Object-Oriented DBMS and Beyond

Klaus R. Dittrich and Andreas Geppert

Department of Computer Science, University of Zurich
{dittrich,geppert}ifi.unizh.ch

Abstract. Over the past 10+ years, object-oriented database systems have gone a long way from research prototypes to commercial products to real-life mission-critical applications. Currently, we also witness the extension of relational systems with salient object features, resulting in so-called object-relational DBMS.
In this paper, we introduce and review the salient features of both approaches, discuss their merits and shortcomings, and for which kinds of applications they are best suited. We also elaborate on further necessary improvements of the current state of the art. Furthermore, we will speculate about several upcoming areas of database research in a broad sense (like global information systems, workflow management, component technology) where object-orientation and object-relational and object-oriented database systems in particular might (and should!) play a leading role.

1 Introduction

Database management systems have a long tradition of research and successful deployment in computing practice. The "classical" systems (hierarchical, network, and relational) are established and viable platforms for data management in a large variety of application areas. With the advent of object-orientation and new requirements imposed by advanced application domains, research has proposed object-oriented database systems beginning in the mid-80s. Since the beginning of the 90s, such systems have been commercially available. After a period of discussion (some may even say "religious warfare") about whether object-oriented DBMSs are actually needed or whether relational systems can meet these new requirements just as well, relational vendors have recently reacted by incorporating object-oriented features into their systems. These extensions are also considered in the forthcoming SQL3-standard.

In this paper, we review object-oriented concepts in database systems and hypothesize about future trends in database systems with respect to object-orientation. Particularly, we argue that database technology will play an increasingly important role in new and emerging areas, such as workflow management and information integration. These areas definitely need database services, and object-oriented database technology is well-suited to meet their requirements such as modeling power and seamless integration. Nevertheless, database functionality as required by these areas can no longer be sufficiently provided by single, monolithic systems; we therefore hypothesize that, in order to be successful, database systems will have to adapt a component-oriented view that allows to bundle services into systems dedicated to the intended application domain.

The remainder of this paper is structured as follows. First, we review and survey object-oriented concepts in database systems. We speculate about future trends in section 3 and the impact of object-orientation and related paradigms (e.g., component technology) on database applications and information systems in section 4. Section 5 identifies challenges and open issues, and section 6 concludes the paper.

2 Object-orientation and Database Systems

Object-oriented database systems try to blend the object-oriented paradigm with database technology. We therefore shortly introduce both areas and then present object-oriented database systems.

2.1 Object-Orientation

The general characteristics of the object-oriented paradigm [24] are objects and object identity, classes, and inheritance/specialization. They should apply to all models, systems, methods, etc. that claim the label of being object-oriented.

An *object* is a concept to represent a real or virtual entity; each object has a state and exhibits a certain kind of behavior. In addition, each object has an identity that distinguishes it from all other objects. Complex object structures can be built by using references among objects.

Classes define the structure and behavior of objects. A class therefore prescribes an intensional set of objects that are similarly structured and exhibit the same behavior. Individual objects can be constructed by instantiating a class. In this way, a class acts as an object factory, and the set of instances of a class existing at a certain point of time is called the *class extension*.

Similarities between (objects of multiple) classes can be expressed by *specialization*. The semantics of this association is *inclusion polymorphism* [5]: an object can be an instance of multiple classes, i.e., each instance of a more specific class (the subclass) is also an instance of the general class (the superclass). Furthermore, an instance of a (direct or indirect) subclass can be used wherever an instance of a general class is required (*substitutability*). Specialization is related to *inheritance* in that all the properties (structure and behavior) defined for the superclass also apply to the subclass—the subclass inherits structure and behavior from the superclass. Subclasses can extend the inherited class definition or can override parts of it.

Nowadays object-orientation is a widely used paradigm, e.g., for user interfaces, system modeling and design (e.g., UML [28]), and object-oriented programming languages are in widespread use (e.g., C++ or Java).

2.2 Database Systems

Database management systems [12] (DBMSs) are software systems for the long-term, reliable, and persistent storage of and concurrent access by multiple users to large sets of data items. *Databases* are collections of data items whose structure is defined by a (database) *schema*. A *data model* specifies data definition constructs (i.e., restricts the set of definable schemas) as well as operators that can be applied to data items constructed under the schema. Ultimately, a *database system* (DBS) is a DBMS together with a concrete database (including its schema).

Popular data models include the hierarchical, network, and relational model. In the relational data model [8], all data items are represented as relations (tables) whose elements are tuples, which in turn are sets of attributes. The popularity of the relational model is mainly due to its query language SQL, which allows to retrieve sets of tuples in a declarative manner. In terms of number of installations, relational DBMSs (RDBMS) are currently by far the most successful DBMSs in the marketplace.

2.3 Object-Oriented Database Systems

During the last decade, a major part of database research has focused on *object-oriented database management systems* (OODBMSs) to combine database technology with object-orientation. Motivation behind this trend are mainly the impedance mismatch between object-oriented languages and RDBMSs and requirements of then new applications such as CAD/CAM, software engineering environments, etc. In the meanwhile, several products are commercially available and occupy a still small but growing market share.

An OODBMS is *a DBMS with an object-oriented data model* [1, 10]. The first requirement refers to the usual database functionality, i.e., persistent data storage, management of large sets of data, secondary storage management, transaction management, and a query language are indispensable. This means that an object-oriented programming language with (only) support for persistent objects would not classify as an OODBMS.

The requirement to support an object-oriented data model [1] refer to:

- objects with identity,
- complex objects,
- classes,
- encapsulation,
- inheritance and specialization,
- computational completeness (the data manipulation language should allow to express any computable algorithm),
- overriding, overloading, and late binding,
- extensibility (classes should be user-definable).

