request processing is suspended, then the state is transferred from a working replica (termed *recoverer replica*) to the new replica (*recovering replica*) and finally, request processing is resumed. Unfortunately, offline recovery results in a loss of availability, which defeats the original goal of replication, that is to provide high availability. The alternative is *online recovery*, in which transaction processing is not stopped while the recovery is performed. The main challenge for online recovery is to attain consistency, since the state to be transferred to the recovering replica(s) is a moving target. While the recovery takes place, new transactions are processed and the state evolves during the recovery itself. Online recovery needs to be coordinated with the replica control protocol to enforce consistency.

Historical Background

Recovery is used in centralized databases to bring the database to a consistent state after a crash [1]. The consistency is attained by ensuring that the updates of committed transactions are reflected in the database and the updates of uncommitted (aborted) transactions are not reflected in it. In clustered databases, centralized recovery is used to bring a failed replica to a local consistent state, but then, since other working replicas may have processed transactions, centralized recovery is not sufficient and it has to be followed by a replica recovery [11]. During replica recovery, the failed replica (or a fresh new one) recovers the current state from the working replicas. What is meant by *current state* the state reflecting all the updates from transactions that will not be processed by the recovering replica, and not reflecting any of the updates from transactions that will be processed by the recovering replica after recovery. A failed replica only needs to recover the missed updates, while a new replica needs to recover the full database.

The seminal paper on online recovery for clustered databases is [8]. In this paper, a suite of protocols for online recovery is proposed. One of the protocols lies in a locking-based online recovery. The full database is locked atomically using recovery locks, a special kind of read lock. This guarantees a quiescent state of the database. Then, the recovery locks are released as data is transferred to the recovering replica. The atomic setting of the recovery locks acts as a synchronization point. All update requests processed before the recovery lock setting should be reflected in the transferred state. Requests submitted after the recovery locks are set should be processed by the recovering replica after recovery finishes. This protocol lies inbetween offline and online recovery. In the beginning, all the data are locked, and therefore the database is unavailable. This situation improves as recovery progresses, since recovery locks are released and the corresponding data become available. Another online recovery protocol proposed in [8] is a multi-round recovery. In this protocol, the state to be transferred (missed updates) to the recovering replica is sent in rounds. The first round would contain all the updates missed until the start of recovery. In the second round, the updates performed during the first round are sent, and so on. When the number of updates performed during the last round is small enough, a last round is run. During the last round, the recovering replica has to store all client requests to process them after finishing recovery. This recovery protocol if fully online and also significantly reduces the number of transactions to be stored during recovery, since it is only during the last round that the recovering replica has to store incoming transactions (typically only the resulting updates from them).

Another piece of work on online recovery was presented in [5]. This paper describes a log-based online recovery protocol in which a failed replica receives the prefix of the log corresponding to the update transactions it has missed. Since the log grows as the recovery progresses, the protocol has a special handshake protocol to finish recovery. The recoverer traverses the log from the first transaction the failed replica missed until it reaches the end of the log. At this point, the end recovery handshake protocol is started to determine which will be the last transaction to be sent as part of the recovery. The recovering replica starts storing requests that follow this transaction to process them after recovery finishes.

Online recovery has also been used for replicated data warehouses across the Internet [9]. In this work, each replica is located at an autonomous organization and exhibits an interface to execute queries. The online recovery protocol exploits the underlying architecture and performs recovery by issuing queries from the recovering replica to the working replicas. It also takes advantage of a facility for historical queries that enables executing a read only query providing a timestamp T. This historical query will return the same results as it happened at time T. The recovery protocol has 3 phases. In the first phase, the recovering site determines the latest time T for which it has all the committed updates. In the second phase, the recovering site runs historical queries at working replicas (that act as recoverer replicas) with a timestamp between the recovery point and a time closer to the present to catch up missed updates. Historical queries do not set read locks, and therefore do not block updates at the recoverer sites. In the final phase, a regular query (non-historical) is run to get the latest updates. In this case, read locks are set to guarantee the consistency.

Online recovery has also been studied in other contexts, such as diverse data replication and Byzantine data replication. In diverse data replication [4], each replica runs a database from a different vendor. This approach enables tolerating software failures since it has been observed that bugs in one product typically

do not appear in a database from a different vendor [4]. In diverse data replication, recovery is slightly more complex, since it requires using a common abstract representation of the data. This might need some tuple translation during recovery in order to align fields with slightly different types.

Byzantine data replication [17] tolerates intrusions and provides continuous correct service despite them. An intrusion happens at a site when it is attacked. This site may behave arbitrarily to disrupt service provision. Intrusions are modeled as Byzantine (also known as arbitrary) failures. Byzantine replication typically resorts to diverse data replication to avoid common vulnerabilities. Typically tolerating f Byzantine failures requires $3 \cdot f+1$ replicas. Byzantine data replication has a more involved recovery, since it should also mask Byzantine recoverers. This means that a sufficient number of recoverers is needed in order to mask Byzantine failures during recovery [2]. An additional issue in Byzantine fault tolerance is that once one replica has been successfully attacked, another one can be attacked, and so on. In order to reduce the window of vulnerability, proactive recovery has been proposed [2]. This approach recovers replicas proactively without waiting until they are crashed or attacked. During recovery, the recovering replica boots from read-only media and recovers the state from working replicas using a Byzantine recovery protocol. During recovery, the state is transferred from multiple working replicas to be able to tolerate Byzantine (attacked) recoverers. Replicas are recovered in a round-robin fashion forced by a reboot provoked by a hardware watchdog. In this way, even if a replica has been attacked silently (without the system and administrator noticing it), it will become operative again, thus, reducing the window of vulnerability of the system.

Foundations

High availability consists of two inseparable aspects. On one hand, it requires the ability to tolerate failures. This is typically achieved by introducing redundancy in the form of replication. However, in order to keep a given degree of availability, the ability to recover failed (or new) replicas is also necessary. Recovering failed replicas requires obtaining a quiescent state from a working replica (or *recoverer replica*) and transferring it to the new replica (or *recovering replica*). This quiescent state can be easily obtained by stopping transaction processing. This results in *offline recovery*. Although offline recovery guarantees the consistency of recovery, its major drawback is that it results in a loss of availability during the recovery process that defeats the original goal of replication, which is to provide high availability. The alternative is *online recovery*, that is, to transfer the state to a recovering replica without stopping transaction processing.

Replication can be used to provide scalability in addition to providing availability. In this case, availability results insufficient as a metrics to express the goodness of a recovery protocol [5]. Performability becomes a more appropriate metrics in this context. *Performability* is defined as the cumulative performance of a highly available system over a period of time in the presence of failures and recoveries. To understand why it is important, an extreme situation will be illustrated. A replicated system has a throughput of 1000 transactions per second (tps). During online recovery the system remains available, but its throughput decreases to 1 tps. Although this system is available, it is clearly worse than a system that would deliver 990 tps during the online recovery (assuming that online recovery takes the same amount of time in both systems). However, when comparing the performability of both systems during online recovery, the former system would offer a very poor performability, while the second would offer a very high one close to the one in which the system is not recovering any replica.

Online recovery protocols typically consist of five phases:

1. *Local Recovery* brings the local state to a consistent state by means of centralized recovery.
2. *Find Last Committed Update Transaction* determines the last update transaction reflected in the local state.
3. *Global Recovery Start* initiates online recovery taking care of obtaining a quiescent state from one or more working replicas to be transferred to the recovering replica.
4. *Global Recovery* transfers a quiescent state from a working replica to the recovering replica.
5. *Global Recovery End* is the handshake protocol to determine the end of recovery.

The first and second phases depend on whether the recovering replica is a failed replica or a fresh new replica. For a failed replica, the first phase is typically performed automatically by the underlying database system upon recovery after a crash. For a failed replica,

the second phase implies traversing the log or any other recovery information repository to find out which was the last committed update transaction. In most protocols, this information does not need to be precise. It can be enough to obtain the identifier of a committed transaction (e.g., a timestamp, log sequence number or transaction identifier, TID) close to the failure instant such that all previously committed transactions are also reflected in the local state. If some updates of latter transactions are reflected in the state, they will be rewritten during recovery. For a fresh new replica, the first phase is empty and the second phase first involves obtaining a checkpoint of the database and then performing the same processing as a failed replica. The checkpoint of the database also needs to be obtained in an online fashion using techniques such as point-in-time recovery [16] in order to keep the system available.

In the third phase, there is some communication between the recovering replica and the working replicas. This interaction has several purposes. First, the working replicas become aware of the new replica wanted to join the system. Second, the recovering replica informs the working replicas about the last known update. Third, a recoverer is elected to transfer the state to the recovering replica.

The fourth phase transfers the state from the recover to the recovering replica. The recovery process is synchronized with replica control to guarantee that a quiescent state is transferred to the recovering replica.

The fifth phase aims at finishing the recovery, which requires splitting the sequence of transactions into two disjoint sets (the ones whose state is reflected in the state transfer to the recovering replica, and the ones that should be processed by the recovering replica after finishing the recovery).

Online recovery depends on the specific features of the replica control protocol being used, such as eager vs. lazy, primary-backup vs. update-everywhere, kernel-based vs. middleware-based replication, etc. In eager replication, the coordination between replica control and online recovery is very tight to guarantee consistency [6,7,13,14,10]. In lazy replication, the coordination can be looser. For instance, in a freshness-based approach [12,3], as far as the freshness requirement is satisfied the coordination can be more relaxed. In a primary-backup approach, the recovery of backups is simpler since they do not execute updates on their own, they just apply the updates coming from the primary [15]. In update-everywhere approaches [6,7,13,14,10] the recovery needs to be interleaved carefully with replica control to guarantee consistency [8,5]. In kernel-based approaches, it is possible to use recovery protocols that use mechanisms within the kernel (such as locking) [8]. However, in middleware-based approaches, the recovery can only use those mechanisms available at the database interface [5].

In this entry, two approaches will be examined in detail: an online recovery for kernel-based replication based on locking [8], and an online recovery for middleware-based replication based on logging [5].

First, a locking-based online recovery approach is studied. This approach is based on the most basic protocol from [8]. This basic version of the protocol transfers the full database. The recovery is coordinated with replica control to guarantee consistency. This coordination is materialized through locking. In what follows, the first generic phases of online recovery for this particular protocol are described. The first phase, local recovery, is orthogonal to online recovery it can just be ignored. The second phase consists in determining the last update known by the recovering replica. Since in this approach the full database is sent, this phase does not exist.

The third phase is recovery start. The recovering replica notifies to the working replicas that is willing to join the system and recover. In this protocol, in order to guarantee the quiescence of the transferred state, the recovery is started by initiating a transaction that sets atomically special read locks or recovery locks over all the tuples in the database. The quiescent state to be transferred corresponds to the state just after the atomic setting of the recovery locks. The recovery then takes place gradually. As soon as a recovery lock is granted, the tuple is read and sent to the recovering replica. Then, the lock is released. It should be noted that recovery locks are read locks and therefore do not delay read operations, only update ones. During the recovery, the recovering replica should store the incoming transactions (only those involving updates) to process them after recovery, since the associated updates are not incorporated in the state being transferred to it.

The end-of-recovery handshake is simple in this protocol. It is initiated after the sending of the last tuple, and the recovery message piggybacks is marked to indicate this fact. After receiving this message, the recovering replica starts to process all the stored

transactions during the recovery. Depending on the replica control in place the recovering replica will execute the update transactions, or will receive the resulting updates from the other replicas (which is usually the case). In the latter case, incoming transactions do not need to be stored-just the update propagation messages from the other replicas.

There are many optimizations that can be performed over this basic protocol as described in [8]. Batching recovery messages limit the amount of data transferred during recovery by recording the updates performed during recovery, shortening the amount of updates to be stored during online recovery by using multiple phases, etc.

The second protocol that will be described is a log-based protocol [5]. This protocol is combined with a pessimistic replica control implemented as a middleware layer. A pessimistic replica control freely executes non-conflictive transactions in the replicated system, while conflictive ones are executed in the same relative order at all replicas. The replica control protocol over which online recovery is built is Nodo, described in [13]. Nodo is based on the notion of conflict classes. A transaction might access one or more conflict classes. Each conflict class has a master replica. Each combination of conflict classes also has a master replica, one of the master replicas of the individual conflict classes. Each conflict class has an associated transaction queue. Update transactions are sent to all replicas in the same order. Read-only transactions are only sent to one of the replicas. Transactions are queued in the conflict class queues relevant to them (the ones that they read or write). When a transaction is at the front of all the queues it has been enqueued, the master of the conflict class combination associated to the transaction executes it locally. If the transaction is read-only, then the results are returned directly to the client and removed from all the queues. If the transaction contains updates, the master extracts them from the database and sends them to all other replicas. Nodo keeps a log for each individual conflict class that is exploited by the online recovery protocol.

The online recovery protocol for Nodo guarantees consistent recovery by careful coordination with the replica control protocol of Nodo. Individual conflict classes can be recovered independently, that is, each individual conflict class could use a different recoverer replica. The recovery of an individual conflict class is detailed in what follows. The first phase, local recovery, occurs when the replica recovers. The second phase, determining the last update reflected in the replica state, is done by looking at the local log. The recovering replica traverses the log to determine the identifier of the last transaction processed for the conflict class being recovered. This identifier is used in the fourth phase to initiate the recovery of the conflict class. One of the working replicas is elected as recoverer replica for this conflict class. In the fourth phase, the recoverer replica traverses the log of the conflict class being recovered. The recoverer replica continues processing updates involving this conflict class. This means that the log grows as is being traversed. The recovering replica just applies all the updates corresponding to the conflict class under recovery, but will discard them, since it is receiving them via the recovery. The recoverer replica will eventually reach the end of the log. At this point the fifth phase is performed to finish recovery. In the end-of-recovery handshake three different roles are involved, namely, the recoverer and recovering replicas, and the master replica of the conflict class under recovery. There is a race condition, since the master produces updates from locally executed transactions, while the recoverer needs to know the last update to be forwarded to the recovering replica and the recovering replica needs to know from which transaction it has to start to process update transactions. When the recoverer reaches the end of the log, it sends a request-end-of-recovery message to all replicas (only the master needs this message in the failure-free case). The master will then piggyback with the next update it forwards an end-of-recovery marker. This marker will indicate to the recoverer that this is the last update to be sent to the recovering replica. The marker will tell the recovering replica which is the last update part of the recovery, and therefore it will know from which point it has to apply updates received from the master replica.

The online recovery for Nodo, as stated before, allows recovering conflict classes independently. In fact, once a conflict class is recovered, the recovering replica could become master of that class. Additionally, several conflict classes can be recovered in parallel using different recoverers for each of them. The online recovery for Nodo has been designed to be adaptive. The goal is to obtain the highest performability.

If the replicated system has a low load, the resources employed to recover the replica are increased (i.e., increasing the number of recoverers). If there is a peak load, during recovery, then the resources devoted to recovery can be decreased (i.e., decreasing the number of recoverers). In this way, performability is maximized. Nodo also enables dealing with overlapping recoveries in parallel. If a batch of sites is recovered in a time interval, they will start recovery at slightly different times. The recovery protocol takes care of recovering simultaneously conflict classes for all recovering replicas it knows. In this way, if one replica starts recovery after another one has recovered two conflict classes, they will perform simultaneously the recovery of the remaining N-2 conflict classes, being N the number of conflict classes, and alone the recovery of the first two conflict classes. This enables a more efficient dissemination of the recovery messages (e.g., exploiting multicast) and efficiently recovering batches of replicas.

Key applications

Online recovery is a crucial technique to provide true high availability, that is, 24×7 operation. Its main potential users are all those organizations that provide services that should be continuously available. Among these potential users one can find enterprise data centers, software as a service (SaaS) platforms, services governed by service level agreements (SLAs), services for critical infrastructures (health-care, energy, police, etc.).

Cross-references

► Data Replication
► Replica control

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
2. Castro M. and Liskov B. Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4):398–461, 2002.
3. Gançarski S. , Naacke H., Pacitti E., and Valduriez P. The leganet system: Freshness-aware transaction routing in a database cluster. Inform. Syst., 32(2):320–343, 2007.
4. Gashi I., Popov P., and Strigini L. Fault Tolerance via Diversity for Off-The-Shelf Products: a Study with SQL Database Servers. IEEE Trans. Depend. Secur. Comput., 4(4):280–294, 2007.
5. Jiménez-Peris R., M. Patiño-Martínez, and Alonso G. Non-Intrusive, Parallel Recovery of Replicated Data. In Proc. 21st Symp. on Reliable Distributed Syst., 2002, pp. 150–159.
6. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 134–143.
7. Kemme B. and Alonso G. A New Approach to Developing and Implementing Eager Database Replication Protocols. ACM Trans. Database Syst., 25(3):333–379, 2000.
8. Kemme B., Bartoli A., and Babaoglu O. Online Reconfiguration in Replicated Databases Based on Group Communication. In Proc. Int. Conf. on Dependable Systems and Networks, 2001, pp. 117–130.
9. Lau E. and Madden S. An Integrated Approach to Recovery and High Availability in an Updatable, Distributed Data Warehouse. In Proc. 32nd Int. Conf. on Very Large Data Bases. 2006, pp. 703–714.
10. Manassiev K. and Amza C. Scaling and Continuous Availability in Database Server Clusters through Multiversion Replication. In Proc. Int. Conf. on Dependable Systems and Networks, 2007, pp. 666–676.
11. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems. Prentice-Hall, 2nd ed., 1999.
12. Pacitti E. and Simon E. Update Propagation Strategies to Improve Freshness in Lazy Master Replicated Databases. VLDB J., 8 (3):305–318, 2000.
13. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent Database Replication at the Middleware Level. ACM Trans. Comput. Syst., 23(4):375–423, 2005.
14. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distributed and Parallel Databases, 14(1): 71–98, 2003.
15. Plattner C. and Alonso G. Ganymed: Scalable Replication for Transactional Web Applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
16. PostgreSQL PostgreSQL Point in Time Recovery. http://www.postgresql.org/docs/8.0/interactive/backup-online.html.
17. Vandiver B., Balakrishnan H., Liskov B., and Madden S. Tolerating Byzantine Faults in Database Systems using Commit Barrier Scheduling. In Proc. 21st ACM Symp. on Operating System Principles, 2007, pp. 59–72.

Ontological Engineering

► Ontology
► Ontology Engineering

Ontologies

► Gazetteers

Ontologies and Life Science Data Management

Robert Stevens[1], Phillip Lord[2]
[1]University of Manchester, Manchester, UK
[2]Newcastle University, Newcastle-Upon-Tyne, UK

Synonyms

Knowledge management

Definition

Biology is a knowledge-rich discipline. Much of bioinformatics can, therefore, be characterized as *knowledge management*: organizing, storing and representing that knowledge to enable search, reuse and computation.

Most of the knowledge of biology is categorical; statements such as "fish gotta swim, birds gotta fly" cannot be easily represented as mathematical or statistical relationships. These statements can, however, be formalized using ontologies: a form of model which represents the key concepts of a domain.

Ontologies are now widely used in bioinformatics for a variety of tasks, enabling integration and management of multiple data or knowledge sources, and providing a structure for new knowledge as it is created.

Historical Background

Biological knowledge is highly complex. It is characterized not by the large size of the data sets that it uses, but by the large number of data types; from relatively simple data such as raw nucleotide sequence, through to anatomies, systems of interacting entities, to descriptions of phenotype.

In addition to its natural complexity, biology has traditionally operated as a "small science" – with a large number of individual, autonomous laboratories working independently. This has resulted in highly heterogeneous data; in addition to the natural complexity of the data, knowledge is often represented in many different ways [5]. There are, for example, at least twenty different file formats for representing DNA sequence.

Ontologies can be used to enable the knowledge management that overcomes these two forms of complexity. First, they can be used to represent complex, categorical knowledge of the sort common in biology. Second, by describing the heterogeneity of the representation of knowledge, they can provide a common, shared understanding that can be used to overcome this heterogeneity.

In post-genomic biology, the first of these has been the most common usage. Here, ontologies are used to generate a controlled vocabulary; for this, the Gene Ontology (GO) [14] provides the paragon for biological sciences. It represents three key aspects of genetics; the *molecular function* of a gene (product), the *biological process* in which the product is involved, and the *cellular component* in which it is located. GO has been used to annotate many genomes and has been used for annotations in UniProt and InterPro. Following on from the success of GO, the Open Biomedical Ontologies (OBO) now provides controlled vocabularies for describing many aspects of biological knowledge (http://obo.sf.net).

The second major use has been to enable access to or querying over multiple independent data sources. EcoCyc [8], for example, uses an ontology to provide a schema to integrate genome, proteome data and a number of pathway resources, while RiboWeb [1] was a similar style of ontology driven application for storing, managing and analyzing data from experiments on ribosomal structure. The TAMBIS system [7] used an ontology to mediate queries to a number of different data sources.

Most of these examples are *post-hoc* additions to existing systems; GO, for example, presents knowledge which is already present in other, less formal, representations. More recently, however, there has been a shift to the use of ontologies as a primary representation. The MGED Ontology (MO) [17] provides a vocabulary for reporting microarray experiments, while the Systems Biology Ontology (SBO) (http://www.ebi.ac.uk/sbo/) supports the representation of systems biology models.

Over the past decade, the use of ontologies has now become well-entrenched as a tool for organizing and structuring biomedical knowledge and, therefore, has become a key part of life science knowledge management [3].

Foundations

It was recognized early in bioinformatics that there is a massive problem with heterogeneous representations of data [5]. Such heterogeneities, particularly at the semantic level, exist both in the meanings of the structures that hold values (the schema) and in the meanings of the values themselves. To query or analyze meaningfully

across data from different sources, therefore, there is a necessary reconciliation step to enable the data to be understood. A database, for example, might have either separate tables for substrate or product, or just one for reactant; as an orthogonal issue, a chemical might be "acetic acid" in one or "ethanoic acid" in another. Both types of mis-match need to be overcome.

This heterogeneity can occur both in descriptions of biology and also bioinformatics: it is possible to disagree on which genes are involved in a process, what those genes are called and what structure is used to hold information about those genes.

Ontologies provide a computable mechanism for working around these problems; they describe the entities and what is known about these entities within the data of biology, and provide a set of labels to describe these entities and their properties. As a result, an ontology can be used to describe the entities within a biological database. Terms from the Gene Ontology, for example, are used to describe the major functional attributes of gene products in many databases; this, in turn, allows comparative studies of genes and their actions across species.

Ontologies need to be represented in a language; these are often called knowledge representation languages. They have a set of language elements for describing categories (also called classes, types, concepts or universals) of instances (also called objects, entities, individuals or particulars). The languages vary in their expressivity – that is, how much is it possible to say about what these elements mean. For example, if we state that *ArB*, where *A* and *B* are classes and *r* is a relationship, does this mean that every *A* has a *B* related to it by *r*; that for any *A* with a *r* relation, this *r* is to a *B*; to how many *B*'s can an *A* have a relationship; that *r* has an inverse relationship, and so on. Some languages allow only trees of categories to be made, others more complex graphs. Finally, languages differ in their computational amenability [11].

Ontologies have found a variety of uses within bioinformatics:

- **Reference Ontology:** An ontology can be used simply as a reference, encompassing what is understood about a domain with high-fidelity. Such an ontology is not skewed by any application bias, except that of correctness.
- **Controlled Vocabulary:** The labels on the categories in an ontology provide a vocabulary with which discussion of those categories can be accomplished. By committing to use that vocabulary – controlling the words used in communication – a controlled vocabulary is established. This is the principal means of overcoming a large portion of heterogeneity.
- **Computational Component:** An ontology can form a component in a software application. The strict semantics of its representation language can be used to make inferences from data described in terms of that ontology. This can simply be retrieving all the children of a given category (all instances of a child are also instances of the parent) to recognizing membership of a category based on facts known about an object.

