
Vancouver, British Columbia, Canada
5 – 10 October 1992



# Tutorials—

## Concepts of Object-Oriented Programming

Raimund K. Ege, Florida International University

This tutorial defines and teaches the basic concepts of object-oriented programming, illustrates the advantages of object-oriented techniques over conventional programming, introduces the components of an object-oriented programming environment, and gives an overview of the features of object-oriented languages and environments. This tutorial will let you make an informed decision about what language/environment will best serve your programming needs.

## Object Design for Modularity, Reuse and Quality

Douglas Bennett, Design Ways

Turning buzzwords like modularity, reusable components, extensibility, testability, and robustness into reality requires more than just a compiler for an object-oriented language, or even an integrated development environment. These properties must be "designed into" the software product. This tutorial shows how to make design decisions so the product will do what its users want and so the buzzwords actually show up in the product. The tutorial will work through the steps of a design project, documenting user and producer needs, modeling the "thing" objects, describing behavior with event response models, and, finally, developing an architecture for the product. Part of each step is evaluating the design against the magic words. The result will be a product structure that is probably very different from conventional software architectures, but one that can be measured against the design criteria.

This tutorial teaches the explicit design of software. It uses notations from several existing methods. You may use your own notations, if you wish.

## A Comparison of Object-Oriented Analysis and Design Methods

Martin Fowler

This tutorial shows how to look inside design and analysis methods to see how they differ, and more importantly, how they are the same. It examines several of the most well-known methods, including Booch, Coad/Yourdon, Odell/Martin, Rumbaugh,

and Schlaer/Mellor, but its analysis techniques can be applied to any method.

In general, each method is made up of several techniques (e.g. ER modeling, state transition diagrams) each of which emphasizes a particular aspect of the system. These techniques can be classified as being for structural (data), behavioral or architectural modeling. The tutorial shows the techniques each method uses and how different methods use different dialects of the same techniques, varying the notation and introducing new concepts. It uses examples to show how the same system is expressed in different ways by the various methods. The tutorial also compares the design approach advocated by each method.

This tutorial will help you decide which method will work best for you. In addition, you will learn how ideas from different methods can be combined to better suit the system under analysis.

## Integrating Analysis and Design Methods

Derek Coleman & Paul Jeremaes, HP Labs, England

Most of those who practice object-oriented analysis and design do not follow any standard method exactly, but combine different techniques to suit their own unique requirements. Each method employs its own set of models, notations, and processes, so it can be difficult to combine them. This tutorial shows how to design a method by providing a framework for understanding and evaluating current methods, applying it to three recent methods (OMT (Rumbaugh et al), Responsibility Driven Design/CRC cards, Booch 91 Method), and combining them to produce a new method, FUSION, that builds on their best aspects.

The tutorial introduces a set of criteria for evaluating methods from the viewpoint of the object-oriented concepts they support, the kinds of models and notations they employ and the process steps that they recommend. The criteria provides a way to understand the underlying similarities and differences between methods.

The criteria were developed as part of a program to assess what kind of method should be made available to HP engineers, which led to the development of the FUSION method. The tutorial ends with a brief account of the usefulness of FUSION in practice.

## Object-Oriented Concurrent Programming

Jean-Pierre Briot, Institute Blaise Pascal, France

This tutorial treats object-oriented concurrent programming (OOCP) as the natural generalization of object-oriented programming. OOCP decomposes a large program into a collection of small modules that run and interact concurrently and are capable of exploiting parallel hardware. The tutorial describes various levels of integration between object-oriented programming and concurrency, leading to the notion of an active object, which unifies object and activity, message passing and synchronization.

The tutorial introduces Concurrent Smalltalk to describe concepts, constructs, and methodology. Examples include programming with continuations, divide and conquer, and pipelining. It also shows how to implement active objects, and uses Actalk as an example. Finally, it compares various OOP models and languages, with a special focus on the Actor computation model.

Although the tutorial uses Smalltalk for examples, you don't need to know Smalltalk to understand it; a quick introduction to Smalltalk syntax is included. It assumes that you understand object-oriented programming well, but little more about concurrency than intuitive concepts of processes and synchronization.

## Types for the Working Programmer

Andrew Black, DEC

There has been recent progress in understanding types for object-oriented languages, but most of it has had little or no impact on real programmers.
This tutorial aims to extract from the confusion those topics that are important to programmers who must use or choose an object-oriented language. The tutorial explains the role of types in object-oriented languages, what types can do for programmers, and how the trend towards distributed and heterogeneous systems and object-oriented databases influences what it means for a program to be type-correct.