OODBMS-products are available commercially since the beginning of the 90s. These systems have in the meantime overcome several of their initial weaknesses (e.g., additional functionality such as schema modification, better performance, etc.). Also, a standard has been defined [6] so that customers are now less dependent on single products.

2.4 Object-Relational Database Systems

Object-relational DBMSs (ORDBMSs) are DBMSs whose data model extends the relational model [8] with object-oriented features. The object-relational extensions [22, 23] are part of the SQL3-standard which is currently under specification by ANSI and ISO. Note that apart from introducing object-oriented concepts, SQL3 defines several other extensions; however, we focus on the object-relational ones.

The object-relational model still considers relations/tables as the fundamental sort of data modeling construct, i.e., every real-world entity to be stored in a database must be represented by a tuple of the appropriate relation. The most important object-oriented extensions refer to support for tuple types, tuple identifiers, abstract data types (ADTs), and inheritance

An ADT is characterized by an (optional) list of supertypes, an internal representation, comparison operators, a constructor, and further type-specific operators. Tuple types (*row types*) can be defined separately from relations. A row type defines a set of attributes, where references to other tuples (of any type), collection-valued types, and atomic types are permitted as attribute types. Tuple types can inherit from other tuple types. Tuples types can be assigned to relations upon relation creation. For each relation, an attribute can be designated as tuple identifier in that the values of this attribute are specified to be system-generated. Relations can be defined as specializations (i.e., subsets) of other relations, with the semantics that each tuple of the specialized relation is also an element of the more general relation.

Although the SQL3-standard is still under definition, vendors have already started to include object-relational features into their products. Illustra (now available as Informix Universal Server) [35] supports user-definable types and inheritance. Additionally, nonstandard functionality can be packaged into so-called *DataBlade modules* that can be linked to a DBMS. Each such module can be regarded as a library of types. Along with each type, functions, operators, and access methods of the type are given. Examples for DataBlade modules include geometric data types, images, and time series. Extensions similar to Data Blades are supported by Oracle8 (cartridges) and DB2 (extenders [20]).

Most relational and object-relational DBMSs also support constraints and/or triggers. *Triggers* or *event-condition-action rules* (ECA-rules) [11] allow to specify situations in which the DBMS has to react automatically by executing a pre-defined action. The semantics of such a rule is that whenever the event occurs and the condition holds, then the action is executed. Possible event types are updates of a tuple or object, conditions are typically specified as queries (with the semantics that the condition holds if the query returns a non-empty result), and actions are specified in the data manipulation language of the DBMS. ECA-rules are useful constructs to implement a variety of tasks, including consistency constraints and materialized view maintenance.

2.5 Strengths and Weaknesses

When identifying the strengths and shortcomings of relational, object-oriented, and object-relational DBMSs for specific types of applications, meaningful criteria include

1. whether real-world structures can be modeled (more or less) naturally,
2. whether there is a design method how to use the respective type of DBMS,
3. whether the DBMS is efficient for typical applications.

For a classification of strengths of the various DBMS-types, the DBMS-matrix [35] has been proposed using complexity of data structures and the need for complex queries as the two dimensions. Following this matrix,

- RDBMSs are well-suited for simple data structures and queries,
- OODBMSs can handle complex data structures, but do not support queries that well,
- ORDBMSs are well-suited for applications characterized by complex data structures and complex queries.

From our point of view, this classification is too schematic. For instance, the ODMG-standard also specifies a query language (OQL) which is for quite some time already implemented in some OODBMS-products (e.g., O_2 [2]). Thus, given that the data models of ORDBMSs and OODBMSs are quite similar, in principle OODBMSs might handle queries equally well as ORDBMSs. Moreover, the classification seems to assume that ORDBMSs can increase the modeling power of RDBMSs (which is then comparable to that of OODBMSs) and at the same time preserve the performance of query processing as present in RDBMSs.

The strength of OODBMSs is that they are well-suited for complex object structures and that they integrate very smoothly with object-oriented programming languages. Complex structures can be more or less directly modeled in the database, and there is no need to convert data structures in application programs to/from simple-structured tuples. In other words, the so-called *impedance mismatch* vanishes. This is particularly important for applications written in object-oriented languages; they can map their objects directly onto database objects. To some extent, the impedance mismatch will continue to exist in ORDBMSs, as queries will return sets of tuples which then have to be converted into programming language objects. Due to complex ADTs and type-specific behavior definable for them, the impedance mismatch will however become smaller in ORDBMSs than in RDBMSs.

Versions and *configurations* [21] represent causal and/or historical developments of objects which for instance are needed in design environments (e.g., CASE). A versioning concept can be nicely integrated into an object-oriented model, so OODBMSs are a natural starting point for adding version support. Many OODBMS-products already support some sort of version or configuration functionality, along with design transactions.

A limitation of current OODBMS-products—but not of OODBMS-technology in general—seems to be that they are less adequate for the management of huge amounts of data (in the range of many giga or even terabytes). Such amounts of data are typically encountered in data warehousing [7] or data mining applications. The critical factor here is efficiency of query processing. RDBMSs are seen at an advantage, because they can rely on a much simpler model as well as on much more experience and research in query processing and optimization than OODBMSs. Moreover, SQL can be easily parallelized [9], which is still an open issue for query languages of OODBMSs.