  There are many technical and social difficulties associated with using an ontology for data management. The choice of representation language can be key; an ontology for use as a computational component probably needs a more computationally amenable and expressive language than an ontology intended to provide a controlled vocabulary. It is often hard to engage with domain experts to ensure that the ontology reflects the domain, while maintaining ontological precision. There are a number of methodologies for ontology building [6], but the discipline is still nascent. Finally, adapting and updating the ontology when it is already in use can require rigorous, yet flexible policies.

Key Applications

Perhaps the best known ontology in biology is the Gene Ontology (GO) [14]. It started in 1998 with an aim of enabling queries across multiple databases for the key aspects or properties of the genes or proteins; it achieves this not by schema reconciliation but enabling the augmentation of existing knowledge; in short, it is one of the best examples of a reference ontology.

Another domain well served by reference models is that of anatomy. The Foundational Model of Anatomy, for example, aims to provide a "symbolic modeling of the structure of the human body in a computable form that is also understandable by humans" [12]. The aim is that this ontology provides a common representation into which others can be mapped.

One of the early uses for ontologies was enabling schema and value reconciliation. The TAMBIS [7] system was an early example. It uses an ontological representation of entities and their properties in biology expressed in a description logic [2]. These

descriptions, that could be composed to more complex concept descriptions, were then transformed to queries against bioinformatics analysis services capable of retrieving instances of the concepts within the ontology. The ontology, then, could tell that the user that a `Protein` might have one or more `Homologs`, while the system would understand that a BLAST search might reveal these `Homologs`. A concept therefore also defined a query plan. More recently, the BioPAX ontology [10] provides a schema for representing biological pathway data; the different contributing databases could then release knowledge in the format. Both of these operate on the level of schema, but reconciliation to a common model also occurs at the level of the values held within a schema. The Gene Ontology, for example, in providing a controlled vocabulary for the functional attributes of gene products has allowed many genome resources to use common values within their schema.

Both BioPAX and TAMBIS enabled querying by assigning objects to categories in the ontology. Ontologies represent the properties by which objects can be recognized to be a member of a category. If these properties are recognizable computationally, then the ontology can be used to classify these members automatically. This approach has been used to classify the phosphatase proteins from three parasite genomes [4]. A final approach to ontological querying is to use the ontology as a basis for statistical analysis of individuals annotated with these ontologies. GO has been widely used for this purpose [9,15].

Finally, ontologies have begun to be used for the representation of metadata about primary experimental data. The MGED society has led the way with the MGED Ontology (MO) which has been used for describing microarray data[17]. This ontology describes a number of aspects about an experiment including: the biological material used; experimental design and microarray equipment. Similar work is now underway for describing proteomics experiments [13]. These are coming together in the Ontology for Biomedical Investigations OBI that is providing a general framework for describing the protocols and analyses for many different kinds of experiment [16].

Future Directions

In the past decade ontologies have come to form a major aspect of information management in the life sciences. The field of ontology development in the life sciences now faces several challenges in the short and medium term.

- The wide scope of biology is a challenge; to describe many parts of it, also needs descriptions of closely related areas such as chemistry, geology and geography.
- Ontologies are starting to get very large. It is not clear whether current methodologies are scalable, both in terms of building, maintaining or using them. This has many implications for the formal expressive structures of the knowledge representation language, the ability to support modularity of these languages, and the social processes used to build ontologies.
- Dealing with change both as a result of the ontology development process and, perhaps more importantly, as a result of changes in knowledge itself. There are many different techniques for dealing with the former situation; there are many fewer for dealing with the latter. If datasets gathered over a long period of time are to be understood in the future, it may become as important to understand what was thought in the past as it is to manage the current sate of knowledge.
- Currently many ontologies deal with a single level of granularity or the view point of a single discipline building sophisticated, computationally amenable ontologies necessitates crossing boundaries of granularity and discipline. It remains, however, unclear how to integrate these sorts of ontology.
- Ontologies currently fulfill the luxury end of the metadata market; they can be very expensive to build, maintain and deploy. Lower the cost is critical. Probably the best way to achieve this is to make them easier for domain scientists to build which leads to a second challenge; maintaining usability of ontologies and representation languages, while increasing their scale and computability.

It seems clear that ontologies will be in heavy use in the future within the life sciences. How well these challenges are answered will determine the uses to which they are put.

Cross-references

► Ontology
► Query Languages for Ontological Data

Recommended Reading

1. Altman R., Bada M., Chai X. Whirl Carillo M., Chen R., and Abernethy N. RiboWeb: an ontology-based system for collaborative molecular biology. IEEE Intell. Syst., 14(5):68–76, 1999.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. (eds.) The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.
3. Bodenreider O. and Stevens R. Bio-ontologies: current trends and future directions. Brief Bioinform., 7(3):256–274, 2006.
4. Brenchley R., Tariq H., McElhinney H., Szoor B., Stevens R., Matthews K., and Tabernero L. The TriTryp Phosphatome analysis of the protein phosphatase catalytic domains. BMC Genome, 8:434, 2007.
5. Davidson S., Overton C., and Buneman P. Challenges in integrating biological data sources. J. Comput. Biol., 2(4):557–572, 1995.
6. Fernàndez-Lòpez M. and Gòmez-Pèrez A. Overview and analysis of methodologies for building ontologies. Knowl. Eng. Rev., 17 (2):129–156, 2002.
7. Goble C.A., Stevens R., Ng G., Bechhofer S., Paton N.W., Baker P., Peim M., and Brass A. Transparent access to multiple bioinformatics information sources. IBM Syst. J., Special issue on deep computing for the life sciences, 40(2):532–552, 2001.
8. Karp P., Riley M., Saier M., Paulsen I., Paley S., and Pellegrini-Toole A. The EcoCyc and metacyc databases. Nucleic Acids Res., 28:56–59, 2000.
9. Lord P.W., Stevens R., Brass A., and Goble C.A. Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. Bioinformatics, 19–(10):1275–1283, 2003.
10. Luciano J. PAX of mind for pathway researchers. Drug Discov. Today, 10:937–942, 2005.
11. Ringland G. and Duce D. Approaches to Knowledge Representation: An Introduction Knowledge-Based and Expert Systems Series. John Wiley, Chichester, 1988.
12. Rosse C. and Mejino J.L.V. A reference ontology for bioinformatics: the foundational model of anatomy. J. Biomed. Inform., 36:478–500, 2003.
13. Taylor C., Paton N., Lilley K., Binz P., Julian Jr R., Jones A., Zhu W., Apweiler R., Aebersold R., Deutsch E., Dunn M., Heck A., Leitner A., Macht M., Mann M., Martens L., Neubert T., Patterson S., Ping P., Seymour S., Souda P., Tsugita A., Vandekerckhove J., Vondriska T., Whitelegge J., Wilkins M., Xenarios I., Yates J. (3rd) and Hermjakob H. The minimum information about a proteomics experiment (MIAPE). Nat. Biotech., 25:887–893, 2007.
14. The Gene Ontology Consortium Gene Ontology: Tool for the Unification of Biology. Nat. Gene., 25:25–29, 2000.
15. Wang H., Azuaje F., Bodenreider O., and Dopazo J. Gene expression correlation and gene ontology-based similarity: an assessment of quantitative relationships. In Proc. IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology. 2004, pp. 25–31.
16. Whetzel P., Brinkman R., Causton H., Fan L., Field D., Fostel J., Fragaso G., Gray T., Heiskanen M., Hernandez-Boussard T., Morrison N., Parkinson H., Rocca-Serra P., Sansone S.A., Schober D., Smith B., Stevens R., Stoeckert C., Taylor C., White J., and Wood A. the FuGo working group development of FuGo: An ontology for functional genomics investigations. OMICS J. Integrat. Biol., 10:199–204, 2006.
17. Whetzel P.L., Parkinson H., Causton H.C., Fan L., Fostel J., Fragoso G., Game L., Heiskanen M., Morrison N., Rocca-Serra P., Sansone S.A., Taylor C., White J., and Stoeckert C.J. The mged ontology: a resource for semantics-based description of microarray experiments. Bioinformatics, 22(7):866–873, 2006.

Ontology

Tom Gruber
RealTravel, Emerald Hills, CA, USA

Synonyms

Computational ontology; Semantic data model; Ontological engineering

Definition

In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. In the context of database systems, ontology can be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals. Ontologies are typically specified in languages that allow abstraction away from data structures and implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schema are models of data at the "logical" or "physical" level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services. In the technology stack of the Semantic Web standards

[1], ontologies are called out as an explicit layer. There are now standard languages and a variety of commercial and open source tools for creating and working with ontologies.

Historical Background

The term "ontology" comes from the field of philosophy that is concerned with the study of being or existence. In philosophy, one can talk about an ontology as a theory of the nature of existence (e.g., Aristotle's ontology offers primitive categories, such as substance and quality, which were presumed to account for All That Is). In computer and information science, ontology is a technical term denoting an artifact that is *designed* for a purpose, which is to enable the modeling of knowledge about *some* domain, real or imagined.

The term had been adopted by early Artificial Intelligence (AI) researchers, who recognized the applicability of the work from mathematical logic [6] and argued that AI researchers could create new ontologies as computational models that enable certain kinds of automated reasoning [5]. In the 1980s the AI community came to use the term ontology to refer to both a theory of a modeled world (e.g., a Naïve Physics [5]) and a component of knowledge systems. Some researchers, drawing inspiration from philosophical ontologies, viewed computational ontology as a kind of applied philosophy [10].

In the early 1990s, an effort to create interoperability standards identified a technology stack that called out the ontology layer as a standard component of knowledge systems [8]. A widely cited web page and paper [3] associated with that effort is credited with a deliberate definition of ontology as a technical term in computer science. The paper defines ontology as an "explicit specification of a conceptualization," which is, in turn, "the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them." While the terms specification and conceptualization have caused much debate, the essential points of this definition of ontology are:

- An ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.
- The specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use.

One objection to this definition is that it is overly broad, allowing for a range of specifications from simple glossaries to logical theories couched in predicate calculus [9]. But this holds true for data models of any complexity; for example, a relational database of a single table and column is still an instance of the relational data model. Taking a more pragmatic view, one can say that ontology is a tool and product of engineering and thereby defined by its use. From this perspective, what matters is the use of ontologies to provide the representational machinery with which to instantiate domain models in knowledge bases, make queries to knowledge-based services, and represent the results of calling such services. For example, an API to a search service might offer no more than a textual glossary of terms with which to formulate queries, and this would act as an ontology. On the other hand, today's W3C Semantic Web standard suggests a specific formalism for encoding ontologies (OWL), in several variants that vary in expressive power [7]. This reflects the intent that an ontology is a specification of an abstract data model (the domain conceptualization) that is independent of its particular form.

Foundations

Ontology is discussed here in the applied context of software and database engineering, yet it has a theoretical grounding as well. An ontology specifies a vocabulary with which to make assertions, which may be inputs or outputs of knowledge agents (such as a software program). As an *interface specification*, the ontology provides a language for communicating with the agent. An agent supporting this interface is not required to use the terms of the ontology as an *internal encoding* of its knowledge. Nonetheless, the definitions and formal constraints of the ontology do put restrictions on what can be *meaningfully* stated in this language. In essence, committing to an ontology (e.g., supporting an interface using the ontology's vocabulary) requires that statements that are asserted on inputs and outputs be *logically consistent* with the definitions and constraints of the ontology [3]. This is analogous to the requirement that rows of a database table (or insert statements in SQL) must be consistent with integrity constraints, which are stated declaratively and independently of internal data formats.

Similarly, while an ontology must be formulated in *some* representation language, it is intended to be a semantic level specification – that is, it is independent of

data modeling strategy or implementation. For instance, a conventional database model may represent the identity of individuals using a primary key that assigns a unique identifier to each individual. However, the primary key identifier is an artifact of the modeling process and does not denote something in the domain. Ontologies are typically formulated in languages which are closer in expressive power to logical formalisms such as the predicate calculus. This allows the ontology designer to be able to state semantic constraints without forcing a particular encoding strategy. For example, in typical ontology formalisms one would be able to say that an individual was a member of class or has some attribute value without referring to any implementation patterns such as the use of primary key identifiers. Similarly, in an ontology one might represent constraints that hold across relations in a simple declaration (A is a subclass of B), which might be encoded as a join on foreign keys in the relational model.

Ontology engineering is concerned with making representational choices that capture the relevant distinctions of a domain at the highest level of abstraction while still being as clear as possible about the meanings of terms. As in other forms of data modeling, there is knowledge and skill required. The heritage of computational ontology in philosophical ontology is a rich body of theory about how to make ontological distinctions in a systematic and coherent manner. For example, many of the insights of "formal ontology" motivated by understanding "the real world" can be applied when building computational ontologies for worlds of data [4]. When ontologies are encoded in standard formalisms, it is also possible to reuse large, previously designed ontologies motivated by systematic accounts of human knowledge or language [11]. In this context, ontologies embody the results of academic research, and offer an operational method to put theory to practice in database systems.

Key Applications

Ontologies are part of the W3C standards stack for the Semantic Web, in which they are used to specify standard conceptual vocabularies in which to exchange data among systems, provide services for answering queries, publish reusable knowledge bases, and offer services to facilitate interoperability across multiple, heterogeneous systems and databases. The key role of ontologies with respect to database systems is to specify a data modeling representation at a level of abstraction above specific database designs (logical or physical), so that data can be exported, translated, queried, and unified across independently developed systems and services. Successful applications to date include database interoperability, cross database search, and the integration of web services.

Cross-references

- ▶ Data Model
- ▶ Data Modeling
- ▶ Knowledge Base
- ▶ Knowledge Engineering

Recommended Reading

1. Berners-Lee T., Hendler J., and Lassila O. The semantic web. Scientific American, May 2001.
2. Gruber T.R. A translation approach to portable ontology specifications. Knowl. Acquisition, 5(2):199–220, 1993.
3. Gruber T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum. Comput. Stud., 43 (5–6):907–928, 1995.
4. Guarino N. Formal ontology, conceptual analysis and knowledge representation. Int. J. Hum. Comput. Stud., 43(5–6):625–640, 1995.
5. Hayes P.J. The second naive physics manifesto. In Formal Theories of the Common-Sense World, Moore (eds.). Hobbs, Ablex, Norwood, MA, 1985.
6. McCarthy J. Circumscription – a form of non-monotonic reasoning. Artif. Intell., 5(13):27–39, 1980.
7. McGuinness D.L. and van Harmelen F. OWL web ontology language. W3C Recommendation, February 10, 2004. Available online at: http://www.w3.org/TR/owl-features/.
8. Neches R., Fikes R.E., Finin T., Gruber T.R., Patil R., Senator T., and Swartout W.R. Enabling technology for knowledge sharing. AI Mag., 12(3):16–36, 1991.
9. Smith B. and Welty C. Ontology – towards a new synthesis. In Proc. Int. Conf. on Formal Ontology in Information Systems, 2001.
10. Sowa J.F. Conceptual Structures: Information Processing in Mind and Machine, Addison Wesley, Reading, MA, 1984.
11. Standard Upper Ontology Working Group (SUO). IEEE P1600.1. Available online at: http://suo.ieee.org/.

Ontology Acquisition

▶ Ontology Elicitation

Ontology Argumentation

▶ Ontology Elicitation

Ontology Elicitation

Pieter De Leenheer
Vrije Universiteit Brussel, Collibra nv, Brussels, Belgium

Synonyms

Ontology acquisition; Ontology learning; Ontology argumentation; Ontology negotiation; Knowledge creation

Definition

Ontology elicitation embraces the family of methods and techniques to explicate, negotiate, and ultimately agree on a partial account of the structure and semantics of a particular domain, as well as on the symbols used to represent and apply this semantics unambiguously.

Ontology elicitation only results in a *partial* account because the formal definition of an ontology cannot completely specify the intended structure and semantics of each concept in the domain, but at best can approximate it. Therefore, the key for scalability is to reach the appropriate amount of consensus on relevant ontological definitions through an effective meaning negotiation in an efficient manner.

Historical Background

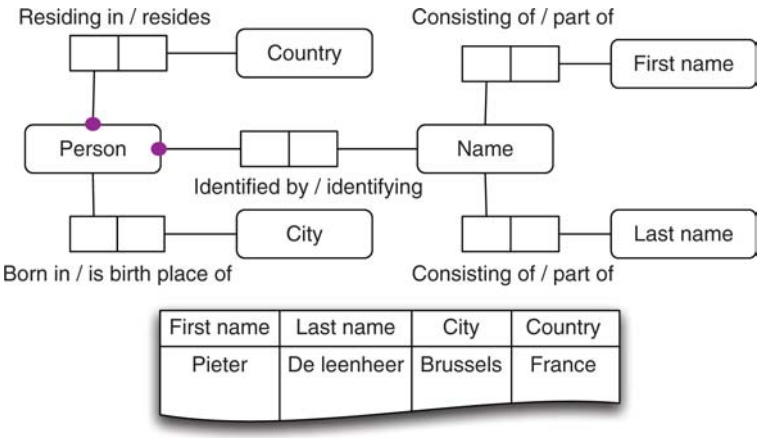
Ontology elicitation is based on techniques of *knowledge acquisition*, a subfield of AI that is concerned with eliciting and representing knowledge of human experts so that it can later be used in some application. Two typical knowledge acquisition methods can be distinguished:

1. *Top-down* (deductive) *knowledge elicitation* techniques are used to acquire knowledge directly from human domain experts. Examples include interviewing, case study, and mind mapping techniques.
2. *Bottom-up* (inductive) *machine learning* techniques use different methods to infer knowledge (e.g., concepts and rules) patterns from sets of data. A well-known example is *formal concept analysis* [10].

More formal methods for top-down knowledge acquisition use *knowledge modeling* as a way of structuring projects, acquiring and validating and storing knowledge for future use. Knowledge models include: symbolic character-based languages (e.g., logic, OWL), diagrammatic representations (networks, ladders, taxonomies, concept maps), tabular representations (e.g., matrices), structured text (e.g., hypertext) [16], and conceptual modeling.

Conceptual Modeling

Certain methods and techniques from the database field for *conceptual modeling* (e.g., ER, UML, dataflow diagrams) have been proven useful for ontology elicitation. For example, in [13], ORM/NIAM has been adopted. Figure 1 shows an example of a minimal ORM diagram (on top) explicating the semantics that is implicit in the relational table schema and population (on the bottom). For example, this ORM diagram already reveals what the table cannot, the semantics of



Ontology Elicitation. Figure 1. Illustration of a minimal ORM diagram explicating the implicit semantics for a relational table.

the relation of attribute "person" to attributes "city" and "country," and that "first name" and "last name" are both part of a "name." Furthermore, "city" and "country" appear not to be related at all.

A key characteristic of NIAM/ORM is that the analysis of information is based on natural language. This brings the advantage that the analysis can be done by the domain experts using their own vocabulary, and hence avoiding invalid interpretations. Furthermore, this attribute-free approach seen in the NIAM/ORM approach promotes semantic stability.

Data Schema Versus Ontology

Data models, such as data or XML schemas, typically specify the structure and integrity of data sets. Hence, building data schemas for an enterprise usually depends on the specific needs and tasks that have to be performed within this enterprise. *Data engineering* languages such as SQL aim to maintain the integrity of data sets and only use a typical set of language constructs to that aim, e.g., foreign keys. The schema vocabulary is basically to be understood intuitively (via the terms used) by the human database designer(s). The semantics of data schemas often constitute an informal agreement between the developers and an intended group of users of the data schema, and finds its way only in application programs that use the data schema instead of manifesting itself as an agreement that is shared amongst the community [20]. When new functional requirements pop up, the schema is updated on the fly. One designated individual usually controls this schema update process.

In (collaborative) ontology elicitation, however, absolute meaning is essential for all practical purposes, hence all elements in an ontology must ultimately be the result of agreements among human agents such as designers, domain experts, and users. In practice, correct and unambiguous reference to concepts or entities in the schema vocabulary is a real problem; often harder than agreeing about their properties, and obviously not solved by assigning system-owned identifiers.

Foundations

In collaborative ontology elicitation, multiple stakeholders have overlapping or contradicting perspectives about the intended structure, semantics, and vocabulary of the domain concepts [5]. This is principally caused by three facts: (i) no matter how expressive ontologies might be, they are all in fact lexical representations of concepts, relationships, and semantic constraints; (ii) linguistically, there is no bijective mapping between a concept and its lexical representation; and (iii) concepts can have different properties and values in different contexts of use. Hence, humans play an important role in the interpretation and negotiation of meaning during the elicitation and application of ontologies [7]. These principles can be illustrated by considering Stamper's *semiotic ladder* [21] that consists of six views or levels on signs from the perspective of physics, empirics, syntactics, semantics, pragmatics and the social world, that together form a complex conceptual structure. In this article, we only consider syntactical or *lexical* level, *semantic* level, and *pragmatic* level (Fig. 2). Ontology elicitation can be considered as a process that gradually takes ontological elements through these levels.

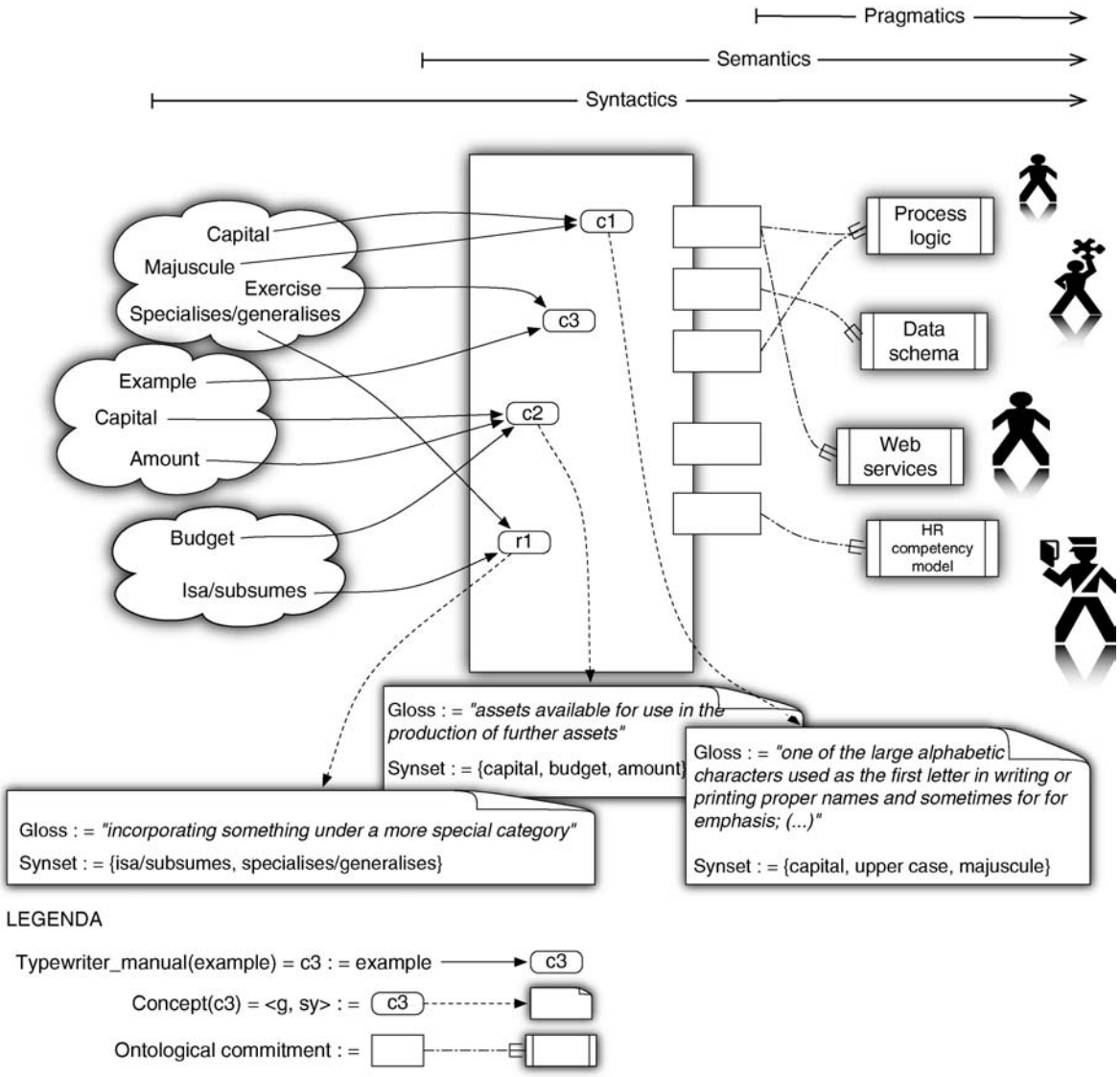
Lexical Versus Semantic Level

At the start of the elicitation of an ontology, its basic terminology for labeling concepts and relationships are extracted from various resources such as a text corpus [3], existing schemas [16], from so-called *serious games* [19] or rashly formulated by human domain experts through, e.g., tagging [22]. Many ontology engineering approaches focus merely on the conceptual modeling task, hence the distinction between lexical level (term for a concept) and semantic level (the concept itself) is often weak or ignored. In order to represent concepts and relationships lexically, they usually are given a uniquely identifying term (or label). However, the meaning of a concept behind a lexical term is influenced by the *elicitation context*, which is the context of the resource the term was extracted from. When eliciting and unifying information from multiple sources, this can easily give rise to misunderstandings and ambiguities, therefore the meaning of all terms used for ontology representation purposes should be articulated appropriately.

