Ersch. in: OOIS 2001 : 7th International Conference on Object-Oriented Information Systems, 27-29 August 2001, Calgary, Canada : proceedings / ed. by Yingxu Wang ... - London [u.a.] : Springer, 2001. - S. 25-34. - ISBN 978-1-85233-546-5

OMS Java: Providing Information, Storage and Access Abstractions in an Object-Oriented Framework

Adrian Kobler, Moira C. Norrie, Beat Signer, and Michael Grossniklaus
{kobler, norrie, signer, grossniklaus}@inf.ethz.ch

Institute for Information Systems
ETH Zurich, CH-8092 Zurich, Switzerland

Abstract. In this paper, we present the main objectives and components of the OMS Java data management framework. We argue that developers of modern information systems require high-level application programming interfaces, storage platform independence and support for universal client access. We describe how the OMS Java framework provides three level of abstractions – storage, information and access in order to realise these objectives. We then present each of these layers in turn – starting with the information abstractions which lie at the core of the system and then going on to the storage and access layers.

1 Introduction

To support the engineering of modern information systems, we require a new generation of open, extensible data management frameworks that provide high-level application programming interfaces and aid modularity and reuse. Ideally such a framework should be general enough that it is independent of the implemention platform, not only in terms of hardware and operating system, but also the storage platform. Additionally, in a world which is rapidly moving to a paradigm of *community information spaces*, it is vital that support for universal client access be an integrated part of such a framework and not simply an afterthought.

Clearly, a framework that meets all of these requirements will be complex. However, the important factor is that the complexity lies in the implementation of the framework and not in its use. Further, to make the complexity manageable and ensure generality, it is important that the framework is based on clear concepts and has a well-defined structure.

OMS Java [KN00a] is a data management framework that we have developed in line with these requirements and goals. In this paper, we present an overview of OMS Java in terms of its three layers of abstraction – *information*, *storage* and *access*. The information abstractions specify the core model on which the framework is based in terms of data constructs, operations and workspaces. The storage abstractions hide persistent storage details from the application developer and provide storage platform independence. The access abstractions enable applications with universal client access to be developed quickly and easily.

We start in section 2 with a more detailed look at the requirements of such frameworks and the abstraction layers. Each of the three abstraction layers is then presented in turn, beginning in section 3 with a description of the information layer which is the core of the framework. Section 4 then presents the storage layer and the main architecture of the OMS Java framework. In section 5, we turn to consider the access layer which deals with the web server part of the OMS framework and universal client access. Concluding remarks are given in section 6.

2 Framework Requirements

Most existing frameworks fail primarily in their support for high-level application programming interfaces. At the core of information management is access to, and the manipulation of, large collections of objects and the associations and dependencies between them. Yet the data abstractions provided by many systems tend not to be at the same level as that of the application model.

One approach is to base information system development on relational technologies, using JDBC [Ree97] as the interface between the Java application objects and the storage objects and forcing the application developer to manage the mapping between the two.

Another approach is to provide a persistent object framework in which case the developer is relieved from the task of having to perform such mappings. Such systems include object-oriented database systems (OODBMS) and also persistent Java systems such as Java Data Objects [JDO00] and PJama [PAD+97]. The problem with those systems is that while the storage management aspect is catered for little or no support is provided for the management of the information space and client access to this space. For example, these systems usually provide little support for accepted database concepts of semantic classification, constraints, triggers and both schema and object evolution. In addition, there is usually no or little support for the process of *information system engineering* and the coding effort is often tedious and repetitive both within and across applications. In the end, the application developer must provide their own mini-framework to aid the development process.

At the level of the application interface, most data management systems and frameworks left it to the developer to use a preferred GUI framework. With the development of the web, the importance of providing web interfaces quickly and easily was recognised, especially as new potential markets for data management technologies emerged in terms of e-commerce and web content management systems. New systems and frameworks rapidly appeared, but most are tied into HTML technologies. It has now been recognised that it is better to exploit XML technologies and use XML [XML00] as an intermediate data format and use XSLT templates [XML00] for presentation. However, in many cases it is not so easy to cleanly adapt existing systems and frameworks to an XML-based solution.