Another area in which RDBMSs rely on sound methods and a bulk of experiences (again, as a consequence of the data model simplicity) is *database design*. Both, the notation and the process are well understood, while this is not really the case for OODBSs (see [34] for a survey). Ideally, a design method and notation supports the conceptual database design and allows to transform the conceptual schema into the logical schema of the used database system. In other words, an equivalent to the entity-relationship-model for the relational case is needed for OODBS-design. Such a design method should be an extension or at least be compatible with existing design methods, such as UML [28].

Another weakness of OODBMSs is that they provide only very limited support for specifying and enforcing consistency constraints (OODBMSs following the principle of persistence by reachability ensure referential integrity, and the ODMG-standard allows to define user-defined keys). Similar to the relational case, consistency constraints can be implemented as ECA-rules [16]; however, since OODBMSs do not (yet) support ECA-rules, explicit constraints mostly have to be enforced by method implementations and application programs.

Summarizing, OODBMSs are a natural choice for applications that need to manage complex object structures and for object-oriented applications that need to avoid the impedance mismatch. It is still to be seen whether they are feasible platforms for emerging application classes such as data warehousing and data mining. Concerning ORDBMSs, we feel that they are still too young in order to assess whether they can live up to their claims. In particular, it remains to be seen whether ORDBMSs can extend their modeling power while preserving the query processing performance of RDBMSs. In fact, we expect that—as data structures become as complex as those managed by OODBMSs—complexity of query processing will be the same in OODBMSs and ORDBMSs.

2.6 And the Winner is...

object-orientation. The advent of ORDBMSs shows that relational vendors accept the the need for object-oriented concepts in databases. Thus, the DBMS of the future will incorporate object-oriented concepts, regardless whether its origin is relational or "pure" object-orientation.

It remains to be seen to which extent and how smoothly ORDBMSs can incorporate object-oriented extensions. Extending an RDBMS to an ORDBMS might imply to re-design major parts of the DBMS, and it is likely that ORDBMS-builders will encounter the same problems and difficulties as OODBMS-designers did. Particularly, efficient object storage and management and efficient query processing and optimization will turn out as major hurdles. Users will also expect performance to remain the same for "traditional" relational data and queries, and it will be interesting to see whether object-relational extensions will hamper the performance of conventional queries.

In other areas, both OODBMSs and ORDBMSs will share their destiny, for instance in database design. Apparently, once abstract data types, inheritance, triggers, user defined functions and so forth are supported, the traditional database design methods must be extended and adapted to the new functionality (which we know from OODBMSs to be a severe problem).

Unfortunately, the academically clean solution will *not* be among the winners: a clear-cut, comprehensive, object-oriented data model and system that would include the "relational world" as a special case. Whereas OODBMS-vendors are too small to engage in and promote such an approach, RDBMS-vendors do not have any interest in it since this would incur huge investments.

3 Beyond OODBMS: OBMS

Given the diversity of developments and different types of DBMSs, we now turn toward some of the major directions in which database technology will evolve. Particularly, we speculate about the future prospects of object database systems.

We believe that the future will show two trends:

1. database technology will be even more ubiquitous in computing,
2. the successful DBMSs will no longer be monolithic systems, but will instead allow to "bundle" user-defined or external components.

In order to accomplish these trends, object-orientation in databases will be a prerequisite. In this respect, "object-orientation" can mean object-oriented or object-relational DBMSs; we thus henceforth subsume both types of systems under the term *objectbase systems* (OBSs and OBMSs).

Typical users work with data that is stored in a variety of data stores, whereby only a fraction of the data is actually managed by database systems. Integrating all these diverse data stores will provide users a uniform view of all their relevant data, will avoid redundancies and inconsistencies, and as a consequence ease the daily tasks of many users significantly. Moreover, although the data stored and managed by tools cannot be migrated to a database, users would benefit from database services, such as query facilities or transactions in their tools [36]. Thus, in the not too distant future we will see database systems expand and provide their services, e.g., to desktop applications, in a componentized form [4, 37].

Data stores managed by these tools and other data repositories will have to be integrated into large-scale information systems, and database services must be available for them. These information systems will be operated over the Internet or Intranets, and users will query them, without willing to bother where the desired information is actually stored (here, "where" means both, the type of data store containing the required information as well as the network node where the data store is located). In other words, the established query languages will be leveraged to much more powerful *universal query services* that are able to retrieve and browse the contents of potentially any kind of data stores anywhere in a network.

In addition to queries, databases will be accessed and manipulated in complex, long-lasting processes. The information system of the future can no longer be idealized as a database and an attached collection of application programs. Instead, in the future we will see databases as one part of large-scale cooperative environments. The behavior of these environments will be defined by process models, and databases, users, and all sorts of software systems will interact to execute processes (or synonymously, *workflows*).

DBS-services will thus be a crucial component of quite varied scenarios, and it is to be expected that no single DBMS will meet all requirements equally well and provide for all its tasks in an efficient manner. Successful solutions thus will be characterized by a component-oriented architecture which allows to add new components needed, to exchange existing components by others better suited for certain tasks, and to remove components whose functionality is not needed for specific applications [37]. In other words, we suggest that the real next great wave will be *bundable DBMSs* that allow to configure application-specific DBMSs out of pre-fabricated components.

Bundling can certainly be accomplished by traditional (purely relational) database systems and procedural programming techniques. Nevertheless, a really powerful component-oriented approach to database management will require object-orientation on all levels:

- only object-oriented modeling facilities provide the expressive power that is needed, e.g., for integrating all sorts of data,
- only object-oriented (software) design principles will allow for the required extensibility and customizability of DBMSs.

Below we discuss several areas which will be influenced by object database technology in more detail.