This is illustrated in Fig. 2: the full arrows denote the *meaning articulation* mappings between terms in organizational vocabularies (cloud on the left) and unique *concept identifiers* (e.g., c_1, r_1, etc.). The mapping of each unique concept identifiers to a particular explication of a meaning, i.e., a *concept definition* is defined by the dashed arrows.

Lexical Variability and Reusability

Even within one conversation, it turned out that in a less than a quarter of the cases, two individuals use



Ontology Elicitation. Figure 2. Three levels of ontology elicitation: lexical level, semantic level, and pragmatic level.

the same symbolic reference for a concept, and hence the freedom to use synonyms should be accommodated [8]. To engender creativity, domain experts should initially be allowed to use their own vocabularies, instead of being harshly restricted by an unfamiliar controlled taxonomy dictated by a central authorship. Gradually, this variability will converge towards one or more vocabularies that are commonly accepted.

For example, thousands of shared vocabularies or so-called folksonomies emerge, are sold and advertised, prosper or wither in a self-organizing manner on Web 2.0, through reuse and adaptation of natural language labels for tagging their resources. Natural language labels for concepts and relationships bring along their inherent ambiguity and variability in interpretation. Folksonomies provide on the one hand an unbounded reusability potential for specific reference in a given application context, which is important for scalable ontology elicitation. On the other hand, however, an analysis of multiple contexts is generally needed to disambiguate successfully [1,5].

Semantic Versus Pragmatic Level

The meaning articulation mappings and the concept definition service respectively provide unambiguous

reference and semantic explication of terms, independent of the preferred vocabulary. However, the meaning of these concepts should be further formalized for appropriately serving application purposes, by combining and linking them with other concepts, and axiomatizing them with semantic constraints and rules. In the section on applications, there is an overview of typical applications, including process logic and (legacy) information system interoperability, web service orchestration, and competency model gap analysis in human resources (HR).

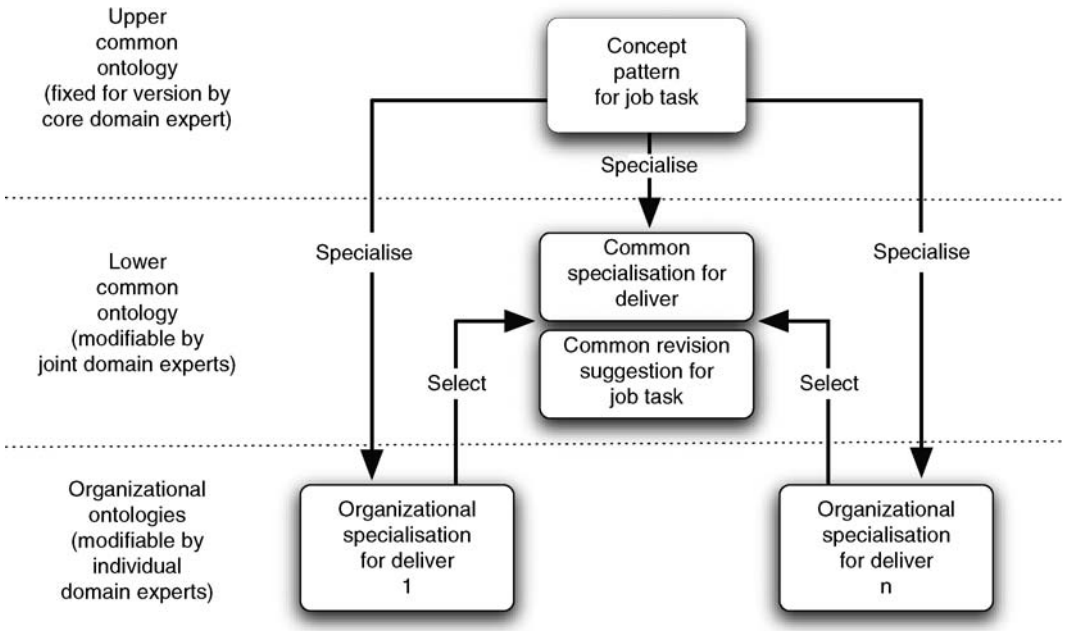
The relevant properties and values for the concepts to be agreed on, depend on the application requirements. For the sake of scalability, as in any realistic system or knowledge engineering scenario, in ontology elicitation, (parts of) existing semantic resources are reused and adopted as much as possible for new application purposes. This asks for a methodological trade-off between the reuse of relevant consensus from existing application contexts as much as possible, while allowing specific variations for new application requirements at stake to be collaboratively negotiated, based on (parts of) existing consensus.

Figure 3 illustrates this with a model for collaborative ontology elicitation, as introduced by [7], and inspired by the Delphi method (http://en.wikipedia.org/wiki/Delphi_method.). For bootstrapping the elicitation of a concept, knowledge workers are given a *pattern* that defines the current insights and interests of the community for that type of concepts. The example here concerns the elicitation of a concept "Deliver," which is a type of "Job Task." Each of the stakeholding organizations elicit the relevant properties and values of "Deliver" by specializing the pattern abstracted from "Job Task." If no pattern exists, it is bootstrapped by the core domain experts overlooking the domain.

Divergence and Conflict

Divergence is the point in collaborative ontology elicitation where domain experts disagree or have a conflict about the meaning of some concept in such a way that consequently their ontologies evolve in widely varying directions. Although they share common goals for doing business, divergent knowledge positions appear as a natural consequence when people collaborate in order to come to a unique common understanding.

Rather than considering it to be a problem, conflicts should be seen as an opportunity to *negotiate* about the subtle differences in interpretation of the domain, which will ultimately converge to a shared understanding disposed of any subjectivity. However, meaning conflicts and ambiguities should only be resolved when relevant. It is possible that people have



Ontology Elicitation. Figure 3. A model for collaborative ontology elicitation.

alternative conceptualizations in mind for business or knowledge they do not wish to share. Therefore, in building the shared ontology, the individual ontologies of the various partners only need to be aligned insofar necessary, in order to avoid wasting valuable modeling time and effort.

Basically they only need to agree on a common specialization of the current properties present in the pattern. However, it could be the case that a considerable part of the stakeholders identifies new relevant properties, or see other properties to be obsolescent. This provides a feedback suggestion to revise the patterns for a next version of the ontology.

Relevant techniques for collaborative ontology negotiation and argumentation include [14,17].

Convergence and Patterns

Once, a common specialization is agreed on, it is lifted up in the upper common levels of the ontology. Gradually, this would result in the emergence of increasingly stable generally deployable ontology patterns that are key for enabling future business interoperability needs in a scalable manner [2,6,9,7].

Key Applications

Ontologies have become an integral part of many academic and industrial applications in various domains, including Semantic Web services, regulatory compliance, and human resources.

Semantic Web Services

Service-oriented (SOA) is an architecture that relies on *service-orientation* as its fundamental design principle. In a SOA environment, independent services can be accessed without knowledge of their underlying platform implementation. Within this paradigm, the creation of automation logic is specified in the form of services. Service orientation is another design paradigm that provides a means for achieving a separation of concerns, which obviously increases the potential for software reusability.

The Semantic Web aims to make data accessible and understandable to intelligent machine processing. Semantic Web services additionally aim to do the same for services available on the Semantic Web, targeting automation of service discovery, composition and invocation. For describing Semantic Web services, it is required to elicit the so-called "domain ontologies" or that formalize the knowledge necessary for capturing the meaning of services and exchanged data. In other words, given a particular business goal, the domain ontologies enable the weaving of the relevant concerns that are separated in relevant services.

A key challenge here is to overcome the *ontology-perspicuity bottleneck* [11] that constrains the use of ontologies, by finding a compromise between top-down imposed formal semantics expressed in expert language and bottom-up emerging real-world semantics expressed in layman user language.

For more on infrastructure, theory, business aspects, and experiences on ontology elicitation and management for Semantic Web applications, see [12].

Regulatory Compliance

Businesses and government must be able to show compliance of their outputs, and often also of their systems and processes, to specific regulations. Demonstrable evidence of this compliance is increasingly an auditable consideration and required in many instances to meet acceptable criteria for good corporate governance. Moreover the number and the complexity of applicable regulations in Europe and elsewhere is increasing. This includes mandatory compliance audits and assessments against numerous regulations and best practice guidelines over many disciplines and against many specific criteria. The implementation of information communications technology also means that previous manual business processes are now being performed electronically and the degree of compliance to applicable regulations depends on how the systems have been designed, implemented and maintained. Keeping up with the rate of new regulations for a major corporation and small business alike is a never ending task. What is the answer to all of this regulatory complexity? First one should simplify regulations where possible and then apply automatic tools to assist.

The automated data demands of networked economies and an increasingly holistic view on regulatory issues are driving and yet partially frustrating attempts to simplify regulations and statutes. In an ideal world companies and other organizations would have the tools and online services to check and measure their regulatory compliance; and governmental organizations would be able to electronically monitor the results. This requires a more systemic shared approach to regulatory assurance assessment and compliance certification.

Lessig [15] has a simple yet profound thesis "Code is law." The application of this concept taken in conjunction with the emergence of regulatory ontologies opens up a new way of assessing whether burgeoning

systems are compliant with regulations they seek and claim to embody. First specific regulations (e.g., data privacy, digital rights management) are converted into and expressed as "Regulatory Ontologies." These ontologies are then used as the base platform for a "Trusted Regulatory Compliance Certification Service."

Over time the resulting ontology describing and managing the areas analyzed can literally replace the regulations and compliance criteria. So much so, it is envisaged that an eventual outcome could be that the formal writing (codification) of future laws will start with the derived ontologies and use intelligent agents to help propose specific legal text which ensures that the policy objectives are correctly coded in law. In addition automatic generation of networked computer applications that are perfectly compliant with the wide variety of directives and laws in any country is one of the ultimate goals of this type of ontology based work.

For an overview of ontology-grounded trusted regulatory compliance, see [18].

Human Resources

Competencies describe the skills and knowledge individuals should have in order to be fit for particular jobs. Especially in the domain of vocational education, having a central shared and commonly used competency model is becoming crucial in order to achieve the necessary level of interoperability and exchange of information, and in order to integrate and align the existing information systems of competency stakeholders like schools or public employment agencies. Only few organizations however, have successfully implemented a company-wide "competency initiative," let alone a strategy for inter-organizational exchange of competency related information.

Several projects (See, e.g., the EU-funded CoDrive project.) aim at contributing to a competency-driven vocational education by using state-of-the-art ontology methodology and infrastructure in order to collaboratively develop a conceptual, shared and formal KR of competence domains.

For a business case study on vocational competency ontology elicitation, see [4].

Future Directions

The ever-changing interoperability requirements between the stakeholding communication partners (See, e.g., diverse (legacy) systems in the open extended enterprise.) requires ontologies to continuously evolve. Usually the domain is too large and complex to be explicated in one single effort, and the knowledge workers understanding of the domain is in continuously changing, requiring timely renegotiation of existing consensus. Therefore, one should not merely focus on the practice of eliciting ontologies in a project-like context, but consider it as a real-time collaborative and continuous process that is integrated with and in the operational processes of the community itself. The shared background of communication partners is continuously negotiated as are the characteristics or values of the concepts that are agreed upon.

There are many additional complexities that should be considered. As investigated in FP6 integrated projects (See, e.g., http://ecolead.vtt.fi/.) on collaborative networked organizations, the different professional, social, and cultural backgrounds among communities and organizations can lead to misconceptions, resulting in costly ambiguities and misunderstandings if not aligned properly. This is especially the case in inter-organizational settings, where there may be many pre-existing organizational sub-ontologies, inflexible data schemas interfacing to legacy data, and ill-defined, rapidly evolving collaborative requirements. Furthermore, participating stakeholders usually have strong individual interests, inherent business rules, and work practices. These may be tacit, or externalized in workflows that are strongly interdependent, hence further complicate the conceptual alignment. Finally this also involves ontology elicitation cost estimation. Simperl and Sure (chapter 7, [12]) propose a parametric cost estimation model for ontologies by identifying relevant cost drivers having a direct impact on the effort invested in ontology elicitation.

For an overview of future directions towards community-driven ontology elicitation and management, see [6].

Cross-references

- ► Emergent Semantics
- ► Ontology
- ► Ontology Engineering

Recommended Reading

1. Bachimont B., Troncy R., and Isaac A. Semantic commitment for designing ontologies: a proposal. In Proc. 13th Int. Conf. on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 2002, pp. 114–121.
2. Blomqvist E. OntoCase – a pattern-based ontology construction approach. In Proc. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS, 2007, pp. 971–988.

3. Buitelaar P., Cimiano P., and Magnini B. Ontology learning from text: methods, evaluation and applications, vol. 123 of Frontiers in Artificial Intelligence and Applications, IOS, Amsterdam, 2005.
4. Christiaens S., De Leenheer P., and de Moor A. Robert Meersman R. Ontologising Competencies in an Interorganisational Setting. In Ontology Management. vol. 7 of Semantic Web and Beyond Computing for Human Experience, Springer, Berlin, 2008, pp. 265–288.
5. De Leenheer P., de Moor A., and Meersman R. Context dependency management in ontology engineering: a formal approach. J. Data Semantics, 8:26–56, 2006.
6. De Leenheer P. and Meersman R. Towards community-based evolution of knowledge-intensive systems. In Proc. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS, 2007, pp. 989–1006.
7. de Moor A., De Leenheer P., and Meersman R. DOGMA-MESS: a meaning evolution support system for interorganizational ontology engineering. In Proc. 14th Int. Conf. on Conceptual Structures, 2006, pp. 189–203.
8. Furnas G., Landauer T., and Dumais S. The vocabulary problem in human-system communication. Commun. ACM, 30(11):964–971, 1987.
9. Gangemi A. Ontology design patterns for semantic web content. In Proc. 4th Int. Semantic Web Conf., 2005, pp. 262–276.
10. Ganter B., Stumme G., and Wille R. (eds.), Formal concept analysis, foundations and applications, LNCS, vol. 3626, Springer, Berlin, 2005.
11. Hepp M. Possible ontologies: how reality constrains the development of relevant ontologies. IEEE Internet Comput., 11(1):90–96, 2007.
12. Hepp M., De Leenheer P., de Moor A., and Sure Y. (eds.) Ontology management, semantic web, semantic web services, and business applications, vol. 7 of Semantic Web and Beyond Computing for Human Experience. Springer, Berlin, 2008.
13. Jarrar M., Demey J., and Meersman R. On reusing conceptual data modeling for ontology engineering. J. Data Semantics, 1(1):185–207, 2003.
14. Kotis K. and Vouros G. Human-centered ontology engineering: the Hcome methodology. Knowl. Inf. Syst., 10:109–131, 2005.
15. Lessig L. Ontology Management, Semantic Web, Semantic Web Services, and Business Applications. Basic Books, 1999.
16. Milton N. Knowledge Acquisition in Practice: A Step-by-Step Guide. Springer, London, 2007.
17. Pinto H., Staab S., and Tempich C. DILIGENT: towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In Proc. 16th European Conf. on Artificial Intelligence, 2004.
18. Ryan H., Spyns P., De Leenheer P., and Leary R. Ontology-based platform for trusted regulatory compliance services. In OTM Workshops, LNCS, vol. 2889, Springer, Berlin, 2003, pp. 675–689.
19. Siorpaes K. and Hepp M. Games with a purpose for the semantic web. IEEE Intell. Syst., 23(3):50–60, 2008.
20. Spyns P., Meersman R., and Jarrar M. Data modelling versus ontology engineering. ACM SIGMOD Rec., 31(4): 12–17, 2002.
21. Stamper R. Information in Business and Administrative Systems. Wiley, NY, 1973.
22. Van Damme C., Hepp M., and Siorpaes K. Folksontology: an integrated approach for turning folksonomies into ontologies. In Proc. ESWC Workshop Bridging the Gap between Semantic Web and Web 2.0, 2007.

Ontology Engineering

Avigdor Gal
Technion – Israel Institute of Technology, Technion City, Haifa, Israel

Synonyms

Ontological engineering

Definition

Ontology Engineering is "the set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies" [2]. It provides "a basis of building models of all things in which computer science is interested" [4]. Ontology engineering aims at providing standard components for building knowledge models. Ontologies play a similar role to design rationale in mechanical design. It allows the reuse of knowledge in a knowledge base by providing conceptualization, reflecting assumptions and requirements made in the problem solving using the knowledge base. Ontology engineering provides the means to build and use ontologies for building models.

Key Points

Eight levels (from shallow to deep) of using ontologies can be defined [4]. At level 1, ontologies are used as a common vocabulary for communication. At level 2, it is used as a conceptual schema of a relational data base. At the third level, ontologies are used as backbone information for using a knowledge base. The remaining five levels are the levels where ontology engineering comes into play. Ontologies at the fourth level are used to answer competence questions and then they are used for standardization (of terminology or of tasks) at level 5. At level 6, ontologies are used for structural and semantic transformation of schemas. Reusing knowledge is done at the seventh level and knowledge reorganization is considered the eighth and highest level of using ontologies.

Ontology engineering makes use of ontologies (in the sense of levels 4–8) to generate standard tools for knowledge representation. This does not imply that knowledge is standardized. Using ontology engineering, one can design knowledge for specific applications, similar to production based on engineering tools in other engineering fields.

The use of ontology engineering is now illustrated in two key applications, namely functional design and schema matching. For the former, [3] describes a seventh level of using ontologies in a real world application of plant and production systems. There, an ontology that describes two types of functional models, two types of organization of generic knowledge, and two ontologies of functionality were put into use in sharing functional design knowledge on production systems. The users (engineers) of the system have indicated that this framework enabled them to make implicit knowledge possessed by each designer explicit, and to share it among team members.

Ontologies are used in schema matching in many ways. One of these, corresponding to the sixth level of use, was presented in the OntoBuilder toolcase [1]. Special ontological constructs were identified for the matching of Web form data. Ontologies were built using these constructs and dedicated matching algorithms were constructed to determine the amount of certainty to assign with the matching of attribute pairs. An example of an ontological construct, unique to OntoBuilder, is *precedence*. This construct determines the order in which attributes are presented to the user on a Web form and generate a partial order on attributes. Attribute similarity is then measured based on their relative positioning in their own ontologies.

Cross-references

► Ontology
► Semantic Matching

Recommended Reading

1. Gal A., Modica G., Jamil H., and Eyal A. Automatic ontology matching using application semantics. AI Mag., 26(1):21–32, 2005.
2. G'omez-P'erez A., Fern'andez-L'opez M., and Corcho O. Ontological Engineering. Springer, Berlin, 2003.
3. Kitamura Y., Kashiwase M., Fuse M., and Mizoguchi R. Deployment of an ontological framework of functional design knowledge. Adv. Eng. Inform., 18(2):115–127, 2004.
4. Mizoguchi R. and Ikeda M. Towards Ontology Engineering. Technical Report AI-TR-96-1, I.S.I.R., Osaka University, 1996.
5. Paslaru Bontas E. and Tempich C. Ontology Engineering: A Reality Check. In Proc. 5th Int. Conf. on Ontologies, DataBases, and Applications of Semantics, 2006, pp. 836–854.
6. Sure Y., Tempich C., and Vrandecic D. Ontology engineering methodologies. In John Davies, Rudi Studir and Paul Warren (Eds). Semantic Web Technologies: Trends and Research in Ontology-Based Systems. Wiley, UK, 2006.

Ontology Learning

► Ontology Elicitation

Ontology Negotiation

► Ontology Elicitation

Ontology Query Languages

► Semantic Web Query Languages

Ontology Visual Querying

Sean Bechhofer, Norman W. Paton
University of Manchester, Manchester, UK

Definition

An *ontology definition language* provides constructs that can be used to describe concepts and the relationships in which they participate. Because such languages define the properties concepts can exhibit, they can be used to restrict the questions that can meaningfully be asked about the concepts. Given a specification of the questions that can legitimately be asked, a user interface can direct query construction tasks towards meaningful requests, which in turn are expected to yield non-empty answers. Thus *ontology visual querying* is the use of an ontology to direct interactive query construction. A related topic is *faceted browsing*, in which the incremental description of concepts of interest is closely integrated with retrieval, thereby providing information about the results of a request as it is being constructed.

Historical Background

The history of visual query languages is almost as long as that of textual query languages, with Query-by-Example [14] developed in parallel with SQL. Query-by-Example contained two features that recur in almost all visual query languages: (i) a representation of the model over which the query is to be expressed; and (ii) a notation for incrementally constructing queries from the collections, relationships and value ranges of interest, with reference to the representation in (i). The evolution of visual query languages [4] has tracked the evolution of data models and interactive paradigms, and proposals have been made that support querying over many different data models (relational, object-oriented, temporal, etc) using a variety of interaction objects (forms, graphs, icons, etc). An orthogonal aspect is the closeness of the relationship between query construction and answer presentation; for example, in *dynamic queries* the answer to a request is constructed automatically and incrementally as a query is refined [12]. This provides immediate feedback to users on the size and nature of the result, but may require specialised storage structures to support incremental result computation.

Ontology visual querying has been developed so that the knowledge expressed in an ontology can be used to direct query construction; query answers may then be constructed from an instance store that is closely integrated with the ontology definition language, or by evaluating requests over external data sources. The latter is quite common, as ontologies are widely used to provide conceptual models for web (e.g., [1]) or data (e.g., [3]) resources.

Foundations

As in other visual query languages, ontology visual querying requires a visual representation of the concepts over which a request is to be constructed. Visual query formulation allows users to explore the domain of interest by *recognition* rather than *recall*: that is, it should not be necessary to remember (or even be fully aware of) the ontology in order to express a query over it.

Ontologies are represented visually other than for querying; for example, ontology design tools typically support both form and graph-based views of concepts and their relationships (e.g., [10]). As concept definition and query formulation may have significant common ground, representations that are useful for ontology browsing and concept definition may also be relevant for querying. For example, in the TAMBIS ontology-based data integration system [2], a query is a concept definition in a Description Logic (DL) [8], so writing a query is essentially the same as defining a new concept.

Although expressive ontology languages may present challenges for navigation and thus query construction, they also present certain opportunities for query interface designers, as various forms of reasoning may be useful for guiding query construction. For example, an interface can prevent the submission of queries that are unsatisfiable (i.e., that are known from the definitions in the ontology to return no results) either by making it impossible for the user to construct such queries or by detecting when such requests have been created (e.g., [7,5]). Such feedback can be seen as *intensional*, with the constraints or knowledge in the ontology determining the behaviour of the interface. Feedback may also be *extensional*, for example, with the number of results to be returned being shown to the user. This is common in facted browsing systems, and such direct result construction is generally supported in combination with closely integrated stores.