The tutorial describes the difference between objects and values, what types are and why they are good for you, what refinement and subtyping are, what problems they solve, and what problems they do not; why contravariance isn't an unnatural act; why inheritance is a relationship between programs, not between classes.

By the end of the tutorial, you will understand how abstract concepts like subtyping can help you solve practical problems such as deciding when one piece of code can be substituted for another.

## Types for the Language Designer

Michael Schwartzbach & Jens Palsberg,
Aarhus University, Denmark

The type systems of object-oriented languages have specific goals: they serve as partial documentation, they provide modularity, and they ensure safety and efficiency at run-time. There are many choices in the design of a type system, and it is hard to evaluate their tradeoffs. This tutorial teaches a coherent theory of type systems for object-oriented languages that can be used for both explicit systems (where the programmer supplies type annotations) and implicit systems (where the compiler must perform type inference), and forty systems based on interfaces, classes, and sets of classes.

The tutorial defines an idealized object-oriented language, inspired by Smalltalk, and develops several type systems for it. This uniform framework makes it easy to compare different approaches. The tutorial explores the limitations of static type checking and shows how dynamic type checking can be introduced. It demonstrates how subclassing is different from subtyping, how specification types differ from implementation types, and the influence of a type system on separate compilation.

## Simulation with DEVS-CLOS

Suleyman Sevinc, University of Sydney , Australia

This tutorial has two purposes: to show how the features of CLOS affect a design and to describe the design of a general-purpose simulation system. The tutorial is designed for those who want to understand the practical importance of the unique features of CLOS and also those who would like to learn how to use or build object-oriented simulation systems.

The tutorial emphasizes the underlying model of object-oriented programming in CLOS and distinguishes it from other models. It also describes the requirements of a simulation system and the design of DEVS-CLOS, a publicly available simulation system. DEVS-CLOS is an extension to CLOS that supports modeling and simulation. It supports visualization, supports hierarchical design of simulations, allows first-order logic in models, and supports adaptive system simulation. The tutorial shows how a system like DEVS-CLOS can be used to solve typical simulation problems, and how CLOS concepts like multimethods, multiple inheritance, pre- and post-methods and dispatching algorithms were used in the design of DEVS-CLOS.

## Advanced CLOS and Meta-Object Protocols

Jon L. White, Lucid

One of the important characteristics of CLOS is its dynamic flexibility to change descriptions, programs, and even data objects "on the fly." A typical CLOS system is implemented by a set of meta-objects, which can be changed by the programmer, essentially letting you create a new language. Thus, CLOS is a reflective programming language.

The main purpose of this tutorial is to teach you how to put a meta-object protocol to a practical use. In addition to describing the implementation of a typical CLOS system and discussing the various issues raised by the book "Art of the Meta-Object Protocol," it gives practical examples of meta-object extensions including techniques for making user-defined metaclasses, for making alternations to SLOT-VALUE so that certain slots can be "persistent," and for making metaclasses whose classes automatically keep a table of all their instances.

The tutorial is aimed at the CLOS programmer or the programming language expert who would like to learn more about reflective programming using a meta-object protocol. The emphasis is on practical uses of a meta-object protocol, not on the philosophy of reflection.

## Introduction to Object-Oriented Database Management Systems

David Maier, Oregon Graduate Institute

This tutorial begins by explaining object-oriented database management systems (OODBMS) in terms of what is the value added beyond record-oriented database systems and object-oriented programming languages. It presents most of the current commercial OODBMS and several advanced prototypes with particular attention to distinguishing them in their data models, application interfaces and system architectures. It will also contrast the OODBMS approach with extended relational systems. A goal of the tutorial is to give participants an appreciation for the consequences of design choices made in the different systems. It concludes with a critique of current market offerings, and suggests there are significant regions of the design space to explore, and needs of advanced applications that are still largely unmet by any database product.

This tutorial is designed both for those considering investing in OODBs and those who just want to understand the technology. It assumes knowledge of object-oriented programming concepts, and some familiarity with conventional database systems, particularly relational databases.