4 Four Cases for Objectbase Technology

In this section, we elaborate on several areas of objectbase technology which might benefit from object-orientation. "Benefit" can hereby bear different (not mutually exclusive) meanings:

- benefit in terms of object-oriented modeling concepts,
- benefit of using object-oriented architectures and implementations,
- benefit of using database services for the construction of (information) systems.

4.1 Component Technology

Component technology is an approach to build software systems out of preexisting software pieces [25]. The attractive aspect of component technology is that it allows to build new systems and extend existing ones at rather little cost, provided that reusable adequate components are available (see Fig. 1). Moreover, existing components often exhibit a higher degree of maturity and robustness than newly developed software modules. However, in order to put component technology to work, several issues must have been settled:

- a software architecture model [32] is needed that allows to specify componentized systems and to add components in a controlled way,
- systems must be dynamically reconfigurable so that new components can be added without redesigning or recompiling the whole system,
- components must be designed and implemented in such a way that they are effectively reusable (e.g., do not make assumptions about the context in which they are going to be used).

A component certainly is more abstract and of larger granularity than an object (it is rather comparable to a module in software engineering). However, object-orientation lends itself to component technology in a very natural way:

- a prerequisite for component technology (as for any kind of software reuse) is information hiding, i.e., encapsulation of object state and behavior,
- specialization, inheritance, and polymorphism provide the appropriate means to specialize and customize services/functionality in subclasses.

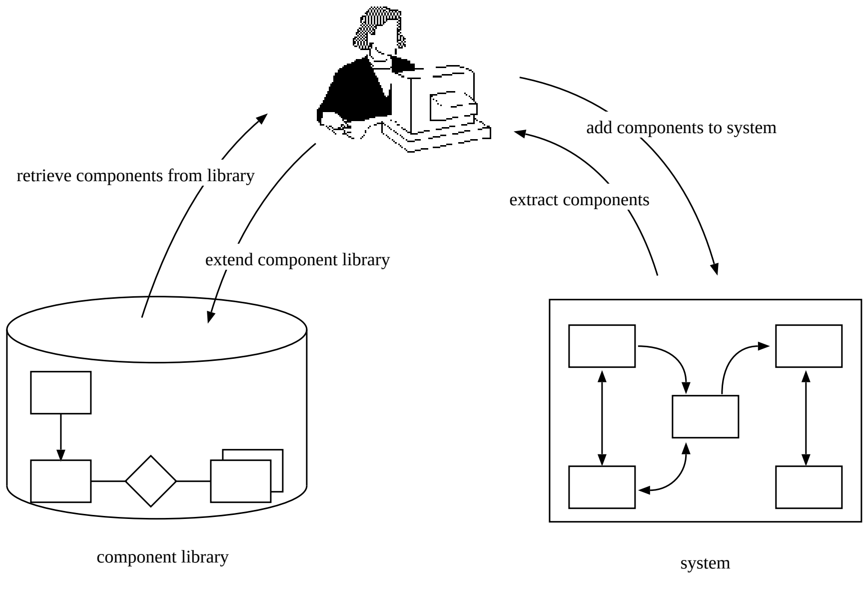

Fig. 1. System Construction based on Component Libraries

Component technology will influence databases in two ways. First, it will allow to build application systems out of components (and DBMSs will be one prominent type of component). Second, given that DBMSs are conceived as componentized software systems, component technology will allow to build DBMSs themselves out of components.

Component technology will be of paramount importance for applications development, especially for heterogeneous and distributed systems [27]. The problem here is to integrate a collection of heterogeneous systems distributed over a network into a coherent and homogeneous application system, using the various systems as components and middleware services [27] as the glue that abstracts from heterogeneity and distribution details.

The prominent representative of component technology in this sense is OMG's Common Object Request Broker Architecture (CORBA) [26]. CORBA defines an object model and an interface definition language (IDL) that is used to specify service signatures. Any functionality offered by some server and/or needed by some client must be specified in IDL. In addition, CORBA defines so-called *Common Object Services* (COS) most of which directly represent database services (e.g., persistence, transaction, query, and relationship services). CORBA has been assessed as a feasible platform for applications that are more oriented towards operation shipping, while it seems to be less beneficial for efficient data-intensive applications [31].

Another representative is Microsoft's *Object Linking and Embedding* (OLE) [27]. In OLE, related functions are packaged into interfaces. Interfaces can be defined using the IDL of DCOM (*Distributed Component Object Model*). An OLE-component is defined by a class that implements one or more interfaces and a class factory (which knows how to create instances of the class).

One step further, component technology is not only used to build application objects, but also to build DBMSs themselves [15]. A DBMS is *unbundled* into a collection of services and potential components implementing them. The notions of "component" and "services" are defined by an underlying (software) architecture model; i.e., this model defines how services and components can be defined and interrelated. Consider concurrency control, object storage, query optimization or versioning as sample (database) services (note that persistence, concurrency control and queries are also elements of OMG's COS). Then, a strict two-phase lock manager would be one possible component implementing the concurrency control service. Given unbundled ODBMS-services and components, a DBMS can be extended or customized by adding new components when new services are needed, or exchanging components when service requirements change.

The appealing aspect of database component technology is that DBMSs can be tailored to the needs of their applications: every DBMS provides just the functionality that is effectively needed, and applications do not have to pay—in terms of performance—for functionality that they do not use. A first step towards component technology are the DataBlades (or extenders, cartridges) in ORDBMSs, and the user-definable classes in OODBMSs. However, these mechanisms solely extend the set of data types. We envision the need to push component technology in DBMSs one step further, namely in that the DBMS itself can be extended or customized. For instance, it does not need a rocket scientist to define an ADT for multimedia objects whereas the critical task is to enable efficient management and retrieval of instances of such types. Thus, it might be necessary to add advanced access paths and methods, and it might even be necessary to extend the query processor and optimizer. Otherwise, instances of ADTs can be modeled and stored, but cannot be retrieved and queried with acceptable performance.