In addition to the feedback described above, systems may also provide alternative renderings of the query being constructed – for example a natural language description of the query. This can be of use in helping naïve or inexperienced users in forming appropriate queries. Note that this involves the rendering of the query in natural language, rather than translating a query posed using natural language.

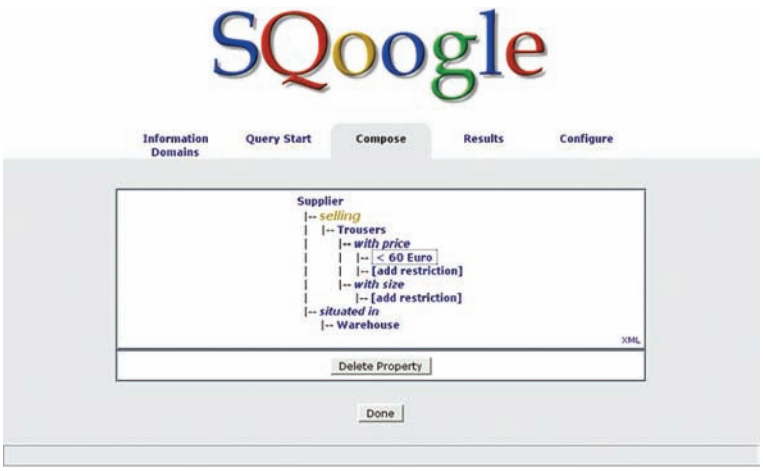
A common approach is to specify queries through an iterative refinement process, in which the content of the ontology and current context of the query impact on the options presented. The principle of intensional navigation uses the vocabulary to guide the user during query formulation, employing constraints in the ontology to either flag to the user that the query is in some way violating the constraints, or preventing the user from forming queries that would be unsatisfiable, and thus return no results. Operations available for query manipulation in SEWASIE [5] include the addition of a new role/property with an associated filler, or the replacement of a filler value. In the latter case, a classification or super/sub class taxonomy is used to support the manipulation, with value fillers being specialised or generalised. Although ontology languages differ in the constructors and expressivity

offered, some notion of hierarchical classification is nearly always present, and so can be exploited in visual query interfaces. An additional operation offered by the TAMBIS system is to *refocus* the query, which takes a sub node of the query and reorganises the query to promote that node to the root. This operation introduces an additional requirement on the ontology language, namely that properties or relations have *inverses*.

Various of the notions discussed above are illustrated in Figs. 1 and 2 for SQoogle, which was developed in the SEWASIE project. Figure 1 shows the composition phase, with the graphical depiction of the query. In Fig. 2, the user is being offered generalisations or specialisations of a particular node in the query.

Overall, ontology visual query systems can be characterized by a number of features:

- *Identification of starting points.* The construction of a query has to start from some place in the ontology; systems may offer predetermined entry points, user defined bookmarks, or a search mechanism across the concepts in the ontology.
- *Query language.* More expressive query languages support more precise question answering, but may contain constructs that require explanation for



Ontology Visual Querying. Figure 1. Visual query expression in SQoogle. The query is to select suppliers that sell trousers that cost less than 60 euros, where the supplier is situated in a warehouse.

Ontology Visual Querying. Figure 2. Visual query refinement in SQoogle. The query can be revised by replacing the concept supplier with a more general (e.g., agent, broker) or specialized (e.g., wholesaler) concept.

users or that are challenging to represent using certain visual paradigms. Ontology visual query languages rarely support features such as aggregation or grouping.

- *Query modification operations.* Query construction involves manipulation of query expressions, for example, to include additional relationships or to specialise a concept named in the query.
- *The ontology definition language.* Richer ontology definition languages are generally more complex to display and navigate, but may provide more options for generalizing and specializing query components, and can express constraints that are useful for directing query construction.
- *Relationship between query language and ontology language.* Proposals implement different relationships between the query language and the ontology definition language. For example, in SEWASIE, the query language and ontology language are explicitly separated. The ontology language supports reasoning services that are used to guide the intensional navigation process described above. In contrast, in TAMBIS, queries *are* concept descriptions, thus the interface is tied more closely to the ontology language.
- *Feedback mechanisms.* Feedback can inform the user about the results of the query or the state of the query with respect to the underlying ontology. Feedback may be intensional (in terms of the ontology), or extensional (in terms of the result set).
- *Query presentation.* Queries may be presented solely using the visual query, or may also offer, for example, natural language renderings of the query.
- *Domain specificity.* Visual interfaces provide the opportunity to represent models or results using general-purpose or domain-specific representations. Most ontology visual query languages are general purpose, but faceted browsing interfaces are often designed to support specific applications.

Table 1 describes a collection of representative visual query systems using a selection of the above criteria: the TAMBIS and SEWASI visual query languages, and the Flamenco and /facet faceted browsing systems. In all these proposals, the ontology directs query construction, and thus the design of the ontology has a significant influence on the utility of the interface.

Key Applications

Ontology visual query systems have most commonly been deployed in areas of science and culture where there are rich data resources to be explored. For example, TAMBIS provided access to multiple biological information sources. Faceted browsing and querying has been widely used to browse image collections and in the cultural heritage domain, for example to support access to Finnish Museums [9] and galleries in the Netherlands [11]. The faceted approach is also common in on-line shopping sites such as eBay.

Ontology Visual Querying. Table 1. Representative examples of ontology visual query systems

Proposal	TAMBIS	SEWASIE	Flamenco	facet
Reference	[7]	[5]	[13]	[8]
Query language	Concept definition	Conjunctive queries	Path expressions	Path expressions
Query modification operations	Property add/remove; filler specialization or generalization; refocus	Property add/remove; filler specialization or generalization	Property add/remove; filler specialization or generalization	Property add/remove; filler specialization or generalization
Ontology definition language	Description logic	Description logic	Hierarchical categories	RDFS
Feedback mechanism	Intensional	Intensional	Extensional	Extensional
Query presentation	Visual	Visual plus NL rendering	Path expression	Path expression
Domain specificity	Generic	Generic	Specific (image repositories)	Generic

Future Directions

Large amounts of a data are beginning to emerge using representation languages like OWL. Current work in ontology languages should see standardisation of the SPARQL query language finalised in the near future. Query interfaces that sit on top these standardized languages will then be required in order to support access to this data – interfaces that support naïve or non-expert users will clearly be required.

Cross-references

► OWL: Web Ontology Language
► Visual Query Language

Recommended Reading

1. Antoniou G. and van Harmelen F. A Semantic Web Primer. MIT Press, Cambridge, MA, 2004.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. (eds.). The Description Logic Handbook. Cambridge University Press, Cambridge, 2003.
3. Calvanese D., De Giacomo G., Lenzerini M., Nardi D. and Rosati, R. Data integration in data warehousing. Int. J. Cooperative Inf. Syst. 10(3):237–271, 2001.
4. Catarci T., Costabile M.F., Levialdi S., and Batín C. Visual query systems for databases: a survey. J. Vis. Lang. Comput. 8(2): 215–260, 1997.
5. Catarci T., Dongilli P., Di Mascio T., Franconi E., Santucci G., and Tessaris S. An ontology based visual tool for query formulation support. In Proc. 16th European Conf. on AI, 2004, pp. 308–312.
6. Colucci S., Noia T.D., Sciascio E.D., Donini F.M., Ragone A., and Rizzi R. A semantic-based fully visual application for matchmaking and query refinement in B2C e-marketplaces. In Proc. 8th ACM Int. Conf. on Electronic Commer. 2006, pp. 174–184.
7. Goble C.A., Stevens R., Ng G., Bechhofer S., Paton N.W., Baker P.G., Peim M., and Brass A. Transparent access to multiple bioinformatics information sources. IBM Syst. J., 40(2):532–551, 2001.
8. Hildebrand M., van Ossenbruggen J., and Hardman L. /facet: a browser for heterogeneous semantic web repositories. In Proc. 5th Int. Semantic Web Conf., 2006, pp. 272–285.
9. Hyvyonen E., Myakelya E., Salminen M., Valo A., Viljanen K., Saarela S., Junnila M., and Kettula S. Museum Finland – Finnish museums on the semantic web. J. Web Semantics 3(2):224–241, 2005.
10. Knublauch H., Fergerson R.W., Noy N.F., and Musen M.A. The protégé OWL plugin: an open development environment for semantic web applications. In Proc. 3rd Int. Semantic Web Conf., 2004, pp. 229–243.
11. Schreiber G., et al. MultimediaN E-culture Demonstrator. In Proc. 5th Int. Semantic Web Conf. 2006, pp. 951–958.
12. Shneiderman B. Dynamic queries for visual information seeking. IEEE Software 11(6):70–77, 1994.
13. Yee K.-P., Swearingen K., Li K., and Hearst M.A. Faceted metadata for image search and browsing. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2003, pp. 401–408.
14. Zloof M.M. Query-by-example: the invocation and definition of tables and forms. In Proc. 1st Int. Conf. on Very Large Data Bases. 1975, pp. 1–24.

On-Wire Security

► Storage Security

OODB (Object-Oriented Database)

► Object Data Models

Open Database Connectivity

Changqing Li
Duke University, Durham, NC, USA

Synonyms

ODBC

Definition

Open Database Connectivity (ODBC) [1] is an Application Programming Interface (API) specification to use database management systems (DBMS). The ODBC API is a library of functions for the ODBC-enabled applications to connect any ODBC-driver-available database, execute Structured Query Language (SQL) statements, and retrieve results. ODBC is independent of programming languages, database systems and operating systems.

Key Points

ODBC (pronounced as separate letters), is a standard database access method developed by the SQL Access Group in 1992. Its objective is to make any application to access any data regardless of the database management systems. To achieve this objective, ODBC inserts a database driver as a middle layer between an application and the DBMS, the purpose of which is to translate the application queries to commands understood by the DBMS. In practice, both the application and the DBMS must be ODBC-compliant; that is, the application must

be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

A procedural API is offered by the ODBC specification for using SQL queries to access data. One or more applications will be contained in an implementation of ODBC, a core ODBC library, and one or more "database drivers." Independent of the applications and DBMS, the core library acts as an "interpreter" between the applications and the database drivers, whereas the database drivers contain the DBMS-specific details. Thus applications can be written to use standard types and features without concerning the specifics of each DBMS that the applications may encounter. Similarly, database driver implementors only need to know how to attach to the core library. This makes ODBC modular.

ODBC operates with a variety of operating systems and drivers existing for relational database as well as non-relational data such as spreadsheets, text and XML files.

Cross-references

- ▶ Data Integration
- ▶ Database Adapter and Connector
- ▶ Interface
- ▶ Java Database Connectivity
- ▶ .NET Remoting
- ▶ Web 2.0/3.0
- ▶ Web Services

Recommended Reading

1. Geiger K. Inside ODBC. Microsoft Press, 1995.

Open Nested Transaction Models

Alejandro Buchmann
Darmstadt University of Technology, Darmstadt, Germany

Synonyms

Extended transaction models; advanced transaction models

Definition

Open nested transactions are hierarchically structured transactions with relaxed ACID properties. Individual subtransactions may commit independently before the complete top level transaction commits. Therefore, conventional rollback is not possible and the effects of a commited subtransaction have to be compensated if the top level transaction aborts. Depending on the particular open nested transaction model, subtransactions may be vital or non-vital and may have alternative or contingency subtransactions. Open nested transaction models are characterized through relaxed visibility rules, abort and commit dependencies.

Historical Background

Open nested transaction models evolved in the 1980s in response to two major sets of requirements: the needs of federated multidatabase systems integrating autonomous legacy database systems, and the demands of long running, cooperative processes for higher levels of concurrency while maintaining some of the main benefits of transactional processes. Extended transaction models influenced the tightly coupled transaction model of distributed object systems and the activity model of the OMG, as well as the transaction and coordination models of Web Services.

Foundations

Transaction models are characterized by the structure of its transactions, the commit and abort dependencies and the visibility rules among transactions.

Nested transactions are hierarchically structured transactions consisting of a top level transaction and subtransactions that may themselves be tree-structured. Typical of the execution model of nested transactions is the fact that higher level transactions do not execute while their subtransactions are active. The order of execution of subtransactions can be either sequential or parallel. The closed nested transaction model proposed by Moss does not specify an execution order and preserves the atomicity, consistency, isolation and durability properties of traditional flat transactions. The commit dependencies of closed nested transactions specify that all subtransactions must commit to their immediate ancestor and the top level transaction can only commit after all the subtransactions terminated. The abort dependencies specify that the whole transaction tree must be aborted if the top level transaction aborts while the abort of a subtransaction can be handled by the immediate ancestor transaction. The visibility rules among transactions and subtransactions specify that subtransactions may see changes of other subtransactions only once they are commited to the common ancestor. Changes of

a nested transaction become visible to the outside world only after the top level transaction commits. Since changes of committed subtransactions are made visible only after the top level transaction commits, they can be undone through conventional roll-back.

Open nested transaction models are also based on hierarchically structured transactions. However, depending on the particular transaction model, the subtransactions may be of different types. Different types of subtransactions require different commit and/or abort dependencies. The visibility rules are also relaxed with respect to those of closed nested transaction models.

Open nested transactions may consist of the following component transactions:

- One top level transaction that has mostly a coordination function
- Vital subtransactions that must all commit for the top level transaction to be allowed to commit
- Non-vital subtransactions of which one or more may abort without causing the top level transaction to fail
- Contingency transactions are alternative subtransactions that often are executed by different autonomous systems or service providers
- Compensating transactions that must be defined to undo the changes of subtransactions that may have committed but must be undone
- Triggers that are executed as subtransactions and may execute either immediately, deferred before the commit of the triggering transaction or as detached transactions.

Open nested transaction models were defined for long running transactions and for transactions executing on federated autonomous (legacy) systems. Therefore, the degree of control of the coordinating top level transaction over the execution of the subtransactions is reduced compared to closed nested transactions running on a single database management system. Distributed short lived transactions and closed nested transactions are typically implemented through a two-phase commit protocol. In a two-phase commit protocol the first phase serves to reach agreement among the participants whether to commit or to abort, and once consensus to commit has been reached, the commit is carried out in the second phase. This protocol requires holding all the resources required by all the subtransactions through the negotiation phase until the global commit. This approach is not feasible for long running transactions because of performance reasons and in federated multidatabase systems because of the autonomy of the participating database systems.

The coordinating top level transaction in an open nested transaction model cannot secure the resources of the participating systems that execute subtransactions as autonomous individual short transactions until all participating systems have executed their subtransactions and are ready to commit. Therefore, subtransactions may commit immediately upon completion and their results are thus visible to the outside world before the top level transaction commits. Compensating transactions execute the semantically inverse operation of the committed subtransaction but do not guarantee that the exact initial state can be restored since other transactions may have executed in the interim.

The commit dependencies of open nested transaction models depend on whether they distinguish between vital and non-vital subtransactions or not. A commit dependency exists between the top level transaction and all vital subtransactions, i.e., the coordinating top level transaction may only commit if all the vital subtransactions committed. This is the default case if non-vital subtransactions are not provided. The abort of a non-vital transaction does not affect the possibility to commit the top level transaction.

The abort dependencies of open nested transaction models are the same as for closed nested transactions: the abort of the parent transaction causes the abort and undo (roll-back or compensation) of the subtransaction. The abort of a non-vital subtransaction is handled by the parent transaction, the abort of a vital transaction causes the abort of the parent and all siblings.

Contingency transactions represent alternative actions. For example, if the top-level transaction represents the booking of a trip consisting of a roundtrip flight, a hotel and a rental car booking, two different flights on different airlines may be defined as a subtransaction and a contingency transaction. Contingency transactions may only commit if the primary subtransaction aborts. Open nested transaction models that do not provide contingency transactions must implement this functionality as application-specific code in the corresponding parent transaction or the top-level transaction.

Triggers have become commonplace mechanisms in commercial relational databases and execute as subtransactions according to the semantics of closed nested transactions. The order of execution may be immediate or deferred, meaning that the trigger

executes either at the point of occurrence of the triggering event or at the end of the triggering transaction, respectively. Some extended transaction models allow the triggering of subtransactions as detached or autonomous transactions, i.e., following the semantics of open nested transactions. This, however, implies all the problems resulting from the violation of the isolation and atomicity properties and requires the definition of compensating transactions for those triggers.

The concepts developed as part of extended transaction models resulted in the definition of the CORBA Activity Service Framework. The Activity Service is a general purpose event signalling mechanism that can be used to program activities to coordinate themselves according to the transaction model under consideration. The Activity Service has also been incorporated in the J2EE framework but is not widely used.

Key Applications

The main application areas for open nested transaction models are multidatabase systems in which autonomous systems are loosely coupled. Web Services with their loose coupling, distribution and often long running interactions prompted renewed interest in extended transaction models and open nested transactions. Applications based on Web Services span the whole range of e-business applications. Mobile commerce is another key application domain for open nested transactions. Proposed mobile transaction models are instances of open nested transaction models, e.g., the model proposed by Chrysanthis and extended in Kangaroo Transactions.

Future Directions

Several proposals for long running activities based on Web Services have been advanced by different consortia. The OASIS Business Transaction Protocol was proposed in 2001 by a consortium consisting of HP, BEA, and Oracle. Their model provides for the execution of business logic between the two phases of a 2 phase commit protocol. Arjuna, Oracle, Sun Microsystems, IONA Technologies and Fujitsu in 2003 founded the OASIS Web Services Composite Application Framework that defines three transaction protocols, each aimed at a specific use case. The WS-TX builds on and extends the web Services Coordination specification and provides two kinds of transaction models. These are not meant to cover all possible use cases and can be extended with the semantics of other extended transaction models as required by emerging applications.

Atomic Transactions are meant for short lived transactional interactions within trusted domains and provide full isolation (no dirty reads and repeatable reads), atomicity (well-formedness and two-phase commit protocol), and durability. For loosely coupled, long running interactions Web Services Transactions can use the Business Activity protocol, a more flexible transaction and coordination protocol that relaxes the ACID properties and draws heavily on previous research on open nested transaction models.

Business Activities are designed for long-lived transactions. They are based on the original Sagas open nested transaction model. Services are treated as Sagas and if there is the appropriate compensation behaviour defined, they can execute the undo behaviour if so instructed by the Business Activity. The responsibility of writing correct compensation services to ensure consistency rests with the developer of a service.

Business Activities consist of (possibly nested) Saga-like service invocations. Such scopes can handle errors, i.e., the abort of a task, through application logic and continue processing without globally aborting. Upon completion a child subtransaction can either leave the scope of the Business Activity or it can signal the parent that its work can be compensated later. In any event, the visibility rules are such that the results of child tasks can be seen by the outside world. Business Activities must record application state and keep a record of all sent and received messages, all request messages must be acknowledged, and requests and responses are decoupled. Two different protocols exist for Business Activities: Business Agreement With Coordinator Complete and Business Agreement With Participant Complete. The main difference between these two protocols is that in the former a participating task may not leave the scope of the Business Activity unilaterally and must wait for it to terminate. In case of abort, the subtask must compensate. In the latter a task may leave the scope of the Business Activity unilaterally.

Cross-references

► Extended Transaction Models and the ACTA Framework
► Compensating Transactions
► ConTract
► CORBA
► Distributed Database Systems

- ▶ Distributed Transaction Management
- ▶ Extended Transaction Models
- ▶ Flex transactions
- ▶ Loose Coupling
- ▶ Multilevel Transactions and Object-Model Transactions
- ▶ Nested Transaction Models
- ▶ Orchestration
- ▶ Sagas
- ▶ Transaction
- ▶ Web Transactions
- ▶ Workflow Transactions

Recommended Reading

1. Buchmann A., Özsu M.T., Hornick M., Georgakopoulos D., and Manola F. A transaction model for active distributed object systems. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.
2. Cabrera L.F., Copeland G., Feingold M. et al. Web services atomic transaction (WS-AtomicTransaction), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-AtomicTransaction.pdf.
3. Cabrera L.F., Copeland, G., Feingold M. et al., Web services business activity framework (WS-BusinessActivity), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-BusinessActivity.pdf.
4. Cabrera L.F., Copeland G., Feingold M. et al. Web services coordination (WS-Coordination), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-Coordination.pdf.
5. Chrysanthis P.K. Transaction processing in a mobile environment. In Proc. IEEE Workshop on Advances in Parallel and Distributed Systems. 1993, pp. 77–82.
6. Chrysantis P. and Ramamritham K. ACTA: The saga continues. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid, (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.
7. Dunham M.H., Helal A., and Balakrishnan S. A mobile transaction model that captures both data and movement behavior. MONET, 2(2):149–162, 1997.
8. Elmagarmid A.K. (ed.), Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers, Los Altos, CA, 1992.
9. Garcia-Molina H. and Salem K. SAGAS. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
10. Houston I., Little M., Robinson I., Shrivastava S.K., and Wheater S.M. The CORBA activity service framework for supporting extended transactions. In Proc. IFIP/ACM Int. Conf. on Dist. Syst. Platforms, 2001, pp. 197–215.
11. Little M. A history of extended transactions. Available at: http://www.infoq.com/articles/History-of-Extended-Transactions.
12. Moss E. Nested Transactions. MIT Press, Cambridge, MA, 1985.
13. Weikum G. and Schek H.J. Concepts and applications of multilevel transactions and open nested transactions. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.

Open Nested Transactions

- ▶ Multilevel Transactions and Object-Model Transactions

Operating Characteristic

- ▶ Receiver Operating Characteristic (ROC)

Operator-Level Parallelism

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Inter-operator parallelism

Definition

Operator-level parallelism (or inter-operator parallelism) is a form of intra-query parallelism obtained by executing concurrently several operators of the same query. By contrast, intra-operator parallelism is obtained by executing the same operator on multiple processors, with each instance working on a different subset of data.

Historical Background

Parallelism has been a key focus of database research since the 1970s. For example, as early as 1978 Teradata was building highly-parallel database systems and quietly pioneered many of the ideas on parallel query execution [5]. However, the intra-query parallelism employed by these early systems was mostly intra-operator or independent parallelism (see Classes of Parallelism below). Gamma [4] was one of the first database systems that allowed operator-level parallelism through pipelining.

Foundations

Parallel processing uses multiple processors cooperatively to improve the performance of application programs. With relations growing larger and queries

becoming more complex, parallel processing is an increasingly attractive option for improving the performance of database management systems. The widespread adoption of the relational database model has enabled the parallel execution of relational queries, as these queries are composed of uniform operators applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. At the same time, multiprocessor systems and high-speed interconnection networks have become mainstream, providing an excellent basis for parallel execution.

Classes of Parallelism

Parallelism in the evaluation of database queries is classified into two main categories: inter-query parallelism (see Inter-Query Parallelism), in which different queries execute on different processors to improve the overall throughput of the system, and intra-query parallelism, in which several processors cooperate for the faster execution of a single query. Intra-query parallelism is further classified into intra-operator and inter-operator parallelism. Intra-operator parallelism (see Intra-Operator Parallelism) is obtained by executing the same operator on multiple processors, with each instance working on a different subset of data. Operator-level parallelism (or inter-operator parallelism), is obtained by executing concurrently several operators of the same query. This latter form of parallelism is the subject of this chapter.

Operator-level parallelism is in two forms: independent parallelism and pipelined parallelism. Independent parallelism (or bushy parallelism) is achieved when there is no dependency between the operators executed in parallel. For example, consider a simple query plan with two select operators and a join, that it is not nested-loops. The select operators are independent of each other and can execute concurrently, thereby exhibiting independent parallelism. Algebraically, independent parallelism can be expressed by a relation of the form f(g(X), h(Y)), where X and Y are relations and f, g, and h are relational operators. In this example, g and h exhibit independent parallelism.