The design and development of the OMS Java framework has always had the goal of supporting the application developer through the provision of a high-level application programming interface based on well-defined information abstractions. Further it was defined with openness and extensibility in mind. The extension and re-engineering of the framework to provide universal client access and web content management support based on XML technologies was carried out within a few person-months.

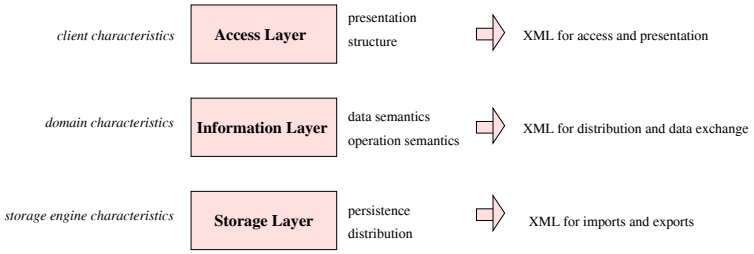

Fig. 1. OMS Java Abstraction Layers

We show the resulting abstraction layers of the OMS Java framework in figure 1. In the middle, reflecting its central role, is the information layer which provides abstractions dealing with the core constructs and operations of the underlying data model in terms of data and operation semantics. At this level, the application developer is concerned with the characteristics of the application domain in terms of information objects and operations.

At the lower level, we have the storage layer which provides storage abstractions dealing with issues of persistence and distribution. These abstractions hide from the application developer the characteristics of particular storage engines. This provides storage platform independence, enabling applications to be easily migrated across platforms and a platform to be chosen that matches the requirements, budget and system support knowledge of a given operational environment.

The access layer provides abstractions dealing with issues of presentation and structure on client devices. At this level, the application developer is concerned with client characteristics in terms of access device, authorisation and both information and presentation preferences.

As stated earlier, the latest versions of the OMS Java framework are based on XML technologies – in particular with respect to the access layer. However, as indicated on the right-hand side of figure 1 , we are actually using XML technologies at all three levels. At the storage level, we use XML as a format for the importation and exportation of data. At the information level, it is used as a general format for data exchange and also for distributed processing. At the access level, we use it for client access and presentation. Accordingly, we have three types of XML documents and therefore three DTDs (Document Type Definitions) corresponding to the three layers.

3 OMS Java Information Layer

The information layer is at the heart of the system in that it defines the data abstractions in terms of which the application domain will be represented. Thus, it specifies the set of constructs and operations that define the core of the application programming interface. In other words, the information layer defines the *data model* on which both the framework and application design are based. This data model must therefore be both semantically expressive to aid the application developer and also amenable to efficient, platform-independent implementation.

The OMS Java framework is based on the OM object data model [Nor93,Nor95] which is a generic collection-based model offering a rich set of collection constructs and also a powerful collection algebra. The model combines features of entity-relationship and object-oriented models, but it is important to note that in contrast to many of these it also has a full operational model that includes a query algebra and language, a data manipulation language and also triggers and constraints.

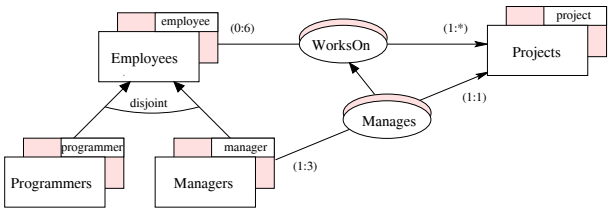

Fig. 2. OM Application Model

In figure 2 , we give an example application model to show the main constructs of the OM model. Entity sets are modelled as collections of objects of a

given type. Specialisation are modelled as subcollections and classification constraints may be placed over two or more subcollections. For example, `Managers` and `Programmers` are both subcollections of `Employees` and they are specified to be `disjoint` which means that they have no common members.