Summarizing, component technology will become an even more fundamental prerequisite of database technology for the construction of both, database applications and DBMSs. Object-orientation is the underlying paradigm that makes component technology practical in database technology.

4.2 Workflow Management

Workflow management [14] is concerned with the modeling and execution of automated production/design/administration processes. *Workflow management systems* (WfMSs) are software systems that provide for the specification and execution of workflows (see Fig. 2). A workflow type specifies the structures of workflows (i.e., subworkflows and/or atomic activities), processing entities (software systems or humans) responsible for carrying out these activities, and execution constraints (e.g., ordering constraints, deadlines, data flow). Workflow types can be instantiated; the workflow instances can be executed under control of the WfMS.

Control data comprises all the data elements that are manipulated or accessed by processing entities and routed between the various processing stations within a workflow. As an example, consider an insurance claim workflow. In this case, relevant data includes all the information about the insurance policy, the insurance case, the claimed amount, etc. Some of this data (or handles to it) will flow through a workflow, while other data will be external to the WfMS and managed by some sort of data stores (e.g., a DBMS). During workflow specification, control data must be specified, and upon workflow execution, it must be routed through the various processing stations.

Workflow management can benefit from object-oriented data modeling facilities for two kinds of data:

- control data,
- internal data structures.

Object-oriented modeling of control data yields several advantages: first, the modeling power of object-orientation can be exploited. For instance, an insurance claim is a quite complex structure, and therefore could be modeled naturally as a complex object. Second, the WfMS execution engine and the processing entities (which in turn can be software systems) need to agree on common data formats, so that using a standard object model would make sense (e.g., ODMG's [6] or OMG's [26] object models would be reasonable choices).

In addition to control data, a WfMS needs to store information about workflow types and workflow executions. The WfMS needs to retrieve this information

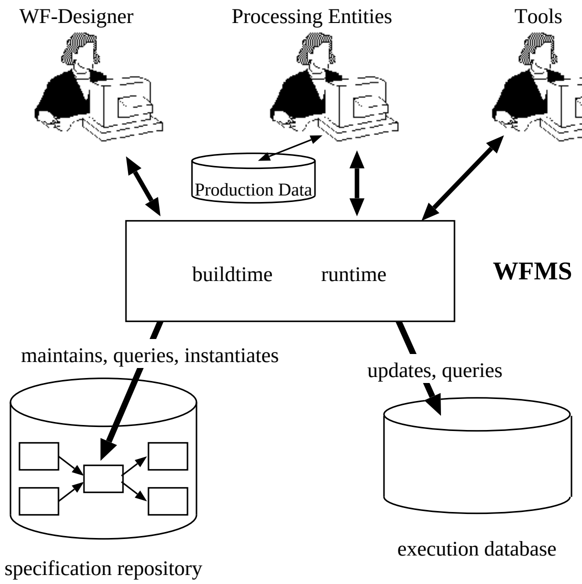

Fig. 2. Workflow Management Systems

in order to properly enact the workflow. For instance, parts of workflow specifications must be retrieved in order to determine whether and which further activities are ready to be executed and which processing entities are eligible for executing the activity. Again, all these various types of information can be modeled much more naturally in an object-oriented way. Currently, most WfMSs still use RDBMSs as platforms (e.g., InConcert [30]), and thus have to suffer the impedance mismatch in case their modeling language is object-oriented.

WfMSs must be extensible, need to be integratable with other components, and should be customizable for all sorts of platforms and environments. For instance, a WfMS should be able to operate in a CORBA-environment, which would open up the possibility to use the common object services and to integrate application objects into a coherent workflow. This integration is eased if the WfMS relies on component technology and object-orientation (e.g., as in ObjectFlow [19]).

These requirements are best met if the WfMS is built out of components. In this way, the WfMS is also flexible to use different DBMSs as platforms, or to implement parts of its functionality using database services. If available, a WfMS can greatly benefit from these services, at all the various stages of workflow specification and execution:

- data modeling and storage service: specifications of workflows information need to be modeled and stored persistently for further inspection and reuse,
- query service: workflow specifications and workflow execution states might have to be inspected, which is best supported by a query service,

- transaction service: during execution, the WfMS must ensure that its operations are recoverable and atomic (e.g., information about current execution states must not get corrupted),
- concurrency control service: typically many clients (e.g., processing entities, workflow designers) will interact with the WfMS and will thereby update its internal data structures. Thus, the WfMS must perform concurrency control in order to ensure that concurrent clients are isolated form each other,
- access control service: the WfMS has to prevent the access to and manipulation of workflow specification and execution information by unauthorized users, which can be best supported by a flexible access control service.

Hence, database services are crucial for a WfMS to manage required information about workflow specifications and executions. One step further, more advanced database services are used to leverage the DBMS to a full-fledged WfMS. In other words, given a componentized DBMS and additional services, a WfMS can be plugged together out of database and other services. For instance, ECA-rules (see above) are used for the implementation of the workflow logic. Every occurrence in the workflow system is expressed as an event, and ECA-rules define what should happen in reaction to this event [17] (for instance, a typical rule might define to execute a successor activity as soon as two preceding activities have terminated with a certain result). This style of workflow execution makes use of further database services, namely ECA-rule definition, event detection, and rule execution. These services must in turn be flexible and customizable in order to tailor their semantics to the need of WfMSs (e.g., specific event types are needed, and rule execution semantics might be different from those found in active DBMSs).