Because the operators participating in bushy parallelism are independent, they do not directly affect the execution of one another. Interference is only indirect, e.g., due to the concurrent use of shared resources like disks, caches, or main memory bandwidth. Thus, independent parallelism is simpler to employ as it is easier to schedule the execution of the participating independent operators, and it has the potential to deliver high performance improvements.

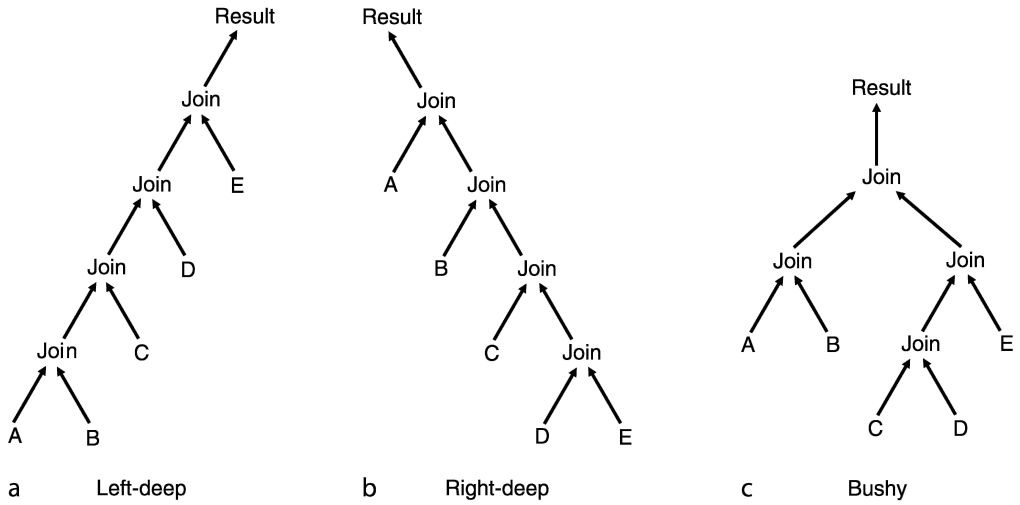
Alternatively, operator-level parallelism can take the form of pipelined parallelism, also called dataflow parallelism. Pipelined parallelism can be achieved when the concurrent operators form producer/consumer pairs in which the consumer can start executing without requiring its entire input to be available. For example, consider the aforementioned simple query that consists of two select operators and a join. The select operator can execute in parallel with the join operator. However, they are not independent, because the intermediate results produced by the select are consumed by the subsequent join. Thus, the tuples output by the select can be pipelined to the join operator to be consumed immediately. This example illustrates a significant advantage of pipelined parallelism: intermediate results are used immediately and are not materialized, saving memory and disk accesses. Algebraically, pipelined parallelism can be expressed by a relation of the form f(g(X)), where X is a relation and f and g are relational operators. The operators that cannot produce tuples unless they have processed their entire input are called Stop-&-Go operators.

Effect of Query Plan Selection on Operator-Level Parallelism

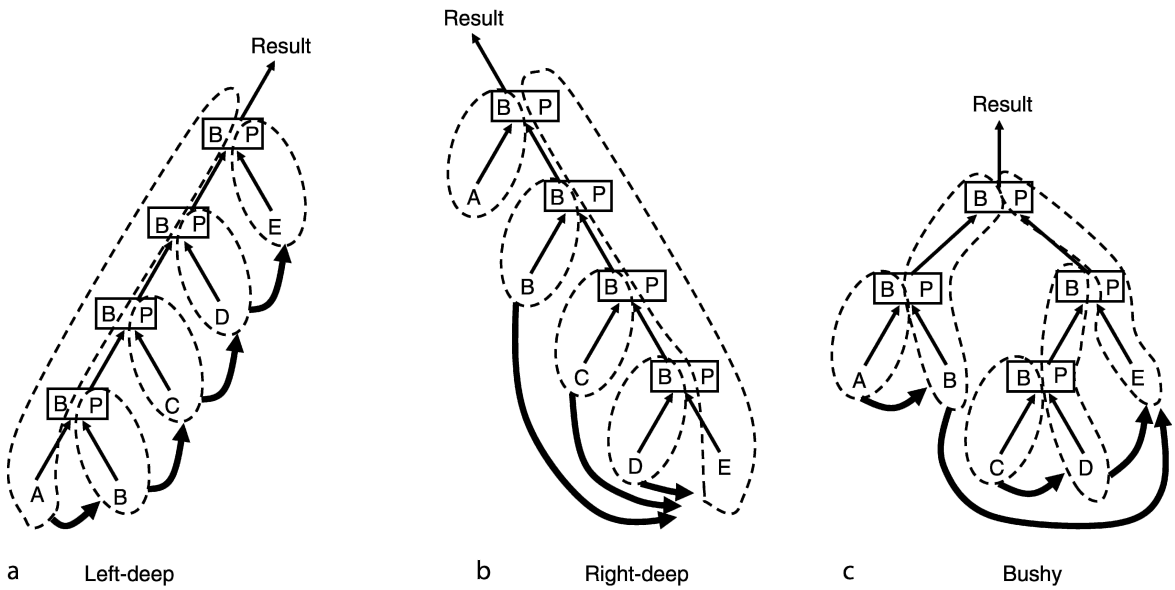
The query plan determines the execution sequence of a query's operators. The selection of a query plan greatly affects the degree of attainable operator-level parallelism. To illustrate this point, and without loss of generality, let's assume a multi-way hash-join query with four joins: $A \times B \times C \times D \times E$ where A, B, C, D, E are relations and $\times$ is the join operator. The query plan is typically depicted graphically as a tree with vertices representing relations. Because every operator in the relational model defines a new relation, the operators in the internal vertices denote the relation they represent. If an operator Y takes relation X as one of its inputs, then a directed edge connects X to Y in the tree representation.

Three forms of query execution trees are explored in the literature: left-deep trees, right-deep trees, and bushy trees. Figure 1 shows the query execution trees for the example multi-way join query used above. Left-deep trees and right-deep trees represent the two extreme strategies of query execution, while bushy trees claim a middle ground.

To compare the trade-offs between the alternative query plans, Fig. 2 shows the execution dependencies



Operator-Level Parallelism. Figure 1. (a) Left-deep, (b) right-deep, and (c) bushy query plans.



Operator-Level Parallelism. Figure 2. Operator dependencies for (a) left-deep, (b) right-deep, and (c) bushy plans.

between the operators of each execution strategy in Fig. 1. The execution dependencies are shown using operator dependency graphs [10]. The dotted lines encircle the operators amenable to pipelined parallelism. The bold directed arcs between subgraphs show which sets of operators must be executed before other sets of operators are executed, thereby determining the maximum level of parallelism and resource requirements (e.g., memory) for the query. As discussed in [10] hash joins have two distinct phases, the build and the probe phase. Since the first phase must completely precede the second, the hash joins in Fig. 2 can be viewed as if consisting of two operators, the build and the probe operator.

The operator dependency graph of the left-deep query plan shows that only a scan, the build phase of a join, and the probe phase of a join can execute in parallel. Thus, although the left-deep query plan has low memory requirements (it needs enough memory to fit the hash tables of two joins) it offers only limited

pipelined parallelism and no independent parallelism. In contrast, the operator dependency graph for the right-deep query plan shows that significant operator-level parallelism is available: all scans but one have a producer/consumer relationship with the build phase of the subsequent hash join, thereby exhibiting pipelined parallelism, while the scan/build pairs are independent of one another so they exhibit independent parallelism. However, the high degree of parallelism comes at the expense of high shared resource pressure. The hash tables for all joins should fit in main memory simultaneously, or risk spilling to disk.

Finally, the operator dependency graph for the bushy query plan has characteristics that are between the left-deep and the right-deep query plans. The bushy plan enables independent parallelism, albeit at a lower degree than the right-deep plan, as about half the scans can proceed in parallel. However, the bushy plan imposes lower pressure on shared resources than the right-deep plan, because fewer operators execute in parallel. Bushy query plans allow the formation of deeper pipelines, some of which extend all the way from a leaf to the root of the tree. Thus, bushy trees enable pipelined parallelism as well, but it may be harder to balance the load within their deeper pipelines due to execution skew.

Because bushy plans achieve a balance between pipelined parallelism, independent parallelism, and resource utilization, researchers further investigated their applicability in improving query execution. For example, segmented right-deep trees [3] (bushy trees of right-deep subtrees) have been shown to outperform their left-deep and right-deep counterparts.

Other Factors Limiting Operator-Level Parallelism

The selection of the query plan determines the degree of available operator-level parallelism. This section discusses factors that limit the effectiveness of operator-level parallelism, given a query plan. Among other things, the discussion in this section touches on issues of load balancing and processor allocation.

The operators participating in independent parallelism interfere only indirectly through the concurrent use of shared resources. The factors limiting the benefit of independent parallelism are the constraints imposed by the hardware resources. All relations that execute in parallel produce intermediate results which increase the data footprint of the application, resulting in higher cache miss rates and higher memory pressure. The larger data footprint, in turn, may oversubscribe memory bandwidth or induce more spills to disk if the relations do not fit in main memory.

Resource contention affects pipelined parallelism as well, but to a lesser degree because the intermediate data in pipelined parallelism are short lived as they are consumed immediately after their production. The benefits of pipelined parallelism are generally limited by three factors [5]: (i) relational pipelines are rarely very long – a chain of length ten is unusual. (ii) some relational operators are blocking operators, i.e., they do not emit their first output until they have consumed all their inputs. Sort and the partitioning phase of hash join are examples of blocking relational operators. Such operators cannot be pipelined, and (iii) there are dependencies between the operators participating in a pipeline. Often, the execution cost of one operator is much greater than the others, a phenomenon referred to as execution skew. In this case, the performance of the pipelined execution is dominated by the slowest operator, which significantly limits parallelism.

The execution skew also gives rise to startup/teardown execution delays: processors assigned to operators at the end of a pipeline are idle at the beginning of the computation, whereas processors assigned to operators at the beginning of a pipeline are idle towards the end of the computation. It is important to note here that data skew may induce execution skew in some cases. For example, in a sort-merge join with data skew, some sort partitions may be much larger than others, creating execution skew.

A potential solution to execution skew is to predict the execution load for each operator and schedule them accordingly across the parallel processors. However, the predictions may fail as the costs are estimated in the query optimization phase using typically inaccurate cost models and statistics.

The assignment of processors to operators and their scheduling is an important and hard problem that affects all forms of operator-level parallelism. It is an optimization problem that attempts to utilize all the available processors efficiently to minimize the execution time of a query. Sometimes operators may need to be scheduled as a team (e.g., producer/consumer pairs), while other times gang scheduling should be avoided (e.g., scheduling together the first and the last operator of a deep pipeline would leave the last

operator mostly idle). Scheduling is easier when there are no dependencies between the operators executing in parallel, in which case load balancing is of primary concern.

The processor allocation is based on the selection of the query plan and estimates on the execution cost of each operator. If the execution cost of some operators is much higher than others, the system may be subject to fragmentation: after a sequence of processor allocations and releases there may be a few processors left idle and rebalancing the workload dynamically is not always possible or beneficial. These cases may benefit from the concurrent employment of multiple forms of parallelism (see next section). However, the application of multiple forms of parallelism adds an extra dimension to the processor allocation problem, making it harder to solve.

Relation to Inter-Query and Intra-Operator Parallelism

Operator-level parallelism is orthogonal to inter-query and intra-operator parallelism and can work synergistically with them to improve performance even further. For example, if there is imbalance in the execution times of a query's operators and there are free processors, intra-operator parallelism can be applied to split a long-running operator into multiple ones, each executing on a smaller subset of data. This will allow for faster execution of the expensive operators and may balance the execution times of operators participating in a pipeline, avoiding execution skew.

For a more comprehensive treatment of operator-level parallelism, the interested reader is referred to [5,10,11].

Key Applications

Several parallel database systems have been developed that utilize operator-level parallelism to improve performance. Systems built in academic institutions include GAMMA [4], BUBBA [2], Volcano [6], MonetDB/X100 [1], and StagedDB [7]. Commercial systems that support operator-level parallelism include Oracle [9] and IBM DB2 [8].

Cross-references

- ▶ Data Skew
- ▶ Execution Skew
- ▶ Intra-Operator Parallelism
- ▶ Inter-Query Parallelism
- ▶ Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join
- ▶ Parallel Query Processing
- ▶ Pipelining
- ▶ Query Plan
- ▶ Stop-&-Go Operator

Recommended Reading

1. Boncz P., Zukowski M., and Nes N. MonetDB/X100: hyper-pipelining query execution. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 225–237.
2. Boral H. Prototyping bubba: a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2(1), 1990.
3. Chen M.-S., Lo M., Yu P.S., and Young H.C. Using segmented right-deep trees for the execution of pipelined hash joins. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 15–26.
5. DeWitt D.J. and Gray J. Parallel database systems: the future of high-performance database computing. Commun. ACM, 35(6):85–98, 1992.
4. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – A high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
6. Graefe G. Volcano – an extensible and parallel query evaluation system. IEEE Trans. Knowl. Data Eng., 6(1):120–135, 1994.
7. Harizopoulos S. and Ailamaki A. Staged D.B.: designing database servers for modern hardware. IEEE Data Eng. Bull., 28 (2):11–16, 2005.
8. IBM Corp. DB2 Version 9 Performance Guide. Part No. SC10–4222–00, 2006.
9. Oracle Corp. Oracle Database Data Warehousing Guide. 10g Release 1 (10.1). Part No. B10736–01, 2003.
10. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 469–480.
11. Yu P.S., Chen M.-S., Wolf J.L., and Turek J.J. Parallel query processing. In Advanced Database Systems, N. Adam, B. Bhargava, (eds.). LNCS, vol. 759, Springer, Berlin, 1993, pp. 239–258.

Operator Scheduling

- ▶ Scheduling Strategies for Data Stream Processing

Operator Tree

- ▶ Query Plan

Opinion Mining

Bing Liu
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Sentiment analysis

Definition

Given a set of evaluative text documents D that contain opinions (or sentiments) about an object, opinion mining aims to extract attributes and components of the object that have been commented on in each document $d \in D$ and to determine whether the comments are positive, negative or neutral.

Historical Background

Textual information in the world can be broadly classified into two main categories, *facts* and *opinions*. Facts are objective statements about entities and events in the world. Opinions are subjective statements that reflect people's sentiments or perceptions about the entities and events. Much of the existing research on text information processing has been (almost exclusively) focused on mining and retrieval of factual information, e.g., information retrieval, Web search, and many other text mining and natural language processing tasks. Little work has been done on the processing of opinions until only recently. Yet, opinions are so important that whenever one needs to make a decision one wants to hear others' opinions. This is not only true for individuals but also true for organizations.

One of the main reasons for the lack of study on opinions is that there was little opinionated text before the World Wide Web. Before the Web, when an individual needs to make a decision, he/she typically asks for opinions from friends and families. When an organization needs to find opinions of the general public about its products and services, it conducts surveys and focused groups. With the Web, especially with the explosive growth of the user generated content on the Web, the world has changed. One can post reviews of products at merchant sites and express views on almost anything in Internet forums, discussion groups, and blogs, which are collectively called the *user generated content*. Now if one wants to buy a product, it is no longer necessary to ask one's friends and families because there are plenty of product reviews on the Web that give the opinions of the existing users of the product. For a company, it may no longer need to conduct surveys, to organize focused groups or to employ external consultants in order to find consumer opinions or sentiments about its products and those of its competitors.

Finding opinion sources and monitoring them on the Web, however, can still be a formidable task because a large number of diverse sources exist on the Web and each source also contains a huge volume of information. In many cases, opinions are hidden in long forum posts and blogs. It is very difficult for a human reader to find relevant sources, extract pertinent sentences, read them, summarize them and organize them into usable forms. An automated opinion mining and summarization system is thus needed. *Opinion mining*, also known as *sentiment analysis*, grows out of this need.

Research on opinion mining started with identifying *opinion* (or *sentiment*) *bearing words*, e.g., great, amazing, wonderful, bad, and poor. Many researchers have worked on mining such words and identifying their *semantic orientations* (i.e., positive or negative). In [5], the authors identified several linguistic rules that can be exploited to identify opinion words and their orientations from a large corpus. This method has been applied, extended and improved in [3,8,12]. In [6,9], a bootstrapping approach is proposed, which uses a small set of given seed opinion words to find their synonyms and antonyms in WordNet (http://wordnet.princeton.edu/). The next major development is sentiment classification of product reviews at the document level [2,11,13]. The objective of this task is to classify each review document as expressing a positive or a negative sentiment about an object (e.g., a movie, a camera, or a car). Several researchers also studied sentence-level sentiment classification [9,14,15], i.e., classifying each sentence as expressing a positive or a negative opinion. The model of feature-based opinion mining and summarization is proposed in [6,10]. This model gives a more complete formulation of the opinion mining problem. It identifies the key pieces of information that should be mined and describes how a structured opinion summary can be produced from unstructured texts. The problem of mining opinions from comparative sentences is introduced in [4,7].

Foundations

Model of Opinion Mining

In general, opinions can be expressed on anything, e.g., a product, a service, a topic, an individual, an organization, or an event. The general term *object* is used to denote the entity that has been commented on. An object has a set of *components* (or *parts*) and a set of *attributes*. Each component may also have its sub-components and its set of attributes, and so on. Thus, the object can be hierarchically decomposed based on the *part-of* relationship.

Definition (object): An *object* O is an entity which can be a product, topic, person, event, or organization. It is associated with a pair, (T, A), where T is a hierarchy or taxonomy of *components* (or *parts*) and *sub-components* of O, and A is a set of *attributes* of O. Each component has its own set of sub-components and attributes.

In this hierarchy or tree, the root is the object itself. Each non-root node is a component or sub-component of the object. Each link is a part-of relationship. Each node is associated with a set of attributes. An opinion can be expressed on any node and any attribute of the node.

However, for an ordinary user, it is probably too complex to use a hierarchical representation. To simplify it, the tree is flattened. The word "*features*" is used to represent both components and attributes. Using features for objects (especially products) is quite common in practice. Note that in this definition the object itself is also a feature, which is the root of the tree.

Let an evaluative document be d, which can be a product review, a forum post or a blog that evaluates a particular object O. In the most general case, d consists of a sequence of sentences $d = \langle s_1, s_2,...,s_m \rangle$.

Definition (opinion passage on a feature): The *opinion passage* on a feature f of the object O evaluated in d is a group of consecutive sentences in d that expresses a positive or negative opinion on f.

This means that it is possible that a sequence of sentences (at least one) together expresses an opinion on an object or a feature of the object. It is also possible that a single sentence expresses opinions on more than one feature, e.g., "The picture quality of this camera is good, but the battery life is short."

Definition (opinion holder): The *holder* of a particular opinion is a person or an organization that holds the opinion.

In the case of product reviews, forum postings and blogs, opinion holders are usually the authors of the posts. Opinion holders are important in news articles because they often explicitly state the person or organization that holds a particular opinion [9]. For example, the opinion holder in the sentence "John expressed his disagreement on the treaty" is "John."

Definition (semantic orientation of an opinion): The *semantic orientation* of an opinion on a feature f states whether the opinion is positive, negative or neutral.

Putting things together, a *model* for an object and a set of opinions on the features of the object can be defined, which is called the *feature-based opinion mining model*.

Model of Feature-Based Opinion Mining: An object O is represented with a finite set of features, $F = \{f_1, f_2,...,f_n\}$, which includes the object itself. Each feature $f_i \in F$ can be expressed with a finite set of words or phrases W_i, which are *synonyms*. That is, there is a set of corresponding synonym sets $W = \{W_1, W_2,..., W_n\}$ for the n features. In an evaluative document d which evaluates object O, an opinion holder j comments on a subset of the features $S_j \subseteq F$. For each feature $f_k \in S_j$ that opinion holder j comments on, he/she chooses a word or phrase from W_k to describe the feature, and then expresses a positive, negative or neutral opinion on f_k. The opinion mining task is to discover all these hidden pieces of information from a given evaluative document d.

Mining output: Given an evaluative document d, the mining result is a set of quadruples. Each quadruple is denoted by (H, O, f, SO), where H is the opinion holder, O is the object, f is a feature of the object and SO is the semantic orientation of the opinion expressed on feature f in a sentence of d. Neutral opinions are ignored in the output as they are not usually useful.

Given a collection of evaluative documents D containing opinions on an object, three main technical problems can be identified (clearly there are more):

Problem 1: Extracting object features that have been commented on in each document $d \in D$.

Problem 2: Determining whether the opinions on the features are positive, negative or neutral.

Problem 3: Grouping synonyms of features (as different opinion holders may use different words or phrase to express the same feature).

Opinion Summary: There are many ways to use the mining results. One simple way is to produce a

feature-based summary of opinions on the object [6]. An example is used to illustrate what that means.

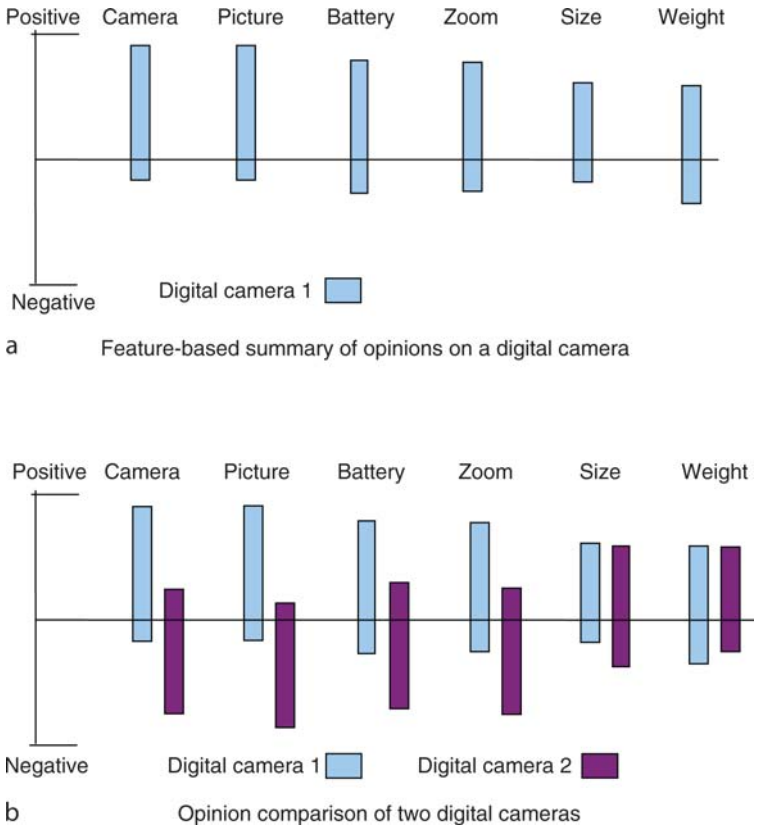
Figure 1 summarizes the opinions in a set of reviews of a particular digital camera, *digital_camera*_1. The opinion holders are omitted. In the figure, "CAMERA" represents the camera itself (the root node of the object hierarchy). One hundred and twenty-five reviews expressed positive opinions on the camera and seven reviews expressed negative opinions on the camera. "picture quality" and "size" are two product features. One hundred and twenty-three reviews expressed positive opinions on the picture quality, and only 6 reviews expressed negative opinions. The ⟨individual review sentences⟩ points to the specific sentences and/or the whole reviews that give the positive or negative comments about the feature. With such a summary, the user can easily see how existing customers feel about the digital camera. If he/she is very interested in a particular feature, he/she can drill down by following the ⟨individual review sentences⟩ link to see why existing customers like it and/or dislike it.

*Digital_camera*_1:

Camera:
  Positive: 125 <individual review sentences>
  Negative: 7 <individual review sentences>
Feature: **picture quality**
  Positive: 123 <individual review sentences>
  Negative: 6 <individual review sentences>
Feature: **size**
  Positive: 82 <individual review sentences>
  Negative: 10 <individual review sentences>
...

Opinion Mining. Figure 1. An example of a feature-based summary of opinions.