Associations between objects are represented as a special form of *binary collection* which is a collection of pairs. A binary collection has two forms of constraint over it – the specification of a source and target collection to restrict pair membership, and cardinality constraints. In the example of figure 2 , `WorksOn` is a binary collection containing pairs of the form *(e,p)* where e is a member of `Employees` and p a member of `Projects`. e may be associated with 0 to 6 members of `Projects` and p can be associated with 1 or more members of `Employees`. Note that there is also a subcollection relationship between binary collections `Manages` and `WorksOn`.

The collection model – constructs, constraints and operations – are generalised over both unary and binary collections and also over collections with set, bag, ranking (ordered, no duplicates) and sequence behaviours. Note that it is also possible to dynamically change the behaviour of a collection. For instance, a collection with *set* behaviour can seamlessly evolve into one with *bag* behaviour. This contrasts with the *Collections Framework* (CF) which is part of the Java 2 Platform Standard Edition (J2SE) and which is supported by most OODBMSs. In the CF, the behaviour of a collection is specified by the corresponding Java class and must be determined during implementation, whereas OM collections are *semantic* collections and can change their behaviour at run-time.

While the full details of the OM model cannot be presented here, there are two special points that merit further discussion with respect to data management frameworks. The first is that the OM model has a much richer classification model than that supported in typical object-oriented programming languages. An application entity may have several roles with role-dependent features. To model this directly requires multiple instantiation, i.e. the possibility for objects to have more than one type. For example, if we removed the `disjoint` constraint in figure 2 , we would allow for the case that an employee is both a manager and a programmer. We then say that an object has type units employee, manager and programmer. Further, the model supports role modelling through a dynamic composition of objects from the information units corresponding to its various type units as determined by context. In the case of accessing an object through a collection, it is the membertype of this collection that determines context.

The OM model can really be considered as a two-level model — the collections and constraints at the upper level and the types at the lower level dealing with object representation and behaviour. Correspondingly, we represent OM objects in the OMS Java Framework by a class `OMObject` which represents the object in terms of a persistent OID and alias. Further, an OM object can be associated to one or more Java instances representing the various *information units* of an object.

Figure 3 shows these two levels. The OMS Java Framework provides the classes for the various OM constructs such as collections and associations. The application developer then specifies *type units* of OM objects in terms of the OMS Java data definition language (DDL). In addition, a developer can provide application classes corresponding to type units for defining methods and special data structures, but it is also possible to use the scripting language of OMS Java for method specification.

Note also that the constructs of the OMS Java framework include an `OMWorkspace` construct. OM has a transactional, persistent workspace model. This means that an application program has a persistent workspace. The updates stored in the workspace are written to the database only when a commit opera-

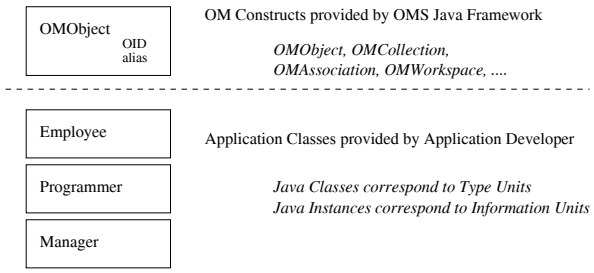

Fig. 3. OM Objects in OMS Java Framework

tion is performed. At any time, a rollback operation may be performed to delete the changes stored in the workspace.

The second point to mention is the importance of the association concept and its realisation as a separate construct in both the model and the framework. The importance of maintaining consistency between mutual references has long been recognised and, for this reason, there is specific support within the model of the ODMG standard [CBB+00]. However, few OODBMSs actually support this within their model and, even in the cases of those that do such as Objectivity/DB (www.objy.com), it is still tedious for the application programmer to specify and maintain these. A major problem is that the associations are specified as part of the target and source class definitions, rather than as a separate construct. As a result, the programmer must develop classes in conjunction. In contrast, OMS Java application developers can focus on one part of the application at a time and then link the parts together through the introduction of associations. Adding an association has no effect on the classes being associated. This brings clear advantages in terms of modular development and class reuse.