Summarizing, workflow management will become prevalent in application systems and will allow to define and enact processes executing in cooperative and distributed environments. Object-oriented database technology enables WfMSs to be designed and implemented in an object-oriented way on all levels. Advanced OBMS-functionality such as ECA-rules allows to implement certain aspects of workflow execution within the OBMSs.

4.3 Information Integration and Global Information Systems

Information integration refers to the provision of a unified and uniform view of information stemming from varied data sources. A typical example of information integration is data warehousing [7]. Aside from data warehousing, information integration can be seen in a much broader context, e.g., in terms of the data sources that are considered. Particularly, information from all sorts of sources might be desired to be integrated, such as voice mail, email, video clips, X-ray images, and so forth. Furthermore, data can be structured (as in relational databases), semi-structured (e.g., email messages), or unstructured (e.g., texts).

The real value of information integration and global information systems lies in the ability to *browse* through and *query* the integrated data. Hereby, "query" does not necessarily mean pure, traditional database queries, i.e., exact queries that return only those data elements that fully match the query. In addition, probabilistic queries as typical for information retrieval should be specifiable which return "imprecise" results that are somehow related to the query. This type of query is of particular importance for semi- or unstructured data sources (e.g., texts, images, Web-pages). Furthermore, it should be possible to specify queries on the global level, without having to worry about which data sources contains the data and how the information is formatted there. These considerations are undertaken by the global information system that decomposes the query, forwards the subqueries to the data sources, receives answers, and combines them according to the user's initial query.

The critical issues in information integration are a uniform and powerful model to define the (integrated) data from the various sources and query facilities to retrieve and combine the data from the sources.

Data warehouses usually use the relational model, because their data sources are relational and so the mapping from the data source to the warehouse is simple. This is no longer the case when general data sources are considered; then an object-oriented model will be much more feasible. Generic behavior (e.g., of "information objects") can be described in abstract classes, and specialized semantics can be captured in subclasses using inheritance. Classes and class-specific operations will be especially important when multimedia data needs to be integrated. Information objects can be complexly structured, so the ability to model complex objects and aggregations are needed. Moreover, especially when objects from different sources are associated to more complex structures, then object identity is a prerequisite [18]. Thus, for the general case of information integration, an object-oriented global model will be useful (e.g., the Garlic system [29] uses and extends the ODMG-model).

Another difference between data warehouses and federated DBMSs [33] and the kind of information integration considered here is that the former maintain copies of data (in the warehouse) and that updates and transaction management are critical issues in federated DBMSs. In contrast, here we focus on the integration of information and on queries against the integrated data sources. All the benefits of an object-oriented model apply to such a query service as well (e.g., representation of complex results at the end-user interface using object references via object identifiers, or presentation of results according to the class-specific behavior). In addition, given that the global model is object-oriented, then an object-oriented query language is a natural choice for formulating queries (i.e., OQL [6] or SQL3).

A universal query service (Fig. 3) requires an architecture model that allows to incorporate any kind of data source. This architecture model in turn will be object-oriented and based on component technology. *Wrappers* are one possibility [29], where a wrapper encapsulates a specific data source and defines a predefined behavior, namely to "understand" queries and translate them into the query language of its data source. Another possibility is OLE/COM, which is used in OLE DB [4] to define and implement components. Typical classes of components in OLE DB are "service providers" and "data providers". A data

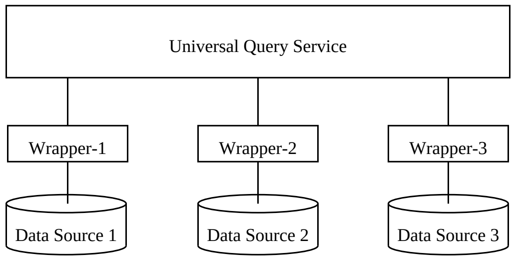

Fig. 3. Universal Query Service

provider is a component that can export so-called rowsets (i.e., sets of rows, which in turn are sets of attributes), and a service provider can take rowsets as input, transform them, and produce new rowsets as output. Thus, provided that information is given in the form of rowsets, any data source can be queried, and different queries can be combined to more complex ones, again producing rowsets as results. Note that the restriction of OLE DB to row-set-based structures might be too strong for some current types of data.

Summarizing, information integration and global information systems will gain more and more importance in Intranets and the Internet. Object database technology can beneficially support these systems and tasks with respect to powerful modeling and querying facilities.

4.4 Repositories

Future information systems will integrate much more diverse information than they do today, they will provide access to these information sources using universal query services, and complex, distributed workflows will execute within them. In order to achieve this vision, database services must be available in a very flexible form. The price for this much higher degree of flexibility is complexity. How can we manage all these componentized database services and configurations of them into complex information systems?

To that end, organizations need "meta databases" that store all the relevant information about their information systems. In order to configure and operate the kind of system we outlined above, knowledge about database services, their usage, integrated data sources, involved applications and so forth must be available. In other words, we need database technology in order to construct the powerful information systems of the future: the construction of these systems must be assisted and guided by repository systems [3].

A *repository manager* supports the management of information about design artifacts (in our case, "artifacts" would represent any kind of information about database services, components, etc.). Users interact with the repository manager through *repository tools*, i.e., editors, browsers, and so forth that allow to store new artifacts in the repository and to extract old ones from it. Aside from the storage of artifacts, a repository manager implements further tasks, such as version and configuration management to represent design histories and to model compositions of artifacts, or workflow control to model and enact design processes.