## Object-Oriented Software Development with the Demeter Method

Karl Lieberherr, Northeastern University

The Demeter Method is a formal method that lifts object-oriented software development to a higher level of abstraction by using a graphical specification language for describing object-oriented programs. Executable programs are automatically generated by a CASE tool (the Demeter System/C++) from the graphical high-level descriptions. Unlike other specification languages, the Demeter Method allows you to keep the binding of methods to classes flexible under changing class structures. The higher level of abstraction leads to shorter and more reusable programs than by programming directly in one of today's object-oriented languages such as C++, Smalltalk, Eiffel or CLOS.

Because the Demeter Method is regularly taught at NU both at the graduate and undergraduate level with the 10 week quarter system, it has become, by necessity, very easy to learn.

This tutorial is for professionals who want to learn powerful, formally defined high-level concepts that describe the programming task in terms of data-model-based graphs and their subgraphs.

## Object-Oriented User Interfaces

Dave Collins, IBM

The principles of objects, polymorphism, classes, and inheritance can apply to the end user's *external* view of a user interface, just as it can apply to the language that is used to implement it. This tutorial shows how to design (not implement) a user interface that is truly object-oriented. It gives guidance on the design of the "externals" of object-oriented user interfaces, and shows how developers can capitalize on the isomorphism between the user's conceptual model of the interface and the constructs provided by object-oriented programming languages. The tutorial exposes a (perhaps surprisingly) deep analogy between object-oriented programming languages and object-oriented user interfaces. This analogy is valuable because it can be used to map "external" user interface design in a clear and natural way onto object-oriented application frameworks. The tutorial is illustrated with historical examples, many on videotape.

## Evaluating Reusable Class Libraries

Timothy Korson, Clemson University

Object-oriented technology not only affects the way we design individual applications, it holds the potential of ushering in the "software industrial revolution." A key to the success of object technology is the widespread availability of high quality class libraries. These class libraries are the re-usable parts of the software industry.

This tutorial teaches how to evaluate object-oriented libraries. The same criteria can be used for selecting commercial libraries or for designing libraries for use in-house. The tutorial also describes practical criteria for specifying and cataloging in-house corporate software libraries. In addition to these technical criteria, the tutorial also shows you the corporate infrastructure that is necessary for enabling large scale reuse.

## Object Engineering

R. Stonewall Ballard,
Component Software Corporation

Converting an object-oriented design into class definitions in an object-oriented language often requires many engineering decisions. It is easy to make these decisions incorrectly for languages that give the programmer a lot of control over data representation, such as C++. This tutorial covers low-level issues in building extensible and evolvable programs, issues that are important if you want to build well-balanced abstractions. It describes how to use multiple inheritance properly in spite of the compiler, when to use forwarding instead of inheritance, how to encapsulate state, and when to use references instead of copying objects.

This tutorial teaches how to convert object-oriented designs into code that is easy to extend and reuse. Many of the issues are more interesting to C++ programmers than Smalltalk programmers, and C++ is the language used for examples, but most of the issues are language independent.

## Teaching Object-Oriented Programming and Design

James C. McKim Jr., Hartford Graduate Center

A course in object-oriented programming and design should address the claims made for the object-oriented paradigm, namely that it promotes reuse, models the problem space, facilitates maintenance,

incorporates changes easily, and shortens the development lifecycle. One way (perhaps the only way) for students to test such claims is to build a small but high quality product as part of the course.

This tutorial shows how to have students in a conventional computer science program build such a product, and addresses such issues as how to pick good projects, whether and how students should work together in teams, how to keep students on schedule, and how to divide a project into a sequence of deliverables within the context of a one semester course.

The tutorial also describes how to scale back the approach so that students in short courses , common in industry, can get maximum benefit in the limited time that is allowed.

## Object-Oriented Project Management

Kenneth Rubin & Adele Goldberg, ParcPlace

Object-oriented projects have to be managed properly to obtain the most benefits from object-oriented technology. This tutorial explains the effect of object-oriented technology on costs, staffing and choice of methodology. It describes the pitfalls that attend object-oriented projects so that you can avoid them, and explains the key processes, including development style, use of prototyping, and reuse. It gives specific guidelines for organizing projects, managing a reuse library, estimating the size and cost of projects, and introducing object-oriented technology into an organization.

## Introduction to Object-Oriented Design

Lori Stipp & Grady Booch, Rational

This tutorial describes the Booch method for object-oriented design, including details of the notation and the design process. It describes the principles that are necessary for thinking and abstracting in terms of classes and objects. The tutorial includes a number of examples, including "war stories" from specific projects. It also covers extensions to the notation beyond that described in *Object-Oriented Design with Applications.*

This tutorial is a condensed version of Booch's popular tutorial given at past OOPSLA's. If you have attended that in the past then you should attend tutorial 18 instead of this one.