Generally, existing OODBMS and frameworks are very limited in terms of the forms of constraints that they manage. In most cases, all consistency checks and controls are left to the programmer. For example, current OODBMS do not support the notion of a *subcollection*. This means that the application programmer must explicitly maintain such a containment relationship himself. In the case of OMS Java, triggers may be used to automatically maintain consistency of a subcollection relationship under updates to collection membership.

To demonstrate how much the application developer's task can be simplified by providing basic collection, subcollection and association constructs, we implemented a simple framework for ObjectStore (www.odi.com) providing these constructs. This framework is used in a practical course on OODBMS, and on every occasion, there is a strong positive feedback from the students regarding the reduction of development time and simplification of code.

Further, the fact that associations are represented as separate constructs enables operations to be performed over these associations. In practice, we have found that the key to many complex queries lies in focusing on the associations and applying operations such as composition, nesting and special forms of selection to locate objects of interest and only then processing these objects.

4 OMS Java Storage Layer

Within the OMS Java framework, persistent storage is provided by an OMS Java server component. One or more OMS Java workspaces can be connected to an OMS Java Server using the Java Remote Invocation Mechanism (Java RMI) as indicated in figure 4. Optionally, also an OMS Java workspace can be made persistent in which case we can distinguish between objects stored only locally and ones managed by the server.

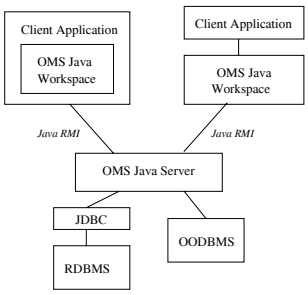

Fig. 4. OMS Java Architecture

Approaches to persistent object management can be classified into *mapping approaches* or *direct storage approaches*. In the case of mapping approaches, application objects are mapped to storage objects or relations. The mapping either be performed statically or dynamically at run-time. As an example of the former approach, a relational DBMS may be used to store application data with Java applications using JDBC to store and retrieve data. In this case, it is the application programmer who performs the mapping between the persistent data and the application objects. Other systems perform the mapping automatically by providing storage abstractions at the level of application objects rather than at the level of storage objects. For example, Sun's Java Blend product (www.sun.com) supports the run-time mapping of Java objects to an underlying DBMS, and the architecture described by the Java Data Objects specification [JDO00] provides a transparent interface for persistent data storage.

In the case of direct storage approaches, the storage engine is capable of storing the application objects directly. For example, the ODMG standard [CBB+00] defines interfaces and language bindings to object-oriented DBMSs such as ObjectStore (www.odi.com) and Objectivity/DB (www.objy.com). In addition, there have been projects such a PJama [PAD+97] which integrate the notion of orthogonal persistence into the Java programming language through changes to the Java Virtual Machine.

Since one of the main goals of the OMS Java framework is to enable application developers to design and implement applications without having to deal with implementation aspects of storage management and, at the same time, have a choice of storage platform, we chose a dynamic mapping approach – but in fact combine many features of existing approaches to persistence.

The storage management component of OMS Java has been designed in such a way that it is possible to use various storage engines for the storage of application objects. For example, as shown in figure 4, a relational DBMS may be used via a JDBC interface or a given ODMG-compliant OODBMS using a Java language binding. In addition, we have implemented storage interfaces based on persistent Java systems such as PJama and ObjectStore PSE, lightweight databases such as Berkeley DB (www.sleepycat.com) and even using simple object serialisation.

We use a mapping approach to achieving persistence. In our mapping, every Java application object is represented by one or more state container objects as indicated in figure 5. Two categories of attribute values can be stored in state containers – base type values and object references which, as with other persistent object systems, are our own unique persistent OIDs.