The summary in Fig. 1 can be easily visualized using a bar chart [10]. Figure 2(a) shows such a chart. In the figure, each bar above the *X*-axis gives the number of positive opinions on a feature (listed at the top), and the bar below the *X*-axis gives the number of negative opinions on the same feature. Obviously, other visualizations are also possible. For example, one may only show the percentage of positive (or negative) opinions on each feature. Comparing opinion summaries of a



Opinion Mining. Figure 2. Visualization of feature-based opinion summary and comparison.

few competing objects is even more interesting [10]. Figure 2(b) shows a visual comparison of consumer opinions on two competing digital cameras. One can clearly see how consumers view different features of each camera.

Sentiment Classification

Sentiment classification has been widely studied in the natural language processing (NLP) community [e.g., 2,11,13]. It is defined as follows: Given a set of evaluative documents D, it determines whether each document $d \in D$ expresses a positive or negative opinion (or sentiment) on an object. For example, given a set of movie reviews, the system classifies them into positive reviews and negative reviews.

This is clearly a classification learning problem. It is similar but also different from the classic topic-based text classification, which classifies documents into predefined topic classes, e.g., politics, sciences, and sports. In topic-based classification, topic related words are important. However, in sentiment classification, topic-related words are unimportant. Instead, opinion words that indicate positive or negative opinions are important, e.g., great, excellent, amazing, horrible, bad, worst, etc. There are many existing techniques. Most of them apply some forms of machine learning techniques for classification (e.g., [11]). Custom-designed algorithms specifically for sentiment classification also exist, which exploit opinion words and phrases together with some scoring functions [2,13].

This classification is said to be at the document level as it treats each document as the basic information unit. Sentiment classification thus makes the following assumption: Each evaluative document (e.g., a review) focuses on a single object O and contains opinions of a single opinion holder. Since in the above opinion mining model an object O itself is also a feature (the root node of the object hierarchy), sentiment classification basically determines the semantic orientation of the opinion expressed on O in each evaluative document that satisfies the above assumption.

Apart from the document-level sentiment classification, researchers have also studied classification at the *sentence-level*, i.e., classifying each sentence as a subjective or objective sentence and/or as expressing a positive or negative opinion [9,14,15]. Like the document-level classification, the sentence-level sentiment classification does not consider object features that have been commented on in a sentence. Compound sentences are also an issue. Such a sentence often express more than one opinion, e.g., "The picture quality of this camera is amazing and so is the battery life, but the viewfinder is too small."

Feature-Based Opinion Mining

Classifying evaluative texts at the document level or the sentence level does not tell what the opinion holder likes and dislikes. A positive document on an object does not mean that the opinion holder has positive opinions on all aspects or features of the object. Likewise, a negative document does not mean that the opinion holder dislikes everything about the object. In an evaluative document (e.g., a product review), the opinion holder typically writes both positive and negative aspects of the object, although the general sentiment on the object may be positive or negative. To obtain such detailed aspects, going to the feature level is needed. Based on the model presented earlier, three key mining tasks are:

1. Identifying *object features*: For instance, in the sentence "The picture quality of this camera is amazing," the object feature is "picture quality." In [10], a supervised pattern mining method is proposed. In [6,12], an unsupervised method is used. The technique basically finds frequent nouns and noun phrases as features, which are usually genuine features. Clearly, many information extraction techniques are also applicable, e.g., conditional random fields (CRF), hidden Markov models (HMM), and many others.
2. Determining opinion orientations: This task determines whether the opinions on the features are positive, negative or neutral. In the above sentence, the opinion on the feature "picture quality" is positive. Again, many approaches are possible. A lexicon-based approach has been shown to perform quite well in [3,6]. The lexicon-based approach basically uses opinion words and phrases in a sentence to determine the orientation of an opinion on a feature. A relaxation labeling based approach is given in [12]. Clearly, various types of supervised learning are possible approaches as well.
3. Grouping synonyms: As the same object features can be expressed with different words or phrases, this task groups those synonyms together. Not much research has been done on this topic. See [1] for an attempt on this problem.

Mining Comparative and Superlative Sentences

Directly expressing positive or negative opinions on an object or its features is only one form of evaluation. Comparing the object with some other similar objects is another. Comparisons are related to but are also different from direct opinions. For example, a typical opinion sentence is "The picture quality of camera x is great." A typical comparison sentence is "The picture quality of camera x is better than that of camera y." In general, a comparative sentence expresses a relation based on similarities or differences of more than one object. In English, comparisons are usually conveyed using the *comparative* or the *superlative* forms of adjectives or adverbs. The structure of a comparative normally consists of the stem of an adjective or adverb, plus the suffix *-er*, or the modifier "more" or "less" before the adjective or adverb. The structure of a superlative normally consists of the stem of an adjective or adverb, plus the suffix *-est*, or the modifier "most" or "least" before the adjective or adverb. Mining of comparative sentences basically consists of identifying what features and objects are compared and which objected are preferred by their authors (opinion holders). Details can be found in [4,7].

Key Applications

Opinions are so important that whenever one needs to make a decision, one wants to hear others' opinions. This is true for both individuals and organizations. The technology of opinion mining thus has a tremendous scope for practical applications.

Individual consumers: If an individual wants to purchase a product, it is useful to see a summary of opinions of existing users so that he/she can make an informed decision. This is better than reading a large number of reviews to form a mental picture of the strengths and weaknesses of the product. He/she can also compare the summaries of opinions of competing products, which is even more useful.

Organizations and businesses: Opinion mining is equally, if not even more, important to businesses and organizations. For example, it is critical for a product manufacturer to know how consumers perceive its products and those of its competitors. This information is not only useful for marketing and product benchmarking but also useful for product design and product developments.

Cross-references

▶ Text Mining

Recommended Reading

1. Carenini G., Ng R., and Zwart E. Extracting Knowledge from Evaluative Text. In Proc. 3rd Int. Conf. on Knowledge Capture, 2005.
2. Dave D., Lawrence A., and Pennock D. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In Proc. 12th Int. World Wide Web Conference, 2003.
3. Ding X., Liu B., and Yu P. A holistic lexicon-based approach to opinion mining. In Proc. 1st ACM Int. Conf. on Web Search and Data Mining. 2008.
4. Ganapathibhotla G. and Liu B. Identifying preferred entities in comparative sentences. In Proc. 22nd Int. Conf. on Computational Linguistics. 2008.
5. Hatzivassiloglou V. and McKeown K. Predicting the semantic orientation of adjectives. In Proc. 8th Conf. European Chapter of Assoc. Comp. Linguistics. 1997.
6. Hu M. and Liu B. Mining and summarizing customer reviews. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004.
7. Jindal N. and Liu B. Mining comparative sentences and relations. In Proc. of National Conf. on Artificial Intelligence, 2006.
8. Kanayama H. and Nasukawa T. Fully automatic lexicon expansion for domain-oriented sentiment analysis. In Proc. 2006 Conf. on Empirical Methods in Natural Language Processing, 2006.
9. Kim S. and Hovy E. Determining the sentiment of opinions. In Proc. 20th Int. Conf. on Computational Linguistics, 2004.
10. Liu B., Hu M., and Cheng J. Opinion observer: analyzing and comparing opinions on the web. In Proc. 14th Int. World Wide Web Conference, 2005.
11. Pang B., Lee L., and Vaithyanathan S. Thumbs up? Sentiment classification using machine learning techniques. In Proc. 2002 Conf. on Empirical Methods in Natural Language Processing, 2002.
12. Popescu A.-M. and Etzioni O. Extracting product features and opinions from reviews. In Proc. 2005 Conf. on Empirical Methods in Natural Language Processing, 2005.
13. Turney P. Thumbs up or thumbs down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In Proc. 40th Annual Mfg. of Assoc. Comp. Linguistics, 2002.
14. Wiebe J. and Riloff E. Creating subjective and objective sentence classifiers from unannotated texts. In Proc. Int. Conf. on Intelligent Text Processing and Computational Linguistics, 2005.
15. Wilson T., Wiebe J., and Hwa R. Just how mad are you? Finding strong and weak opinion clauses. In Proc. National Conf. on Artificial Intelligence, 2004.

Optical Storage

▶ Storage Devices

Optimistic Replication

► Optimistic Replication and Resolution

Optimistic Replication and Resolution

Marc Shapiro
INRIA Paris-Rocquencourt and LIP6, Paris, France

Synonyms

Optimistic Replication; Reconciliation-Based Data Replication; Lazy Replication; Multi-Master System

The term "Optimistic Replication" is prevalent in the distributed systems and distributed algorithms literature. The database literature prefers "Lazy Replication"

Definition

Data replication places physical copies of a shared logical item onto different sites. *Optimistic replication* (OR) [11] allows a program at some site to read or update the local replica at any time. An update is *tentative* because it may conflict with a remote update. Such conflicts are resolved after the fact, in the background. Replicas may *diverge* occasionally but are expected to converge eventually (see entry on EVENTUAL CONSISTENCY).

OR avoids the need for distributed concurrency control prior to using an item. It allows a site to execute even when remote sites have crashed, when network connectivity is poor or expensive, or while disconnected from the network. *Disconnected operation,* the capability to compute while disconnected from a data source, e.g., in mobile computing, requires OR. In *computer-supported co-operative work*, OR enables a user to temporarily insulate himself from other users.

The defining characteristic of OR is that any synchronization between sites occurs in the background, after local termination, i.e., off the critical path of the application.

Historical Background

The first historical instance of OR is Johnson's and Thomas's replicated database (1976). (The vocabulary used in this history is defined in Section "Foundations.")

Usenet News (1979) was an important and inspirational development. News supports a large-scale ever-growing database of (read-only) items, posted by users all over the world. A Usenet site connects infrequently (e.g., daily) with its peers. New items are flooded to other sites, and are received in arbitrary order. Users occasionally observe ordering anomalies, but this is not considered a problem. However, system administrators must deal manually with conflicts over administrative operations.

In 1984, Wuu and Bernstein's replicated mutable key-value-pair database use an operation log, transmitted by an *anti-entropy* protocol: site A sends to site B only the tail of A's log that B has not yet seen [15]. Concurrent operations either commute or have a natural semantic order; non-concurrent operations execute in happens-before order.

The Lotus Notes system (1988) supports co-operative work between mobile enterprise users. It replicates a database of discrete items in a peer-to-peer manner. Notes is state-based, and uses a Last-Writer Wins policy. A deleted item is replaced by a *tombstone*.

Several file systems, designed in the early 1990s to support disconnected work, e.g., Coda [6], are state based and uses version vectors for conflict detection. Conflicts over some specific object types (e.g., directories or mailboxes) cause automatic resolver programs to run. The others must be resolved manually.

Golding (1992) [3] studies a replicated database of mutable key-value pairs. This system purges an operation from the log when it can prove that it was delivered to all sites. Consistency is ensured by defining a total order of operations.

Bayou (1994–1997) is an innovative general-purpose database for mobile users [9]. Bayou is operation-based and uses an anti-entropy protocol. Each site executes transactions in arbitrary order; transactions remain tentative. The eventual serialization order is the order of execution at a designated *primary* site. Other sites roll back their tentative state, and re-execute committed transactions in commit order.

In 1996, Gray et al. argued that OR databases for disconnected work cannot scale [4], because conflict reconciliation is expensive, conflict probability rises as the third power of the number of nodes, and the wait probability further increases quadratically with disconnection time.

Breitbart et al. [1] describe a partially-replicated database that uses a form of OR. Each item has a designated primary site and may be replicated at any number of secondary sites. A read may occur at a secondary site but a write must occur on the primary. It follows that write transactions update a single site. If transactions are serialisable at each site, and update propagation is restricted to avoid ordering anomalies, then transactions are serialisable despite lazy propagation.

The Computer-Supported Cooperative Work (CSCW) community invented (1989) a form of OR called Operational Transformation (OT). Conflicting operations are *transformed*, by modifying their arguments, in order to execute in arbitrary order [12].

Foundations

Figure 1 depicts a logical item *x*, concretely replicated at three different sites. In OR, any site may *submit* or *initiate* a transaction reading or writing the local replica. If the transaction succeeds locally, the system *propagates* it to other sites, and *replays* the transaction on the remote sites, in a *lazy* manner, in the background. Local execution is *tentative* and may be rolled back later, because of a *conflict* with a concurrent remote transaction. (The happens-before and concurrency relations are defined formally by Lamport [7]. Transaction A happens-before B, if B was initiated on some site after A executed at that site. Two transactions are concurrent if neither happens-before the other.)

OR is opposed to pessimistic (or eager) replication, where a local transaction terminates only when it commits globally. Pessimistic replication establishes a total order for committed transactions, at the latest when each transaction terminates. In contrast, OR generally relaxes the ordering requirements and/or converges to a common order *a posteriori*. The effects of a tentative transaction can be observed, thus OR protocols may violate the *isolation* property and allow cascading aborts and retries to occur.

Transmitting and Replaying Updates

In OR, updates are propagated lazily, in the background, after the transaction has terminated locally. Transmission usually uses peer-to-peer epidemic or anti-entropy techniques (see entry on Peer-to-Peer Content Distribution).

A site that receives a remote update *replays* it, i.e., incorporates it into the local replica. There are two main approaches. In the *state-based* approach, the initiator site transmits the after-values of the transaction, and other sites assign the after-value to their local replica. In the *operation-based* approach, the initiator sends the program of the transaction itself, and other sites re-execute the transaction.

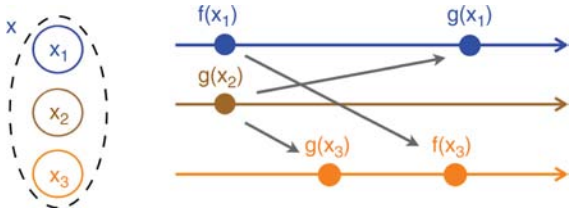
State-based replay is guaranteed to be deterministic. State-based replay can be more efficient, since the replay code is just a write. On the downside, if the granularity is large, then state-based transmission is expensive and replay is subject to false conflicts. Furthermore, logical operations are more likely to commute than writes, thus operation-based replay typically causes fewer aborts.

Conflicts

Each transaction taken individually is assumed correct (the C of the ACID properties), i.e., it maintains semantic invariants. For example, ensuring that a bank account remains positive, or that a person is not scheduled in two different meetings at the same time.

As is clear from Fig. 1, concurrent transactions may be delivered to different sites in different orders. (Dependent transactions are assumed to execute in dependency order; see Section "Scheduling Transactions Content and Ordering.") However, consistency requires that local schedules be equivalent. In this respect, one may classify pairs of concurrent transactions as commuting, non-commuting, and antagonistic. Transactions *conflict* if they are mutually non-commuting or mutually antagonistic.

The relative execution order of *commuting* transactions is immaterial; they require no remote synchronization. Formally, two transactions T_1 and T_2 commute if execution order $T_1;T_2$ returns the same results to the



Optimistic Replication and Resolution. Figure 1. Three sites with replicas of logical item *x*. Site 1 initiates transaction *f*, Site 2 initiates *g*. The system propagates and replays on remote sites. Site 3 executes in the order *g;f*, whereas Site 1 replays *f* before *g*. Eventually, Site 2 will also execute *f*.

user and leaves the database in the same state as the order $T_2;T_1$. For instance, depositing €10 in a bank account commutes with a depositing €20 into the same account, and also commutes with withdrawing €100 from an independent account.

If running concurrent transactions together would violate an invariant, they are said *antagonistic*. Safety requires aborting one or the other (or both). For instance, if T_1 schedules me in a meeting from 10:00 to 12:00, and T_2 schedules a meeting from 11:00 to 13:00, they are antagonistic since no combination of both T_1 and T_2 can be correct.

If two transactions are *non-commuting* and neither is aborted, then their relative execution order must be the same at all sites. Consider for instance T_1 = "transfer balance to savings" and T_2 = "deposit €100." Both orders $T_1;T_2$ and $T_2;T_1$ make sense, but the result is clearly different. There must be a system-wide consensus on the order chosen.

Conflict Resolution and Reconciliation

Conflict resolution rewrites or aborts transactions to remove conflicts. Conflict resolution can be either manual or automatic. Manual conflict resolution simply allows conflicting transactions to proceed, thereby creating conflicting versions; it is up to the user to create a new, merged version.

Reconciliation detects and repairs conflicts, and combines non-conflicting updates. Thus transactions are *tentative*, i.e., a tentatively-successful transaction may have to roll back for reconciliation purposes. OR resolves conflicts *a posteriori* (whereas pessimistic approaches avoid them *a priori*).

In many systems, data invariants are either unknown or not communicated to the system. In this case, the system designer conservatively assumes that, if concurrent transactions access the same item, and one (or both) writes the item, then they are antagonistic. Then, one of them must abort, or both.

A few systems, such as Bayou [14] or IceCube [10] support an application-specific check of invariants.

Last Writer Wins

When transactions consist only of writes, a common approach is to ensure a global precedence order.

For instance, many replicated file systems follow the "Last Writer Wins" (LWW) approach. Files have timestamps that increase with successive versions. When the file system encounters two concurrent versions of the same file, it overwrites the one with the smallest timestamp with the "younger" one (highest timestamp). The write with the smallest timestamp is lost; this approach violates the Durability property of ACID.

Semantic Resolvers

A resolver is an application-specific conflict resolution program that automatically merges two conflicting versions of an item into a new one. For example, the Amazon online book store resolves problems with a user's "shopping cart" by taking the union of any concurrent instances. This maximizes availability despite network outages, crashes, and the user opening multiple sessions.

A resolver should ensure that the conflicting transactions are made to commute. In a state-based approach, a resolver generally parses the item's state into small, independent sub-items. Then it applies a LWW policy to updated and tombstoned sub-items, and a union policy to newly-created sub-items.

The most elaborate example exists in Bayou. A Bayou transaction has three components: the dependency check, the write, and the merge procedure. The former is a database query that checks for conflicts when replaying. The write (a SQL update) executes only if the consistency check succeeds. If it fails, the merge procedure (an arbitrary but deterministic program) provides a chance to fix the conflict. However, it is very difficult to write merge procedures in the general case.

Operational Transformation

In Operational Transformation (OT), conflicting operations are *transformed* [12]. Consider two users editing the shared text "abc". User 1 initiates *insert ("X",2)* resulting in "aXbc" and User 2 initiates *delete (3)*, resulting in "ab." When User 2 replays the insert, the result is "aXb" as expected. However for User 1 to observe the same result, the delete must be transformed to *delete(2)*.

In essence, the operations were specified in a non-commuting way, but transformation makes them commute. OT assumes that transformation is always possible. The OT literature focuses on a simple, linear, shared edit buffer data type, for which numerous transformation algorithms have been proposed.

OT requires two correctness conditions, often called TP1 and TP2. TP1 requires that, for any two concurrent operations A and B, running "A followed by {B transformed in the context of A}" yield the same

result as "B followed by {A transformed in the context of B}." TP1 is relatively easy to satisfy, and is sufficient if replay is somehow serialized.

TP2 requires that transformation functions themselves commute. TP2 is necessary if replay is in arbitrary order, e.g., in a peer-to-peer system. The vast majority of published non-serialized OT algorithms have been shown to violate TP2 [8].

Scheduling Transactions Content and Ordering

In order to capture any causal dependencies, transactions execute in happens-before order. As explained in Section "Conflicts," antagonistic transactions cause aborts, and non-commuting transactions must be mutually ordered. This so-called *serialization* requires a consensus.

Whereas pessimistic approaches serialize *a priori*, most OR systems execute transactions tentatively in arbitrary order and serialize a posteriori. Some executions are rolled back; cascading aborts may occur.

A prime example is the Bayou system [14]. Each site executes transactions in the order received. Eventually, the transactions reach a distinguished *primary* site. If a transaction fails its dependency check at the primary, then it aborts everywhere. Transactions that succeed commit, and are serialized in the execution order of the primary.

The IceCube system showed that it is possible to improve the user experience by scheduling operations intelligently [10]. IceCube is a middleware that relieves the application programmer from many of the complexities of reconciliation. Multiple applications may co-exist on top of IceCube. Applications expose semantic annotations, indicating which operation pairs commute or not, are antagonistic, dependent, or have an inherent semantic order. The user may create atomic groups of operations from different applications. The IceCube scheduler performs an optimization procedure over a batch of operations, minimizing the number of aborted operations. The user commits any of the alternative schedules proposed by the system.

Freshness of Replicas

Applications may benefit from *freshness* or quality-of-service guarantees, e.g., that no replica diverges by more than a known amount from the ideal, strongly-consistent state. Such guarantees come at the expense of decreased availability.

The Bayou system proposes qualitative "session guarantees" on the relative ordering of operations [13]. For instance, Read-Your-Writes (RYW) guarantees that a read observes the effect of a write by the same user, even if initiated at a different site. RYW ensures, that immediately after changing his password, a user can log in with the new password. Other similar guarantees are Monotonic-Reads, Writes-Follow-Reads, and Monotonic-Writes.

Systems such as TACT control replica divergence quantitatively [5]. TACT provides a time-based guarantee, allowing an item to remain stale for only a bounded amount of time. TACT implements this by pushing an update operation to remote replicas before the time limit elapses. TACT also provides "order bounding," i.e., limiting the number of uncommitted operations: when a site reaches a user-defined bound on the number of uncommitted operations, it stops accepting new ones.

Finally, TACT can bound the difference between numeric values.

For this, each replica is allocated a quota. Each site estimates the progress of other sites, using vector clock techniques. The site stops initiating operations once its cumulative modifications, or the estimated remote updates to the item, reach the quota. At that point the site pushes its updates and pulls remote operations. For example a bank account might be replicated at ten sites.

To guarantee that the balance observed is within €50 of the truth, each site's quota is €50/10 = €5. Whenever the difference estimated by a site reaches €5, it synchronises with the others.

Optimistic Replication Versus Optimistic Concurrency Control

The word "optimistic" has different, but related, meanings when used in the context of replication and of concurrency control.

Optimistic replication (OR) means that updates propagate lazily. There is no *a priori* total order of transactions. There is no point in time where different sites are guaranteed to have the same (or equivalent) state. Cascading aborts are possible.

Optimistic concurrency control (OCC) means that conflicting transactions are allowed to proceed concurrently. However, in most OCC implementations, a transaction validates before terminating. A transaction is serialized with respect to concurrent

transactions, at the latest when it terminates, and cascading aborts do not occur.

Key Applications

Usenet News pioneered the OR concept, allowing to share write-only information over a slow, but cheap network using dial-up modems over telephone lines.

Mobile users want to be able to work as usual, even when disconnected from the network. Thus, mobile computing is a key driver for OR applications. Systems designed for disconnected work that use OR include the Coda file system [6], the Bayou shared database [14], or the Lotus Notes collaborative suite.

Another important application area is Computer-Supported Collaborative Work. In this domain, users must be able to update shared artefacts in complex ways without interfering with one another. OR allows a user to insulate himself temporarily from other users. A key example is the Concurrent Versioning System (CVS), which enables collaborative authoring of computer programs [2]. Bayou and Lotus Notes, just cited, are also designed for collaborative work.

OR is used for high performance and high availability in large-scale web sites. A recent example is Amazon's "shopping cart," which is designed to be highly available, even if the same user connects to several instances of the Amazon store discussed earlier.