## The Pragmatics of Building Object-Oriented Systems

Grady Booch, Rational

This tutorial expands upon the process described in Booch's *Object-Oriented Design with Applications* to provide a prescriptive, iterative process for object-oriented development. It describes both the macro and micro process of development. It gives guidelines for managing iterative development, and addresses the pragmatic issues of milestones and planning, staffing, integration and release management, reuse, testing, quality assurance and metrics, documentation, tools, and technology transfer.

This tutorial assumes that you are familiar with Booch's design method.

## A Case Study of Domain Analysis: Health Care

Martin Fowler & Thomas Cairns,
St Mary's Hospital Medical School , England

Three years ago the UK National Health Service decided to develop a generic model to describe all aspects of its management and operation. Several projects were launched to describe the health care process using the object-oriented modeling technique Ptech (now published by James Martin and Jim

Odell). One of these projects, the Cosmos project, produced the Cosmos Clinical Process Model (or CCPM), which describes the medical record. The CCPM contains approximately 70 object types in a highly abstract structure describing clinical procedures, observations, evidence and assessment, accountabilities and contracts, care planning, and clinical knowledge. The model has gained widespread interest in both the UK and Europe and will be used as the focus for a number of EEC sponsored projects, and is taking a leading standardization role in Europe.

The CCPM is an example of the result of domain analysis. Its use of abstractions, some of which are applicable outside health care, and its use of an operational/knowledge divide can guide those working on large-scale generic models in other areas. The tutorial describes the relationship between generic and specific models since these issues are key in the acceptance of a highly generic application area model by those with specific applications to develop. The question of adapting an object-oriented method for particular needs is also addressed.

## The Analysis and Design of Distributed Systems

Mehmet Aksit, University of Twente

The design of distributed object-oriented systems involves a number of considerations that rarely arise in sequential object-oriented design or in non-object-oriented languages. The tutorial describes analysis and design techniques for data abstraction, inheritance, delegation, persistence, atomicity, concurrency, synchronization, and coordinated behavior in a distributed object-oriented framework. Special attention will be paid to the uniform integration of these concepts with the object-oriented paradigm. Discussions will be accompanied by examples that arose from constructing such systems.

## Specification Techniques for Object-Oriented Software

Mahesh H. Dodani, University of Iowa

There are many techniques for specifying software. However, they are not usually presented in a way that is directly applicable to object-oriented software. This tutorial surveys both operational and descriptive specification techniques, shows how to use them in several popular object-oriented software engineering methods, and provides case studies of developing useful, formal specification mechanisms that are appropriate for object-oriented software engineering.

The tutorial first describes criteria for choosing specification mechanisms from both a theoretical (syntax and semantics, specifying properties, reasoning about properties, verification) and practical (ease of use, modularity, applicability, supporting tools) perspective. These criteria are then used to survey several popular operational (data flow diagrams, finite state machines, and petri nets) and descriptive (entity-relationship models, logic, and algebraic specifications) specification mechanisms. Finally it shows how to extend two of these techniques (algebraic specifications and finite state machines) so that they are suitable for object-oriented software engineering.

## Writing Efficient C++ Programs

Scott Meyers, Brown University

This tutorial teaches the competing meanings of "high performance;" the characteristics of object-oriented systems that can decrease performance; how to locate and eliminate computational bottlenecks in C++ programs; and the trade-offs between high performance and system reusability, maintainability, and portability. It describes in detail the factors that affect the performance of C++ software. Whether your primary concern is high system speed, small system size, fast recompilation, or a combination of these, this tutorial provides you with the tools necessary to come up with an appropriate object-oriented design, to implement it efficiently in C++, and to fine-tune it for maximum performance.

The reasons for bottlenecks in C++ programs are often surprising. Contrary to popular belief, virtual functions usually exact a negligible performance cost, while unexpected calls to constructors and destructors frequently hamstring applications. This tutorial teaches you what is really important if you want to deliver high performance, and the techniques you need to achieve it.

## The Design and Management of C++ Class Libraries

Arthuir Riel, Vanguard Training

Producing reusable C++ class libraries takes more than just knowing the language: it requires careful design. This tutorial shows how to create truly reusable C++ class libraries. It includes how to design a minimal public interface, variable-size objects, memory leakage, heuristics for operator overloading, designing reusable base-classes, using inheritance and containment, and deciding how to place classes in a class library.