Two aspects that have to be taken into account in the design of a client/server persistent object framework are the handling of large collections of objects and the number of simultaneously open remote connections. For example, in OMS

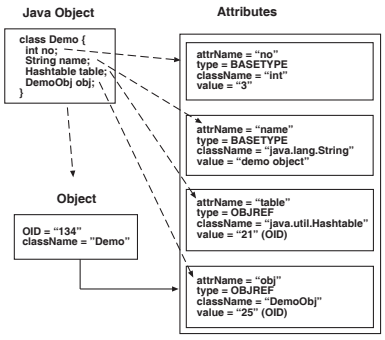

Fig. 5. Java Instance and its State Containers

Java, all data structures for bulk information processing such as collections are based on *maps* and *lists*. Hence, the OMS Java workspace uses special proxy classes `DBMap`, `DBList` and `DBIterator` to access collections of objects remotely through JAVA RMI without retrieving the entire collection object over the network. These proxy classes have corresponding remote objects on the server side and access to a specific instance of a map or list is given by its unique OID. Note that there is also a proxy class for state container objects. To keep the number of open connections small and network traffic efficient, the only connections are through the various client and associated server objects described above.

Integrating a new DBMS into the OMS Java framework as a storage platform requires that only a few interface classes, for example, for state container objects, maps, lists and iterators be implemented using the API of that DBMS. Our experience shows that additionally a small number of DBMS specific classes also need to be implemented – making it a total of typically around 10 classes. For optimisation reasons, it is possible to configure the storage manager to use indexes for certain values and to extend it with customised data structures. For this, a developer can specify them through API of a specific DBMS or define them using a DBMS-independent high-level framework such as *the eXtreme Design Framework* [Kob01] which is based on the same storage management component as OMS Java.

It is beyond the scope of this paper to describe the OMS Java storage framework in detail. More information about it can be found in [SKN98,KN00b] where we also discuss the extensibility support of OMS Java in terms of spatial and temporal data management.

5 OMS Java Access Layer

As stated previously, a modern data management framework must support the development of web-based applications and this includes access from, not only standard desktop web browsers, but also various forms of mobile client devices such as phones and PDAs. In such a rapidly evolving environment as the web (along with mobile and pervasive computing), it is vital that such a framework is not tied into any one protocol or type of client device.

Generally, we use the term *document* to refer to a presentation unit delivered to the client in response to a single access request. Thus, it may be an HTML document, a WML document, a PDF file or a voice text — the format of which may depend on a whole range of parameters such as the client device, user preferences, context as well as the information requested.

To achieve a general access layer, it is important to separate out the notions of *information*, *content* and *presentation*. The former corresponds to the actual objects of the information layer that represent application entities. By *content*,

we mean the content of documents that will be presented to the user. Such a document may be composed from many information objects. The *presentation* specifies the precise presentation format of these documents in terms of both the document format and the layout.

In figure 6, we show the general web server architecture for the access layer of the OMS Java framework. For a specific application, all client access is via a single Java servlet [Mos98] – the Entry Servlet. The Entry Servlet detects the user agent type from the HTTP request header and delegates the handling of the request to the appropriate servlet. For example, we show servlets to handle requests from HTML browsers, XML browsers and also WAP phones in terms of WML browsers.

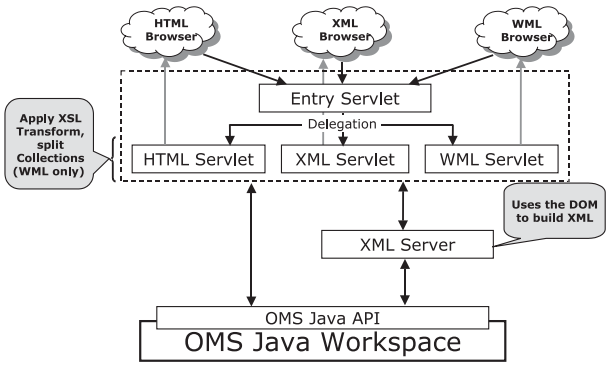

Fig. 6. OMS Java Web Access

The request handling servlets then access the database by connecting to an OMS Java workspace via the OMS Java API. The connection may either be direct or via the OMS Java XML server. Direct connections deal with requests that do not involve data retrieval such as updates or checking membership of an object in a collection. The XML server forwards requests to the OMS Java workspace and generates XML representations for any data objects to be returned to the servlets. The requesting servlets then use the appropriate XSLT templates to transform the XML results to the corresponding documents to be returned to the client.