Repositories are related to database technology in two ways. First, repository support is crucial to methodize DBMS-development and usage. Second, a repository system can in turn be built using objectbase technology.

As for the first relation, OBMSs and their application systems will be manageable only if *information about them* is available in an appropriate form. For instance, assume an ORDBMS with a large number of pluggable ADTs (or DataBlades, extenders). Information about these artifacts is crucial for managing and maintaining the ORDBS. Lack of this information will lead to redundancies and inconsistencies, and chaos will be the consequence. Similarly, imagine a global information system that integrates data from a large number of heterogeneous sources. This information system will be manageable only if meta data about each of the sources is available (which data they contain, how they can be accessed, how regularly they are updated and how frequently updates are propagated, etc). Moreover, wrappers may have to be built for each of these sources, and both, the design and the implementation of the wrappers should be reusable during the integration of further sources. Or, assume a workflow system with a large number of (potentially versioned) workflow specifications, processing entities, etc. Again, all this information must be kept accessible to manage and maintain the workflow system and to reuse artifacts for future specifications.

Hence, while the increased flexibility of component technology and extensibility of DBMSs are certainly to be appreciated, users and administrators need support to master the equally increasing complexity of DBMSs. Otherwise, if sufficient support to systematically develop and maintain these systems is not provided, users will probably refrain from using all these advanced features.

Fortunately, OBMS-technology itself can help overcome this problem (in some sense, OBMSs can bootstrap themselves). This is the second relationship between repositories and object database technology: OBMSs can be leveraged to repository managers when adequate OBMS-services are available. Version management facilities are required to define and manage the complex artifacts of a repository. OQL and SQL3 are feasible languages to query repositories and to retrieve artifacts. An ECA-rule service can be used to specify and enforce the complex constraints typical for repositories and to implement notification mechanisms. Workflow management and design transaction support can be provided as discussed in section 4.2 above, and by customizing a concurrency control and transaction service. The latter case is also a nice example for the customization of internal DBMS-components, i.e., adding a new ADT is not sufficient here.

Summarizing, repository technology is mission-critical for users to master the complexity of OBMS-technology and its applications. On the other hand, repositories are also typical applications that can be built using OBMS-functionality and component technology.

5 Challenges to Future Work

In this section we present a set of challenges to research and practice that have to be met in order to turn OBMSs into an even more useful technology and to master the complexity of advanced OBMS-applications.

As for the *functionality of OBMSs*, additional concepts need to be integrated into existing systems and products. For instance, support for ECA-rules is required as a basic mechanism to implement a variety of tasks such as consistency constraint enforcement. ORDBMSs already support at least simple forms of ECA-rules (they consider, e.g., only a restricted set of event types), while for OODBMSs only research prototypes are available (e.g., [13]). In order to convince vendors that ECA-rules are a viable technology, users should urge them to give support for consistency constraints, notification, etc., and researchers should come up with models that are powerful yet usable and that can be implemented in a robust and efficient manner. Another area which requires more work are query languages and query processing techniques for global information systems. In this respect, research is necessary to develop systematic information integration techniques (e.g., easy-to-build and reusable wrappers). The existing and well-understood query languages and query processing techniques should be extended to cover also distributed and heterogeneous data sources, including semi- and unstructured data, and exact as well as probabilistic queries.

Concerning the usage of OBMSs, a better understanding of and more experience in how to develop OBMS-applications is desperately needed. Design methods and corresponding tools should be developed. These methods and tools should built upon the existing methods and tools established in software engineering, since those are already in use and designers will not be willing to use multiple, completely different methods/tools in parallel. While database design at first glance is an item on the research agenda, it also needs input from practitioners, because only they have real experience in building real-life, mission-critical large systems.

In a broader sense, appropriate design and development support is the decisive factor for OBMS-usage. As we have outlined above, OBMSs lend themselves in a natural way to play important roles in a variety of scenarios, ranging from information integration and global information systems to workflow management. Component technology and OBMSs are viable base technologies for these scenarios, but users will rely on the appropriate support of how to develop these applications in a systematic way.

Therefore, as a first challenge in this context, component object models must be developed that are powerful enough to describe componentized systems and OBMSs. Existing and new models (e.g., CORBA, COM, wrappers) need to be evaluated with respect to which scenario they support well, and experiences in using them are needed. Second, a method how to bundle components together into powerful OBMS-applications is required. This method should be systematic (i.e., rely on established techniques from software engineering) and should enable reusability of designs and openness of constructed systems.

6 Conclusion

In this paper, we have reviewed and surveyed object-oriented concepts in database systems and have hypothesized about (some) future trends and challenges in this area. We foresee that the trend towards large information systems that are built out of smaller units and that are required to (inter)operate in distributed environments will continue. DBMSs will have to exhibit a much higher degree of openness and flexibility in order to smoothly integrate into such large-scale systems and contribute their services in an optimal way.

From an abstract point of view, OBMSs (but not necessarily products as they are available today) are powerful enough to meet these requirements. However, even for them much more work remains to be done, e.g., with respect to methods for how to systematically unbundle and construct DBMSs. Approaches and experiences might—and should—be adapted from other research areas, such as component technology and software architecture. If the "componentization" efforts succeed, then OBMS-services will turn into commodities met in all sorts of future information systems and tools.

Acknowledgements

We thank Dimitrios Tombros, Markus Kradolfer, and Ruxandra Domenig for their helpful comments on an earlier version of this paper.