## Hardware Support for Object-Oriented Systems

Mario Wolczko, University of Manchester, England

When the performance penalty of object-oriented systems is mentioned, a common response is to blame antiquated hardware designs for not supporting object-oriented languages as they deserve. To what extent can the performance gap between conventional languages and object-oriented languages be closed using hardware? What architectural changes benefit object-oriented systems, and by how much?

There have been many attempts to make hardware that better supports object-oriented programming. This tutorial describes these systems, and the extent that they have succeeded or failed in their aims. These systems include SOAR, Rekursiv and MUSHROOM, as well as some features from mainstream architectures such as SPARC. Issues covered include: choice of instruction set, design of the memory system including caches and virtual memory hardware, scaleability, use of parallelism, and hardware/software trade-offs.

A common misconception is that if something is implemented in hardware then it must be fast. One aspect of the tutorial is to show that there are limits to what hardware support can achieve. However, better hardware can reduce the cost of some language features, such as dynamic binding, and can make a system more scaleable.

## Efficient Implementation of Object-Oriented Programming Languages

Craig Chambers, University of Washington
David Ungar, Sun Labs

This tutorial is for those who like to know how object-oriented languages work "under the hood," why some things are expensive while other things are cheap, and why language definitions include certain restrictions. This tutorial provides that, and more. It describes features of object-oriented languages that are difficult to implement efficiently, and how this has affected language designs. It describes state-of-the-practice and state-of-the-art techniques for implementing languages like C++ and Smalltalk.

Language features include dynamic binding and generic operations, inheritance, user-defined control structures, static type systems, multiple inheritance and virtual base classes, and mixing object-oriented and non-object-oriented programming.

The tutorial also addresses questions like: What are the trade-offs in using the various implementation techniques? What problems remain that block more efficient object-oriented language implementations? What might be promising areas for future research?

## Constraint-Based Languages and Systems

Bjorn Freeman-Benson , University of Victoria, Canada
Alan Borning, University of Washington

A constraint is a relation that should be satisfied, for example, that a line remain horizontal, that a resistor in an electrical circuit simulation obey Ohm's Law, or that the height of a bar in a bar chart be proportional to some number in an application program. Constraints have been used in a variety of languages and systems, particularly in user interface tool kits, in planning and scheduling, and in simulation. They provide an intuitive declarative style of programming that integrates well with object-oriented systems.

This tutorial teaches what constraints are, how to use them in applications such as user interfaces, how to implement them (including how to implement constraint hierarchies), and how to embed them in object-oriented and logic programming languages. You don't have to know anything about constraints, but it would be helpful to have a strong background in programming languages.

## Visual Programming Languages from an Object-Oriented Perspective

Allen L. Ambler, University of Kansas
Margaret M. Burnett,
Michigan Technical University

Visual programming language research has evolved greatly since its early days. At first, attempts at visual programming mostly took the form of flowchart-like diagrams. But in recent years, a wide number of innovative approaches have been incorporated into visual languages, including object-oriented programming, form-based programming, programming by demonstration, and dataflow programming. Unfortunately, while many of these systems represent important ideas, only a few have been successful as complete visual programming languages. This tutorial explains why this is true, and describes ways in which the problem can be addressed.

This tutorial explores the issues behind the successes and failures of earlier approaches from a design perspective. It identifies characteristics of successful visual programming languages, and explains how to design an object-oriented language that maintains those characteristics. It shows solutions to a number of problems by looking at existing visual programming languages, including Prograph.

## Intermediate Smalltalk: Practical Design and Implementation

Trygve Reenskaug, TASKON , Norway

Careful design before programming is as important in Smalltalk as in other languages. This tutorial describes techniques that have proven useful in commercial development of large Smalltalk systems. The tutorial shows how to develop a language independent design and then go to detailed programming and testing. The examples will be based on Smalltalk-80 and use the release 4.1 graphical user interface framework.

## Writing Efficient Smalltalk Programs

Ken Auer, Knowledge Systems Corp.

Smalltalk has a reputation for being slow and a memory hog. In reality, the Smalltalk environment offers many ways to create inefficient code, and programmers often exploit those opportunities.

However, there are just as many ways to create efficient code, some of which may not be available in more traditional languages. Whether an application is efficient or not has more to do with the way the programmer has used the available tools than the tools themselves.