Cross-references

- ▶ Eventual Consistency
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer-to-Peer System
- ▶ Strong Consistency Models for Replicated Data
- ▶ Traditional Concurrency Control for Replicated Databases
- ▶ WAN Data Replication

Recommended Reading

1. Breitbart Y., Komondoor R., Rastogi R., and Seshadril S. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
2. Cederqvist P. et al. Version Management with CVS. Network Theory, Bristol, 2006.
3. Golding R.A. Weak-Consistency Group Communication and Membership. Ph.D. thesis, University of California, Santa Cruz, CA, USA, 1992, tech. Report no. UCSC-CRL-92-52. Available at ftp://ftp.cse.ucsc.edu/pub/tr/ucsc-crl-92-52.ps.Z
4. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
5. Haifeng Yu and Amin Vahdat. Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication. In Proc. 21st Int. Conf. on Distributed Computing Systems, USA.
6. Kistler J.J. and Satyanarayanan M. Disconnected operation in the Coda file system. ACM Trans. Comp. Syst., 10(5):3–25, 1992.
7. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7): 558–565, 1978.
8. Oster G., Urso P., Molli P., and Imine A. Proving correctness of transformation functions in collaborative editing systems. Rapport de recherche RR-5795, LORIA – INRIA Lorraine, 2005, Available at http://hal.inria.fr/inria-00071213/.
9. Petersen K., Spreitzer M.J., Terry D.B., Theimer M.M., and Demers A.J. Flexible update propagation for weakly consistent replication. In Proc. 16th ACM Symp. on Operating System Principles, 1997, pp. 288–301.
10. Preguiça N., Shapiro M., and Matheson C. Semantics-based reconciliation for collaborative and mobile environments. In Proc. Int. Conf. on Cooperative Inf. Syst., 2003, pp. 38–55.
11. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
12. Sun C. and Ellis C. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proc. Int. Conf. on Computer-Supported Cooperative Work, 1998, p. 59.
13. Terry D.B., Demers A.J., Petersen K., Spreitzer M.J., Theimer M.M., and Welch B.B. Session guarantees for weakly consistent replicated data. In Proc. Int. Conf. on Parallel and Distributed Information Systems, 1994, pp. 140–149.
14. Terry D.B., Theimer M.M., Petersen K., Demers A.J., Spreitzer M.J., and Hauser C.H. Managing update conficts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–182.
15. Wuu G.T.J. and Bernstein A.J. Efficient solutions to the replicated log and dictionary problems. In Proc. ACM SIGACT-SIGOPS 3rd Symp. on the Principles of Dist. Comp., 1984, pp. 233–242.

Optimization and Tuning in Data Warehouses

Ladjel Bellatreche
LISI/ENSMA–Poitiers University, Futuroscope Cedex, France

Definition

Optimization and tuning in data warehouses are the processes of selecting adequate optimization techniques in order to make queries and updates run faster and to maintain their performance by maximizing the use of data warehouse system resources. A data warehouse is

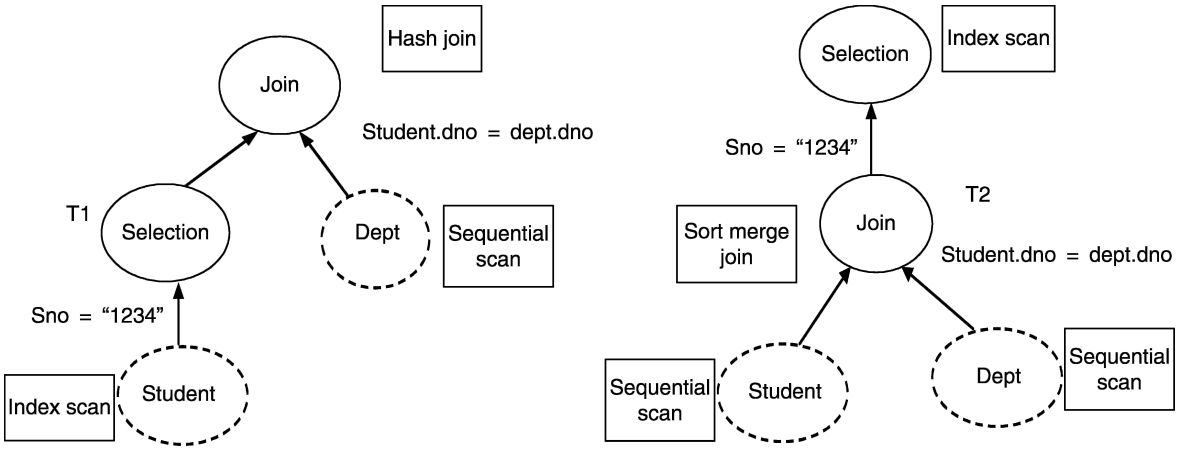
usually accessed by complex queries for key business operations. They must be completed in seconds not days. To continuously improve query performance, two main phases are required: physical design and tuning. In the first phase, data warehouse administrator selects optimization techniques such as materialized views, advanced index schemes, denormalization, vertical partitioning, horizontal partitioning and parallel processing. Generally, this selection is based on most frequently asked queries and typical updates. Physical design generates a configuration Δ containing a number of optimization techniques. This configuration should evolve, since data warehouse dynamically changes during its lifetime. These changes necessitate a tuning phase so as to keep the performance of the warehouse from degrading. Changes may be related to the content of tables, sizes of optimization structures selected during physical design (materialized views, indexes and partitions), frequencies of queries/updates, addition/deletion of queries/updates, etc. The role of the tuning phase is to monitor and to diagnose the use of configuration Δ and different resources assigned to Δ (like buffer, storage space, etc.). For instance, if an optimization technique, like an index, is not used by the whole workload, it will be dropped by a tuning tool and might be replaced with another technique.

Historical Background

There has been extensive work in query optimization since the early 1970s in traditional databases. Several algorithms and systems have been proposed, such as *System-R project*, and its ideas have been largely incorporated in many commercial optimizers. Note that each SQL query corresponding to a select-project-join query in the relational algebra may be represented by many query trees. The leaves of each query tree represent base relations and non-leaf nodes are algebraic operators like selections, projections, unions, joins. An intermediate node indicates the application of the corresponding operator on the relations generated by its children, the result of which is then sent further up. Thus, the edges of a tree represent data rows from bottom to top, i.e., from the leaves which correspond to data in the database, to the root, which is the final operator producing the query answer. For a complicated query, the number of all possible query trees may be very high due to many algebraic laws that hold for relational algebra: commutative and associative laws of joins, laws involving selection and projection push down along the tree, etc.

To choose an optimal query tree, a database optimizer may employ one of two optimization techniques: *rule-based optimization approach* and *cost-based optimization approach*. The rule-based optimizer is the oldest one. It is simple since it is based on a set of rules concerning join algorithms, the use of an index or not, the choice of the external relation in a nested loop, and so on. The optimizer chooses an execution plan based on the available access paths and their ranks. For instance, Oracle's ranking of the access paths is heuristic. If there is more than one way to execute a query, then the rule-based optimizer always uses the operation with the lower rank. Usually, operations of lower rank execute faster than those associated with constructs of higher rank. In cost-based optimization, the optimizer estimates the cost of each possible execution plan (Query execution plan is a set of steps used to access data in relational databases. Figure 1 gives an example of an execution plan, where dashed circles and solid circles represent base tables and algebraic operations, respectively. Other execution plans might be generated.) by applying heuristic formulas using a set of statistics concerning database (sizes of tables, indexes, tuple length, selectivity factors of join and selection predicates, sizes of intermediate results, etc.) and hardware (size of buffer, page size, etc.). For each execution plan, the query optimizer performs the following tasks: (i) it selects an order and grouping for associative-and-commutative operations like joins, unions and intersection, (ii) it chooses implementation algorithms for different algebraic operations: for example, selections may be implemented using either a sequential scan or an index scan; join operation may be implemented in different ways: *nested loop*, *sort-merge join* and *hash join* (see Fig. 1), (iii) it manages additional operators like group-by, sorting, etc., and (iv) it manages the intermediate results ($T1$ and $T2$ are an example of intermediate results of two execution plans of the same query in Fig. 1), etc. Cost-based optimizer chooses the plan that has the lowest cost using dynamic programming approaches (e.g., in System R).

Cost-based optimization is more effective than rule-based optimization, since all decisions taken on a query execution plan are validated by a cost model. An important point to be mentioned is that the quality of cost-based optimization depends strongly on the



Optimization and Tuning in Data Warehouses. Figure 1. Two different query execution plans.

recency of the statistics. Determining which statistics to create is a difficult task [6].

Due to the difficulty query optimizers have in selecting an optimal execution plan, some commercial database systems offer the data warehouse administrator to use *hints* in order to force query optimizer to choose an execution plan.

The above cited optimization techniques are enough to optimize traditional database applications (called, OLTP: On-Line Transaction Processing). It will be interesting to see whether they are also sufficient for decision support applications built around a large data warehouse.

A data warehouse is usually modeled with a relational schema (star schema, snow flake schema). A star schema consists of a single fact table that is related to multiple dimension tables via foreign key joins. Dimension tables are relatively small compared to the fact table and rarely updated. They are typically *denormalized* so as to minimize the number of join operations required to evaluate a query. Due to the interactive nature of decision support applications, having a fast query response time is a critical performance goal. The above optimization techniques are not suitable for data warehouse applications due to their different requirements and workload. Data warehouse applications operate in mostly-read environments, which are dominated by large and complex queries. The typical queries on the star schema are called *star join queries*. They are characterized by: (i) a *multi-table* join among a *large fact table* and dimension tables and (ii) each of the dimension tables involved in the join operation has *multiple* selection predicates (a selection predicate has the following form: $D_i.A_j\ \theta$ *value*, where A_j is an attribute of dimension table D_i and θ is one of six comparison operators $\{=,<,>,\leq,\geq\}$, and value is the predicate constant) on its descriptive attributes and (iii) no join operation between dimension tables. Unfortunately, conventional query optimization techniques are not efficient for star-join queries for two main reasons. First, traditional indexing techniques (B-tree) are not efficient to process such a selection predicate. This is due to the fact that dimension attributes often have low cardinality (e.g., gender) and each selection predicate typically has a low selectivity. Second, since the fact table is very large, computing the multi-table joins using traditional join algorithms (nested loop, sort-merge, hash joins) is inefficient because it requires scanning the fact table.

Without efficient optimization techniques, such queries may take hours or days, which is unacceptable in most cases. As a consequence, the physical design becomes sophisticated to cope with complex decision support queries [6]. To speed up these queries, in addition to the existing ones (developed for OLTP applications), a large spectrum of optimization techniques were proposed in the literature and mostly supported by commercial database systems. These techniques include materialized views, partitioning, advanced indexing schemes, denormalization, parallel processing. In the next section, all these techniques will be described in details. For each one, its principle, advantages, disadvantages and selection problem will be presented.

Foundations

The various optimization techniques selected during the physical design process may be classified into two main categories: redundant techniques and non-redundant techniques.

Redundant Techniques

This category includes four main techniques: materialized views, advanced indexing schemes, denormalization, and vertical partitioning.

1. *Materialized views.* A virtual view (A view is a derived relation defined in terms of base relations.) can be materialized by storing its tuples in the databases. Materialized views are used to precompute and to store aggregated data. They can also be used to precompute joins with or without aggregations. So, materialized views are suitable for queries with expensive joins or aggregations. Once materialized views are selected, all queries will be rewritten using materialized views (this process is known as *query rewriting*). A rewriting of a query Q using views is a query expression Q' referencing to these views. The query rewriting is done *transparently* by the query optimizer. To generate the best rewriting for a given query, a cost-based selection method is used. Two major problems related to materialized views are: (i) the view selection problem and (ii) the view maintenance problem.

Views selection problem. The database administrator cannot materialize all possible views, as he/she is constrained by some resources like, disk space, computation time, maintenance overhead and cost required for query rewriting process. Hence, he/she needs to select an appropriate set of views to materialize under some resource constraint. Formally, view selection problem (VSP) is defined as follows: given a set of most frequently used queries $Q = \{Q_1,Q_2,...,Q_n\}$, where each query Q_i has an access frequency f_i ($1 \leq i \leq n$) and a resource constraint M, the view selection problem consists of selecting a set of materialized views that minimizes one or more objectives, possibly subject to one or more constraints. Many variants of this problem have been studied: (i) minimizing the query processing cost subject to storage size constraint [5], (ii) minimizing query cost and maintenance cost subject to storage space constraint [9], (iii) minimizing query cost under a maintenance constraint [8], etc. This problem is known to be an NP-hard problem [8]. Several algorithms were proposed to deal with this problem [8,9]. The following table summarizes all possible formalizations of view selection problem. Two symbols are used in this table (√ and ?), where each one has its own interpretation: √: formalizations and selection algorithms already exist and ?: Inapplicable.

Objectives	*Constraints*		
	Without constraints	Maintenance cost	Storage cost
Query cost	√	√	√
Maintenance cost	√	?	√
Query cost & Maintenance Cost	√	?	√

View maintenance problem. Note that materialized views store data from base tables. In order to keep the views in the data warehouse up to date, it is necessary to maintain the materialized views in response to the changes at the base tables. This process of updating views is called view maintenance which has generated a great deal of interest. Views can be either recomputed from scratch, or incrementally maintained by propagating the base data changes onto the views. As recomputing the views can be prohibitively expensive, the incremental maintenance of views is of significant value [8].

2. *Indexing* has been at the foundation of performance tuning for databases for many years. A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns. An index can be either clustered or non-clustered. It can be defined on one table (or view) or many tables using a join index [10]. The traditional indexing strategies used in database systems do not work well in data warehousing environments since most OLTP queries are point queries. B-trees, which are used in most common relational database systems, are geared towards such point queries. In the data warehouse context, indexing refers to two different things: (i) indexing techniques and (ii) index selection problem.

Indexing techniques. A number of indexing strategies have been suggested for data warehouses: Value-List Index, Projection Index, Bitmap Index, Bit-Sliced Index, Data Index, Join Index, and Star Join Index. Bitmap index is probably the most important result

obtained in the data warehouse physical optimization field. The bitmap index is more suitable for low cardinality attributes since its size strictly depends on the *number of distinct values of the column on which it is built*. Besides disk space saving (due to their binary representation and potential compression), such index speeds up queries having Boolean operations (such as AND, OR and NOT) and COUNT operations. Bitmap join index is proposed to speed up join operations. In its simplest form, it can be defined as a bitmap index on a table R based on a single column of another table S, where S commonly joins with R in a specific way.

Index selection problem. The task of index selection is to automatically select an appropriate set of indices for a data warehouse (having a fact table and dimension tables) and a workload under resource constraints (storage, maintenance, etc.). It is challenging for the following reasons [3]: the size of a relational data warehouse schema may be large (many tables with several columns), and indices can be defined on a set of columns. Therefore, the search space of indices that are relevant to a workload can be very large [2]. To deal with this problem, most selection approaches use two main phases: (i) *generation of candidate attributes* and (ii) *selection of a final configuration*. The first phase prunes the search space of index selection problem, by eliminating *non-relevant* attributes. In the second phase, the final indices are selected using greedy algorithms [4], linear programming algorithms [3], etc. The *quality of the final set of indices depends essentially on the pruning phase*. To prune the search space of index candidates, many approaches were proposed [1–3], that can be classified into two categories: *heuristic enumeration-driven approaches* and *data mining driven approaches*.

In heuristic enumeration-driven approaches, heuristics are used. For instance, in [4], a greedy algorithm is proposed that uses optimizer cost of SQL Server to accept or reject a given configuration of indices. The weakness of this work is that it *imposes the number of generated candidates*. IBM DB2 Advisor is another example belonging to this category [15], where the query parser is used to pick up selection attributes used in workload queries. The generated candidates are obtained by a few *simple combinations* of selection attributes [15].

In data mining-driven approaches, the pruning process is done using data mining techniques, like in [2]. In this approaches the number of index candidates is not a priori known as in the first category. The basic idea is to generate frequent closed itemsets representing groups of attributes that could participate in selecting the final configuration of bitmap join indexes. A data mining based approach has been developed for selecting bitmap join indexes [2].

3. *Vertical partitioning* can be viewed as a redundant structure even if it results in little storage overhead. The vertical partitioning of a table T splits it into two or more tables, called, sub-tables or vertical fragments, each of which contains a subset of the columns in T. Note that the key columns are duplicated in each vertical fragment, to allow "reconstruction" of an original row in T. Since many queries access only a small subset of the columns in a table, vertical partitioning can reduce the amount of data that needs to be scanned to answer the query. Unlike horizontal partitioning, indexes or materialized views, in most of today's commercial database systems there is no native database definition language support for defining vertical partitions of a table [14].

To vertically partition a table with m non-primary keys, the number of possible fragments is equal to $B(m)$, which is the mth Bell number [12]. For large values of m, $B(m) \cong m^m$. For example, for m = 10; $B(10) \cong 115{,}975$. These values indicate that it is futile to attempt to obtain optimal solutions to the vertical partitioning problem. Many algorithms were proposed and classified into two categories: grouping and splitting [12]. Grouping starts by assigning each attribute to one fragment, and at each step, joins some of the fragments until some criteria is satisfied. Splitting starts with a table and decides on beneficial partitionings based on the query frequencies.

In the data warehousing environment, [7] proposed an approach for materializing views in vertical fragments, each including a subset of measures possibly taken from different cubes, aggregated on the same grouping set. This approach may unify two or more views into a single fragment.

4. *Denormalization* is the process of attempting to optimize the performance of a database by adding redundant data to save join operations. Denormalization is usually promoted in a data warehouse environment.

Non Redundant Techniques

In this category, two main techniques exist: horizontal partitioning and parallel processing.

1. *Horizontal partitioning* represents an important aspect of physical database design. It allows tables, indexes and materialized views to be partitioned into *disjoint* sets of rows that are physically stored and accessed separately [14] or in parallel. Horizontal partitioning may have a significant impact on performance of queries and manageability of very large data warehouses. Not only do data partitions reduce the time it takes to perform database maintenance and management tasks, by eliminating non-relevant partition(s), they also have a positive effect on the performance of applications. Another characteristic of horizontal partitioning is its ability to be combined with other optimization structures like indexes, materialized views. Splitting a table, a materialized view or an index into smaller pieces makes all operations on individual pieces much faster. Contrary to materialized views and indexes, data partitioning does not replicate data, thereby reducing space requirements and minimizing update overhead [13].

A native database definition language support is available for horizontal partitioning, where several fragmentation modes are available [11]: range, list and hash. In the range partitioning, an access path (table, view, and index) is decomposed according to a range of values of a given set of columns. The hash mode decomposes the data according to a hash function (provided by the system) applied to the values of the partitioning columns. The list partitioning splits a table according to the listed values of a column. These methods can be combined to generate composite partitioning (List-List, Range-Range, Hash-Hash, Range-List, ...). Recently, a new mode of horizontal partitioning became available in Oracle11g [11], called *virtual column-based partitioning*. It is defined by one of the above mentioned techniques and the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata.

Two versions of horizontal partitioning are available [12]: *primary* and *derived* horizontal partitioning. Primary horizontal partitioning of a table is performed using predicates defined on that relation. It can be performed using the different fragmentation modes above cited. Derived horizontal partitioning is the partitioning of a table that results from predicates defined in other table(s). The derived partitioning of a table R according to a fragmentation schema of table S is feasible if and only if there is a join link between R and S.

In the context of relational data warehouses, derived horizontal partitioning is well adapted. In other words, to partition a data warehouse, the best way is to *partition some/all dimension tables using their predicates, and then partition the fact table* based on the fragmentation schemas of dimension tables. This fragmentation takes into consideration requirements of star join queries (these queries impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands). To illustrate this fragmentation, suppose that a relational warehouse is modeled by a star schema with d dimension tables and a fact table F. Among these dimension tables, g tables are fragmented ($g \leq d$). Each dimension table D_i ($1 \leq i \leq g$) is partitioned into m_i fragments: $\{D_{i1}, D_{i2}, ..., D_{im_i}\}$, where each fragment D_{ij} is defined as: $D_{ij} = \sigma_{cl_j^i}(D_i)$, where cl_j^i and σ ($1 \leq i \leq g$, $1 \leq j \leq m_i$) represent a conjunction of simple predicates and the selection operator, respectively. Thus, the fragmentation schema of the fact table F is defined as follows: $F_i = F \ltimes D_{1j} \ltimes D_{2k} \ltimes ... \ltimes D_{gl}$ ($1 \leq i \leq m_i$), where $\ltimes$ represents the semijoin operation.

Derived horizontal partitioning has two main advantages in relational data warehouses, in addition to classical benefits of data partitioning: (i) precomputing joins between fact table and dimension tables participating in the fragmentation process of the fact table [1] and (ii) optimizing selections defined on dimension tables. Similar advantages hold for bitmap join indexes.

2. *Parallel Processing* Data partitioning is always coupled with parallel processing. To design a parallel data warehouse, two main issues must be addressed: data partitioning and data placement (allocation). Data placement is a key factor for high performance parallel data warehouses. Determining an effective data placement is a complex administration problem depending on many parameters including system architecture, database and workload characteristics, hardware configuration, etc. The easier way to design a parallel data warehouse is to first partition dimension tables using the primary horizontal partitioning and then derived partition the fact table. This partitioning alternative generates a set of sub-star schemas. In order to ensure a high performance of complex queries, these sub-star schemas shall be allocated to various machines in efficient manner.

Tuning

Before talking about tuning phase, a summarization of physical design is required to understand the need for tuning.

The first point concerns the different formalizations of problems of selecting optimization techniques in physical design phase. They are mostly based on a set of most frequently asked queries, *a priori known*. However, in dynamic environments, like data warehousing with various ad-hoc queries, it is difficult to identify potential useful optimization structures in advance. The second point is about the similarities between optimization techniques: materialized views and indexes, bitmap join indexes and derived horizontal partitioning. These similarities are not always taken into account during physical design phase. This situation may incur the following limitations: (i) non-consideration of the mutual interdependencies between optimization structures (sometimes it is better to select more materialized views than indexes and vice-versa or replacing bitmap join indexes by a non redundant technique like derived horizontal partitioning to reduce maintenance overhead) gives sub-optimal solutions, (ii) absence of metrics for efficient distribution of storage space between redundant optimization techniques and (iii) redistribution of space among optimization structures after update operations.

Based on the above points, tuning tools are recommended since they supervise the good use of different optimization techniques selected during physical design phase. Tuning tools might be triggered when user requirements evolve (new queries/updates, not considering some existing queries), query frequencies change, sizes of tables, materialized views, indexes and partitions increase, etc. To keep data warehouse applications running at high performance, several aspects of physical design should be tuned: buffer pool, allocation of working memory, materialized views, indexes, storage space, horizontal partitioning, vertical partitioning, data placement, etc.

During the data warehouse life cycle, some structures may be added/dropped (e.g., materialized views and indexes), merged (indexes and horizontal partitions) or splitted (e.g., horizontal partitions). For instance, some commercial database systems provide monitoring tools observing the good utilization of indexes. If an index is not used by a workload, it will be dropped. Its storage space might be used for creating another optimization technique. Merging operations deal mainly with indexes and horizontal partitions. They are crucial for data warehouse applications [5,14]. This is because optimization structures are often either too large (for redundant structures) to fit in the available storage, or cause updates to slow down significantly. They are supported by most commercial database systems.

Many commercial database systems offer tools for physical design tuning: "What-If" analysis tool of SQL Server used to facilitate manual tuning. SQL Server proposes a tuning using a relation-based approach. The optimizer can replace a large useful index with smaller, less useful ones. For example, operations that required a single traversal through a complex index may be implemented as the intersection of two traversals through simple indexes. Other transformations include index merging (implementing an index scan of relation B as a full scan through relation A), prefixing (building an index on a; b instead of a; b; c), and the removal of structures. DB2 design advisor tool provides integrated recommendations for indexes, materialized views, shared-nothing partitioning and multidimensional clustering. ORACLE 10G takes as input a workload and a set of optimization candidates for that workload (these candidates are generated by Oracle Automatic Tuning Optimizer) and provides a recommendation for the overall workload.

In academic research work, a tuning tool called AutoPart which combines horizontal and vertical table partitioning to reduce I/O costs for each query by eliminating unnecessary accesses to non-relevant data is proposed [13]. AutoPart recommends the combination of partitioning with a small set of key indexes. A similar work proposed to combine derived horizontal partitioning with bitmap join indexes [1].