References

1. M. Atkinson, F. Bancilhon, D.J. DeWitt, K.R. Dittrich, D. Maier, and S.B. Zdonik. The Object-Oriented Database System Manifesto (a Political Pamphlet). In *Proc. 1^{st} Intl. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989.
2. F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System.* Morgan Kaufmann Publishers, 1992.
3. P.A. Bernstein and U. Dayal. An Overview of Repository Technology. In *Proc. 20^{th} Intl. Conf. on Very Large Data Bases*, Santiago, Chile, September 1994.
4. J.A. Blakeley. Data Access for the Masses through OLE DB. In *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, Montreal, Canada, June 1996.
5. L. Cardelli and P. Wegner. On Understanding Types, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4), 1985.
6. R.G.G. Cattell and D. Barry, editors. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers, San Francisco, 1997.
7. S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), March 1997.
8. E. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6), 1970.
9. D.J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6), June 1992.

10. K.R. Dittrich. Object-Oriented Data Model Concepts. In A. Dogac, T.M. Özsu, A. Biliris, and T. Sellis, editors, *Advances in Object-Oriented Database Systems*, volume 130 of *Computer and System Sciences*. Springer, 1994.
11. K.R. Dittrich, S. Gatziu, and A. Geppert. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In *Proc. 2^{nd} Intl. Workshop on Rules in Database Systems*, Athens, Greece, September 1995. Springer.
12. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin/-Cummings, 2^{nd} edition, 1994.
13. S. Gatziu, A. Geppert, and K.R. Dittrich. Integrating Active Mechanisms into an Object-Oriented Database System. In *Proc. 3^{rd} Intl. Workshop on Database Programming Languages (DBPL)*, Nafplion, Greece, August 1991.
14. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2), April 1995.
15. A. Geppert and K.R. Dittrich. Constructing the Next 100 Database Management Systems: Like the Handyman or Like the Engineer? *ACM SIGMOD Record*, 23(1), March 1994.
16. A. Geppert and K.R. Dittrich. Specification and Implementation of Consistency Constraints in Object-Oriented Database Systems: Applying Programming-by-Contract. In *Proc. Datenbanken in Büro, Technik und Wissenschaft (BTW)*, Dresden, Germany, March 1995. Springer.
17. A. Geppert, M. Kradolfer, and D. Tombros. Realization of Cooperative Agents Using an Active Object-Oriented Database Management System. In *Proc. 2^{nd} Intl. Workshop on Rules in Database Systems*, Athens, Greece, September 1995. Springer.
18. M. Härtig and K.R. Dittrich. An Object-Oriented Integration Framework for Building Heterogeneous Database Systems. In *Proc. IFIP DS-5 Conf. on Semantics of Interoperable Database Systems*, Lorne, Australia, November 1992.
19. M. Hsu and C. Kleissner. ObjectFlow: Towards a Process Management Infrastructure. *Distributed and Parallel Databases*, 4(2), 1996.
20. DB2 Relational Extenders. White Paper (http://www.software.ibm.com/pubs/-papers/), IBM Corp., May 1995.
21. R. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4), 1990.
22. K. Kulkarni, M. Carey, L. DeMichiel, N. Mattos, W. Hong, M. Ubell, A. Nori, V. Krishnamurthy, and D. Beech. *Introducing Reference Types and Cleaning Up SQL3's Object Model*. International Organization for Standardization, August 1995.
23. J. Melton. A Shift in the Landscape. Assessing SQL3's New Object Direction. *Database Programming & Design*, 9(8), August 1996.
24. O. Nierstrasz. A Survey of Object-Oriented Concepts. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. ACM Press, New York, 1989.
25. O. Nierstrasz and L. Dami. Component-Oriented Software Technology. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*. Prentice Hall, London, 1995.
26. The Object Management Group. *The Common Object Request Broker: Architecture and Specification. Revision 2.0*, July 1995.
27. R. Orfali, D. Harkey, and J. Edwards. *The Essential Client/Server Survival Guide*. John Wiley & Sons, 2^{nd} edition, 1996.

28. Rational Software Corp., Santa Clara, CA. *Unified Modeling Language: Notation Guide*, 1997.
29. M.T. Roth and P. Schwarz. Don't Scrap it, Wrap it! A Wrapper Architecture for Legacy Data Sources. In *Proc. 23rd Intl. Conf. on Very Large Data Bases*, Athens, Greece, August 1997.
30. S.K. Sarin. Object-Oriented Workflow Technology in InConcert. In *Proc. of the IEEE COMPCON Spring*, Santa Clara, February 1996.
31. J. Sellentin and B. Mitschang. Möglichkeiten und Grenzen des Einsatzes von CORBA in DB-basierten Client/Server-Anwendungssystemen. In *Proc. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft*, Ulm, Germany, March 1997. Springer.
32. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
33. A.P. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, 22(3), September 1990.
34. I.-Y. Song and E.K. Park. Object-Oriented Database Design Methodologies: A Survey. In T.W. Finin, C.K. Nicholas, and Y. Yesha, editors, *Selected Papers from 1st Intl. Conf. on Information and Knowledge Management*, volume 752 of *Lecture Notes in Computer Science*. Springer, 1992.
35. M. Stonebraker and D. Moore. *Object-Relational DBMSs*. Morgan Kaufmann Publishers, 1996.
36. D. Vaskevitch. Database in Crisis and Transition: A Technical Agenda for the Year 2001. In *Proc. ACM-SIGMOD Intl. Conf. on Management of Data*, Minneapolis, May 1994.
37. D. Vaskevitch. Very Large Databases. How Large? How Different? In *Proc. 21st Intl. Conf. on Very Large Data Bases*, Zurich, Switzerland, September 1995.

This article was processed using the LaTeX macro package with LLNCS style