This tutorial teaches Smalltalk programmers how to write efficient programs. The emphasis is on how to exploit, rather than sacrifice, the benefits of good object-oriented design. You should have at least six months experience with Smalltalk writing non-toy programs to get the most benefit from the tutorial.

## Testing Object-Oriented Software

Edward Berard, Berard Software Engineering

Testing of object-oriented software is just as important as testing non-object-oriented software, but the process is fundamentally different because of such factors as information hiding, encapsulation, and inheritance. This tutorial teaches you the terms and concepts of testing software in general, and object-oriented software in particular; how to apply a number of different testing techniques to object-oriented software; how to construct test cases; and an appreciation of what is involved in planning a successful software testing effort. The testing techniques that are covered include both white-box testing such as basis path testing and coverage testing, and black-box testing techniques such as equivalence class partitioning and boundary value analysis.

This course is designed for those with experience in object-oriented software engineering who would like to follow a more rigorous approach to testing.

## Object-Oriented Geometry and Graphics

Jan Krc-Jediny & Augustin Mrazik, ArtInAppleS, Czechoslovakia

Spatial information is an important part of geographic information systems, CAD and CAM systems, and user interfaces for visualization of any kind of information. The object-oriented approach to spatial object modelling results in much more understandable designs than non-object-oriented approaches.

This tutorial describes the design of several key components of a system for dealing with spatial information which are: analytical geometry closed with respect to union, intersection and complement operations; hierarchically structured objects for maintaining topological information; object identity and problems arising from change of spatial objects in the database and user interface, transactions and copies of objects; dependencies of spatial objects and their copies in the MVC user-interface architecture.

You should have experience in building user-interfaces and should have some background in analytical geometry. Examples are shown in Smalltalk, so a basic knowledge of Smalltalk would be helpful.

## The Design of an Object-Oriented Operating System: A Case Study of Choices

Roy H. Campbell & Nayeem Islam,
University of Illinois
Peter W. Madany, SUN Microsystems

This tutorial describes the object-oriented design of a complete operating system, written to be object-oriented, with a user and application interface that is object-oriented. The main objective is to illustrate object-oriented design trade-offs by studying a large object-oriented system, the Choices operating system.

Choices is an object-oriented multiprocessor operating system that runs native on SPARC stations, Encore Multimaxes, and IBM PCs. The system is built from a number of frameworks that implement a general file system, persistent store for persistent objects, process switching, parallel processing, distributed processing, interrupt

handling, virtual memory, networking, and interprocess communication.

If you bring an IBM/PC 386-based portable computer running MS-DOS to the course then you may experiment by writing application programs for PC-Choices. All participants will receive a copy of PC-Choices on a floppy.

Participants should have experience with building object-oriented systems and have a basic understanding of operating systems design. Reading knowledge of C++ is helpful, but not necessary.

## Teaching "Object-Think" with Multi-Sensory Engagement

Peter Coad, Object International, Inc.

This is a tutorial about how to teach the "big picture" of objects, especially in an industrial environment where time is short. It is based on the whole brain theory and multi-sensory engagement. The tutorial shows over twenty specific techniques for teaching using multi-sensory engagement, many of these unified into "The Object Game," which lets those new to objects see and manipulate them. The tutorial describes a set of specific techniques for leading someone into more effective "object think," and compares and contrasts whole-brain multi-sensory engagement with other "object think" techniques, such as graphics, languages, and CRC cards.

This tutorial is designed for those trying to teach object-oriented development who have been frustrated by the inability of some people to reach effective "object think." It will also be of interest to those planning to teach object-oriented programming, or if you are responsible for selecting such a course.

## Schema Updates for Object-Oriented Database Systems

Roberto Zicari , Johann Wolfgang Goethe University, Germany

OODBMS are often used for complex data whose structure is likely to change over time, yet the problem of schema updates has not been completely solved by any commercial or research OODBMS. This tutorial describes the schema modification problem and why it is important, what is really offered by products, what a good solution would be like, and whether we are likely to see it soon.

The tutorial reviews several commercial OODBMS products that provide facilities for updating the schema, namely: GemStone (Servio Corporation), ITASCA(Itasca), O2(O2Technology), ObjectStore (ObjectDesign), Ontos (Ontologic) and Statice (Symbolics). It also describes the solutions proposed in some experimental research prototypes and the open problems that remain.