Key Applications

The proposed techniques within this paper can be applied in any database applications having the same characteristics (huge tables, complex queries with many join operations and restriction) and requirements of data warehouse (response time). Scientific and statistical database applications are a good example. Historically, the main techniques explored in this paper were proposed and supported in decision support applications. Materialized views and advanced

indexing schemes could be easily applied in traditional OLTP applications, when update operations are not important. Horizontal partitioning can also be applied, but moderately. The choice of partitioning attributes is a crucial performance issue. For example, if a database is partitioned based on changing value attributes like Age, the database will be faced with the problem of *instance migration*.

Future Directions

As mentioned in the previous section, physical design and tuning are very crucial decisions for the performance of data warehouse applications. In this section, some of the interesting open issues for physical design and tuning are highlighted:

1. Multi-objective algorithms for indexes: most index selection algorithms have one objective function which represents the query processing cost subject to one constraint representing the storage cost. It will be interesting to propose multi-objective algorithms for selecting indexes. Such formalizations will reduce the query processing and maintenance cost (which is not negligible for indexes). Since there is a strong similarity between materialized views and indexes, an easier way to deal with this problem is to adapt multi-objective algorithms for materialized views to indexes.
2. Incorporating query rewriting using materialized views in selection process: after selection of materialized views, all queries will be rewritten using them. Choosing an optimal rewriting is a difficult problem. It will be interesting to combine the problem of selecting materialized views and the problem of rewriting queries. A simple combination may involve the formalization of materialized view selection subject to time requiring for query rewrite process.
3. Pruning search space of materialized view selection problem: selecting materialized views is an NP-hard problem. To prune search space of this problem, vertical and horizontal partitioning (primary and derived) might be used, because a materialized view may involve selection, projection, join operations.
4. Partition allocation over table spaces: assigning different fragments generated by horizontal partitioning process over various table spaces may be a crucial issue for performance of queries. This problem does not get enough attention from data warehouse research community. This problem is quite similar to data placement studied in distributed and parallel databases areas. It will be interesting to adapt the existing algorithms. Most of these algorithms are static. Tuning of data placement will be recommended since partition usages change, partition might be merged/splitted, etc.
5. Supporting derived horizontal partitioning: Today's commercial database systems support derived horizontal partitioning, where a table is decomposed based on the fragmentation schema of only one table, using referential partitioning [11]. In real data warehouse applications, a fact table may be derived partitioned based on fragmentation schemas of several dimension tables in order to satisfy star join queries requirements. From an industry perspective, this situation is quite unsatisfactory and requires further thought.

Cross-references

- ▶ Bitmap-Based Index Structures for Multidimensional Data
- ▶ Data Partitioning
- ▶ Index Join Physical Schema Design
- ▶ Query Rewriting Using Views
- ▶ Semijoin
- ▶ View Maintenance in Data Warehouses
- ▶ Virtual Partitioning

Recommended Reading

1. Bellatreche L., Boukhalfa K., and Mohania M.K. Pruning search space of physical database design. In Proc. 18th Int. Conf. Database and Expert Syst. Appl. 2007, pp. 479–488.
2. Bellatreche L., Missaoui R., Necir H., and Drias H. Selection and pruning algorithms for bitmap index selection problem using data mining. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 2007, pp. 221–230.
3. Chaudhuri S. Index selection for databases: a hardness study and a principled heuristic solution. IEEE Trans. Knowl. Data Eng., 16 (11):1313–1323, 2004.
4. Chaudhuri S. and Narasayya V. An efficient cost-driven index selection tool for microsoft sql server. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 146–155.
5. Chaudhuri S. and Narasayya V. Index merging. In Proc. 15th Int. Conf. on Data Engineering. 1999, pp. 296–303.
6. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
7. Golfarelli M., Maniezzo V., and Rizzi S. Materialization of fragmented views in multidimensional databases. Data & Knowl. Eng., 49(3):325–351, June 2004.
8. Gupta H. Selection and maintenance of views in a data warehouse. Ph.D. Thesis, Stanford University, September 1999.

9. Lawrence M. Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses. In Proc. The Genetic and Evolutionary Computation Conf., 2006, pp. 699–706.
10. O'Neil P. and Quass D. Improved query performance with variant indexes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 38–49.
11. Oracle Data Sheet. Oracle partitioning. White Paper: http://www.oracle.com/technology/products/bi/db/11g/, 2007
12. Özsu M.T. and Valduriez P. Principles of distributed database systems. Second edition. Prentice Hall, Englewood Cliffs, NJ, 1999.
13. Papadomanolakis S. and Ailamaki A. Autopart: automating schema design for large scientific databases using data partitioning. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, pp. 383–392.
14. Sanjay A., Narasayya V.R., and Yang B. Integrating vertical and horizontal partitioning into automated physical database design. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 359–370.
15. Valentin G., Zuliani M., Zilio D.C., Lohman G.M., and Skelley A. Db2 advisor: an optimizer smart enough to recommend its own indexes. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 101–110.

Optimization of DAG-Structured Query Evaluation Plans

► Multi-Query Optimization

Optimization of Parallel Query Plans

► Parallel Query Optimization

OQL

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Synonyms

Object query language

Definition

OQL was developed to play the role of SQL for *Object-Oriented Databases*, especially those adhering to the *ODMG Standard* [4] where the language is defined. Unlike SQL, OQL is a functional language, and its operators can be composed to an arbitrary level of nesting within a query provided the query remains type-correct. Fegaras and Maier [8] have shown how OQL expressions have a direct translation into monoid *Comprehensions*.

Optimisation techniques for OQL that exploit its inherent functional nature are discussed in [5,6,8]. OQL has been influential in the development of the SQL3 standard and also the functional core of the XQuery language for XML. Thus optimisation techniques developed for OQL are also applicable to these languages.

Key Points

The fundamental modelling concept of *object identifiers* for entity instances was accepted into the database mainstream in the late 1980s, and the move to using SQL-like syntax for querying such data models followed soon after. Early influential systems were OSQL [2] and *AMOSQL* (q.v.). This resulted in query language proposals for object-oriented databases such as the very influential O2 query language [1] and its successor OQL, which was included in the ODMG Standard [4]. For example, the DAPLEX query

```
FOR EACH S IN STUDENT
  SUCH THAT name(S)="Fred Jones"
     PRINT name(S), age(S);
```

is expressed as follows in OQL, basically by syntactic reordering of the query clauses and using path expressions rather than function application:

```
SELECT S.name, S.age FROM STUDENT S
WHERE S.name="Fred Jones"
```

When restricted to sets, monoid comprehensions are equivalent to set monad comprehensions [3], which capture precisely the nested relational algebra [8]. Most OQL expressions have a direct translation into the monoid calculus. For example, the OQL query

```
SELECT DISTINCT HOTEL.price
FROM HOTEL IN(
  SELECT h
  FROM c IN CITIES, h IN c.hotels
  WHERE c.name="Arlington")
WHERE EXIST r IN HOTEL.rooms:r.bed_num=3
AND HOTEL.name IN (
  SELECT t.name
  FROM s IN STATES, t IN s.attractions
  WHERE s.name = "Texas" );
```

finds the prices of hotels in Arlington that have rooms with three beds and are also are named after a tourist attraction in Texas. This query is translated into the following monoid comprehension [7]:

```
fold(Union,Empty,
  [ price(h) | c <- Cities; h <- hotels
  (c); name(c) = ''Arlington'';
       fold(Or,False,
       [bednum(r)=3 | r <- rooms(h) ]),
       fold(Or,False,
  [ name(h)=name(t) | s <- States; t <-
  attractions(s); name(s)=''Texas'']))]
```

Here, as in Functional Programming

```
fold(Or,False,[x1,x2, ... xn]) = x1
Or x2 Or ... xn Or False
```

computes the logical *Or* of a list of boolean values, so it is true only if *some* of them are true. Likewise *fold(Union,Empty,L)* copies the list L into a set without duplicates. Mathematically *fold* implements *monoid* operations with a given *merge* operation and a *zero*.

Cross-references

► AMOSQL
► Comprehensions
► Functional Query Language

Recommended Reading

1. Bancilhon F., Delobel C., and Kanellakis P.C. Building an Object-Oriented Database System, The Story of O2. Morgan Kaufmann, Los Altos, CA, 1992.
2. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, In Proc. 1st Int. Conf. on Extending Database Technology. 1988, pp. 251–270.
3. Buneman P., Libkin L., Suciu D., Tannen V., and Wong L. Comprehension syntax. ACM SIGMOD Rec., 23(1):87–96, 1994.
4. Cattell R.G.G. (ed.). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.
5. Cluet S. and Delobel C. A general framework for the optimization of object-oriented queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1992, pp. 383–392.
6. Fegaras L. Query unnesting in object-oriented databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1998, pp. 49–60.
7. Fegaras L. Query Processing and Optimization in λ-DB. In The Functional Approach to Data Management, Chapter 13. P.M.D., Gray L., Kerschberg P.J.H., and King A. (eds.). Springer, Berlin, 2004.
8. Fegaras L. and Maier D. Towards an effective calculus for Object Query Languages. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1995, pp. 47–58.

ORA-SS Data Model

► Object Relationship Attribute Data Model for Semi-structured Data

ORA-SS Schema Diagram

► Object Relationship Attribute Data Model for Semi-structured Data

Orchestration

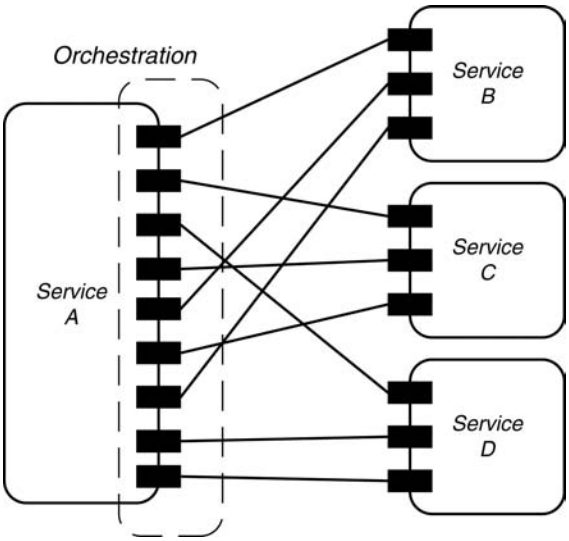
W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, The Netherlands

Definition

In a Service Oriented Architecture (SOA) services are interacting by exchanging messages, i.e., by combining services more complex services are created. Orchestration is concerned with the composition of such services seen from the viewpoint of single service.

Key Points

The terms "orchestration" and "choreography" describe two aspects of integrating services to create business processes [2,3]. The two terms overlap somewhat and the distinction is subject to discussion. Orchestration and choreography can be seen as different "perspectives." Choreography is concerned with the exchange of messages between those services and is often be characterized by analogy "Dancers dance following a global scenario without a single point of control." Orchestration is concerned with the interactions of a single service with its environment. Here an analogy can also be used. In orchestration, there is



Orchestration. Figure 1. Orchestration.

someone, "the conductor", who tells everybody in the orchestra what to do and makes sure they all play in sync.

Figure 1 illustrates the notion of orchestration. Service A is interacting with other services to create a more complex service. The dashed area shows the focal point of orchestration, i.e., the control-flow related to message exchanges of a single party. Languages such a BPEL are proposed to model and enact such orchestrations [1]. Note that languages like BPEL are very close to traditional workflow languages, i.e., the same types of control-flow patterns need to be supported.

Orchestration often assumes that services have a "buy side" and a "sell side," i.e., services can be used by other services ("sell side") and at the same time use services ("buy side"). Orchestration is mainly concerned with the "buy side." Unlike choreography, there is a single party coordinating the process.

Cross-references

- ▶ BPEL
- ▶ Business Process Management
- ▶ Orchestration
- ▶ Web Services
- ▶ Workflow Management

Recommended Reading

1. Alves A., Arkin A., Askary S., Barreto C., Bloch B., Curbera F., Ford M., Goland Y., Guzar A., Kartha N., Liu C.K., Khalaf R., Koenig D., Marin M., Mehta V., Thatte S., Rijn D., Yendluri P., and Yiu A. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.
2. Dumas M., van der Aalst W.M.P., and ter Hofstede A.H.M. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley, New York, 2005.
3. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, Berlin, 2007.

ORDB (Object-Relational Database)

- ▶ Object Data Models

Order Item

- ▶ Clinical Order

Order Statistics

- ▶ Quantiles on Streams

Ordering

- ▶ Similarity and Ranking Operations

Orientation Relationships

- ▶ Cardinal Direction Relationships

Oriented Clustering

- ▶ Subspace Clustering Techniques

Origin

- ▶ Provenance
- ▶ Provenance in Scientific Databases

OR-Join

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

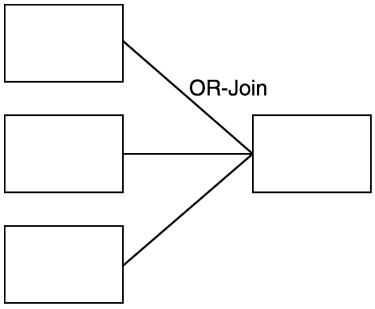
Synchronous join

Definition

The point of convergence within a workflow following alternative, mutually exclusive paths.

Key Points

An OR-Join (Fig. 1) represents a point within a workflow where two or more alternative workflow branches re-converge following an OR-Split into a single common activity as the next step within the workflow. In contrast with an AND-Join, no parallel activity execution has occurred at the join point, therefore no synchronization is required. With an OR-Join a thread of control may arrive at the specific activity via any of several alternative preceding activities.



OR-Join. Figure 1. OR-Join.

Cross-references

- ▶ Join
- ▶ OR-Split
- ▶ Process Life Cycle
- ▶ Workflow Management and Workflow Management System

OR-Split

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

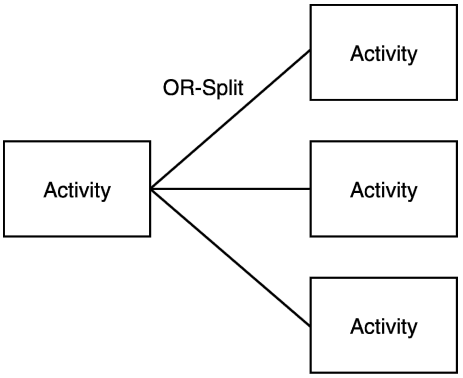
Conditional branching; Conditional routing; Switch; Branch

Definition

A point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.

Key Points

An OR-Split (Fig. 1) establishes alternative and mutually exclusive workflow branches. For example, in a mortgage application, different paths may represent different branches based on conditional logic, such as credit risk or the amount to be barrowed. As paths are mutually exclusive, no parallel execution of activities occur and thus no synchronization is required, so the workflow branching converges with an OR-Join rather than an AND-Join. An OR-Split is conditional and the (single) specific transition to next activity is selected according to the outcome of the Transition Condition(s).



OR-Split. Figure 1. OR-Split.

Cross-references

- ▶ AND-Join
- ▶ OR-Join

► Process Life Cycle
► Split
► Workflow Management and Workflow Management System

OSD

► Network Attached Secure Device

OSQL

Tore Risch
Uppsala University, Uppsala, Sweden

Definition

OSQL [1,2] is an functional query language and data model similar to Daplex, first implemented in the Iris DBMS [4]. The data model of OSQL is object oriented with three kinds of system entities: objects, types, and functions. A database consists of a set of objects, the objects are classified into types, and functions define the semantics of types. The data model is similar to an ER model with the difference that both entity relationships and attributes are represented as functions and that (multiple) inheritance among entity types is supported. OSQL provide object identifiers (OIDs) as first class objects, and, unlike Daplex, queries can return OIDs in results. Queries are expressed using a SELECT syntax similar to SQL. Derived functions are also defined using select statements similar to functions in SQL-2003.

Key Points

With the OSQL data model a database consists of a set of objects. The objects are classified into subsets by types and each type has an extent consisting of the objects belonging to that type. Type inheritance is supported with the type named OBJECT as most general type. The extent of a type is a subset of the extents of its supertype(s). For example if entity type STUDENT is a subtype of type PERSON then the extent of type STUDENT is also a subset of the extent PERSON. Types are defined dynamically using a CREATE TYPE statement, e.g.,:

```
CREATE TYPE STUDENT SUBTYPE OF PERSON;
```

Functions define relationships among entities and properties of entities. Functions can be stored in the databases, derived in terms of other functions, or be defined as foreign functions implemented in some conventional programming language. Stored functions correspond to tables in relational databases, and derived functions are parameterized views similar to function definitions in SQL-2003.

Queries and derived functions are defined declaratively using a SELECT statement, e.g.,:

```
SELECT NAME(P)
FOR EACH PERSON P
WHERE AGE(P)>20 AND SEX(P) =
''Female'';

CREATE FUNCTION GRANDPARENTS(PERSON
P)-> PERSON
AS SELECT PARENT(PARENT(P));
```

Queries are expressed as constraints over extents. Functions composition allows easy traversal of relationships between entity types. As in Daplex, if a function returns a set of objects (e.g., PARENT) functions applied on it iterate over the elements of the set. This is a form of extended path expressions through function composition.

OSQL was implemented in the Iris DBMS [4] and HP's OpenODB product. The Amos II DBMS [3] uses a modified OSQL language, AmosQL.

Cross-references

► AmosQL
► Daplex
► Functional Data Model

Recommended Reading

1. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, In Proc. 1st Int. Conf. on Extending Database Technology. 1988, pp. 251–270.
2. Fishman D.H., Beech D., Cate H.P., Chow E.C., Connors T., Davis J.W., Derrett N., Hoch C.G., Kent W., Lyngbaek P., Mahbod B., Neimat M.A., Ryan T.A., and Shan Iris M.C. An Object-Oriented Database Management System, ACM Trans. Off. Inf. Syst., 5(1):48–69, 1987.

3. Risch T., Josifovski V., and Katchaounov T. Functional data integration in a distributed mediator system. In Functional Approach to Data Management – Modeling, Analyzing and Integrating Heterogeneous Data, P. Gray, L. Kerschberg, P. King, A. Poulovassilis (eds.). Springer, Berlin, 2003.
4. Wilkinson K., Lyngbaek P., and Hasan W. The iris architecture and implementation, IEEE Trans. Knowl. Data Eng., 2(1):63–75, 1990.

Overlay Network

Wojciech Galuba, Sarunas Girdzijauskas
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Definition

An *overlay network* is a communication network constructed on top of an another communication network. The nodes of the overlay network are interconnected with logical connections, which form an *overlay topology*. Two overlay nodes may be connected with a logical connection despite being several hops apart in the underlying network. Overlay networks may define their own *overlay address space* which is used for efficient message routing in the overlay topology.

Key Points

When a distributed application is deployed in a computer network, the individual nodes on which the application is running need to be able to discover and communicate with one another. A solution to this problem is the overlay network. The overlay network interconnects all the application nodes and provides the basic communication primitives such as flooding, random walks or point-to-point overlay message routing and multicast.

Overlay networks are typically deployed on top of the Internet and by far the most common usage is in peer-to-peer systems. For example, Gnutella, an early peer-to-peer file sharing system connects all the peers in an overlay network, each peer shares its files, and files are searched for using query flooding in the overlay network.

Overlay network topologies can be divided into two broad classes: unstructured and structured. Unstructured overlay networks do not construct a globally consistent topology, instead peers choose their neighbor sets independently and in a largely ad-hoc way. In unstructured overlay networks nodes reach the other nodes by message flooding or random walks. Structured overlays define an address space and each of the overlay nodes has a unique address. The addresses are used to construct an overlay topology that enables efficient and scalable messages passing between the overlay nodes. In most of the modern structured overlays the expected number of routing hops scales as $O(log\ N)$ with the network size. Distributed Hash Tables are a specific case of structured overlay networks. Apart from structured and unstructured there also exist hybrid overlays.

Overlay networks are designed to be robust to *churn*, i.e., arrivals and departures of the overlay network nodes to and from the network. As overlay network nodes loose their overlay topology connections, new connections have to be added in their place. In structured overlay networks the additional challenge is to maintain the overlay topology such that the overlay routing remains efficient, i.e., the routing paths are kept short.

Cross-references

▶ Distributed Hash Table
▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
▶ Peer-to-Peer System

OWL: Web Ontology Language

Sean Bechhofer
University of Manchester, Manchester, UK

Synonyms

Web ontology language

Definition

The Web Ontology Language OWL is a language for defining ontologies on the Web. An OWL Ontology describes a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects. OWL ontologies can be used to describe the properties of Web resources. Where earlier representation languages have been used to develop tools and ontologies for specific user-communities in areas such as sciences, health and e-commerce, they were not necessarily designed to be compatible with the

World Wide Web, or more specifically the Semantic Web, as is the case with OWL.

Features of OWL are a collection of expressive operators for concept description including boolean operators (intersection, union and complement), plus explicit quantifiers for properties and relationships; the ability to specify characteristics of properties, such as transitivity or domains and ranges; a well defined semantics facilitating the use of inference and automated reasoning; the use of URIs for naming concepts and ontologies; a mechanism for importing external ontologies; and compatability with the architecture of the World Wide Web, in particular other representation languages such as RDF and RDF Schema.

OWL consists of a suite of World Wide Web Consortium (W3C) Recommendations – six documents published in February 2004 describe Use Cases and Requirements, an Overview of the language, a Guide, Reference, OWL Semantics and a collection of Test Cases [3].

Key Points

Ontology languages allow the representation of ontologies. An ontology "defines a set of representational primitives with which to model a domain of knowledge or discourse" (see Ontology). The definition of an ontology can encompass a wide range of artefacts, from simple word lists, through taxonomies, thesauri and rich logic-based models and there are a corresponding range of languages for their representation.

Standardization of representation languages is a cornerstone of the Semantic Web effort. A standard representation facilitates interoperation – in particular, well-defined, unambiguous *semantics* ensure that applications can agree on the meaning of expressions. OWL is intended to provide that standard representation.

OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes. The design of the language was influenced by a number of factors. Description Logics, Frame-based modeling paradigms, and Web languages RDF and RDF Schema were key inputs, as was earlier work on languages such as OIL and DAML+OIL.

Knowledge Representation in a Web setting introduces particular requirements such as the distribution across many systems; scalability to Web size; compatibility with Web standards for accessibility and internationalization; and openness and extensibility. OWL uses URIs for naming and extends the description framework for the Web provided by RDF to address some of the issues above.

OWL defines three sublanguages: OWL Lite, OWL DL and OWL Full. OWL Full is essentially RDF extended with additional vocabulary, with no restrictions on the way in which that vocabulary is used. OWL DL places restrictions on the way in which the vocabulary can be used in order to define a language for which a number of key reasoning tasks (for example concept satisfiability or subsumption) are decidable. OWL Lite further restricts the expressivity allowed – for example, explicit union or complement are disallowed in OWL Lite. OWL DL and OWL Lite have a model theoretic semantics that corresponds to a Description Logic (DL) [1] and thus facilitate the use of DL reasoners to provide reasoning support for the language [2].

The design of representation languages often involves trade-offs, and there are limitations on what can be expressed using OWL, in particular in OWL-DL. These limitations have been selected primarily to ensure that these language subsets are well-behaved computationally, with decidable procedures for concept satisfiability. For example, OWL does not provide support for general purpose rules, which are seen as an important paradigm in knowledge representation, for example in expert systems or deductive databases. Extensions to OWL are being proposed to cover, among others, rules, query, additional expressivity, metamodeling and fuzzy reasoning.

Cross-references

- ▶ Description Logics
- ▶ Ontology
- ▶ RDF
- ▶ RDF Schema
- ▶ Semantic Web

Recommended Reading

1. Baader F., Calvanese D., McGuinness D.L., Nardi D., and Patel-Schneider P.F. (eds.). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge, UK, 2003.
2. Horrocks I., Patel-Schneider P.F., and van Harmelen F. From *SHIQ* and RDF to OWL: the making of a web ontology language. J. Web Semant., 1(1):7–26, 2003.
3. World Wide Web Consortium. Web Ontology Language (OWL). W3C Recommendation. Available at: http://www.w3.org/2004/OWL/.