Note that we are not storing an XML documents, but rather generating them dynamically based on a hierarchical view of data derived at access time. Actually what we generate is the DOM (document object model) structure, rather then the XML document itself, since it serves only as an intermediate structure in our architecture and is immediately processed using XSLT templates to produce the document sent to the client.

Using generic XSLT templates for various client devices, we are able to provide generic browsers and editors for the current set of supported client types. Adding a new client type involves implementing the corresponding servlet and writing the appropriate XSLT templates. Our initial framework supported HTML, XML and WML browsers and we were able to add support for NTT DoCoMo's Imode internet access system (mainly use by mobile phones in Japan) with only a few hours work. In [SGN01], we describe the architecture in detail and describe also the implementation of a community agenda using the framework.

While the above architecture is sufficient to support universal client access, developing a specific application interface in terms of documents and presentations requires a rather tedious and sometimes complex process of producing

customised XSLT templates. We therefore chose to further extend the OMS Java framework to fully support the engineering of web applications through *web content management*.

A general problem of current web content management systems such as Vignette [Cor01] and Interwoven [Inc00] is the fact that they tend to address the problem from the side of document generation and management rather than from that of semantic information content. In contrast, many of the proposals from the database side (e.g. [CFP99,FLM98]) tend to focus on presenting the contents of a database on the web and neglect the operational aspects of developing and managing a complex web site.

To support web content, meant extending the data managed by OMS Java to include also document content and presentation data as well as the application data. The extended framework OMS Java CMS (Content Management System) provides in-built support for web content management via pre-defined objects types and collections. The resulting schema is rather large and complex so we show only the main components of the schema in figure 7.

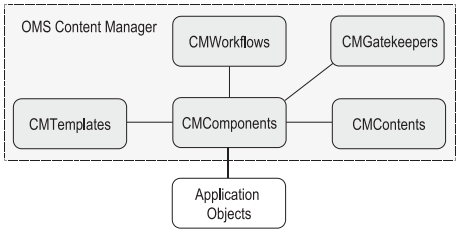

Fig. 7. OMS Java Web Content Management

Documents are composed from components which may be either simple or compound. A simple component is an object of a simple component type such as text, URL, link or image. A compound component is known as a *container* and these represent recurring component groups. To provide multi-lingual support, we distinguish between components and contents and represent the language-dependent part of components.

The rendering of objects is based on XSLT templates which are also objects of the OMS Java workspace. However, in contrast to most existing web content management systems, the transformation is not based on a single static XSLT stylesheet, but rather performed using a dynamically generated stylesheet. Each component may be associated with a default template and, optionally, several context specific templates.

Ideally, one must be able to change and extend a web application while it is up and running. Components therefore have a life cycle consisting of development, approval and active phases with only objects in the active phase actually appearing on the web site to users other than the developers. In OMS Java CMS, the state of the components are represented and controlled by workflows. Gatekeeper objects are responsible for filtering document components according to specified criteria such as the workflow state. Further details of OMS Java CMS can also be found in [SGN01].

6 Conclusions

In this paper we have presented an overview of the OMS Java framework. Our intention was to focus not on the technical detail, but rather on the requirements that must be demanded of modern data management frameworks if they are to support information system engineering. It is insufficient to cater only for the requirements of persistent storage. One must also ensure that the abstractions

provided to the application programmers enable then to concentrate on the application semantics and to work at a high-level of abstraction. For this reason, we have based the core of the OMS Java framework on a generic, semantic object data model. Further, we stress the importance that data management frameworks must also support the general benefits of object-oriented software construction and support modularity and re-usability and the importance of using separate association constructs to achieve this.

Finally, we emphasise the importance of providing the necessary access abstractions to support the development of application interfaces in the context of universal clients. For full support of web site engineering, this requires also support for content management. Having developed an initial version of the OMS Java CMS framework for this purpose, we are currently developing applications with a view to refining and/or extending this part of the framework.

References

[CBB+00] R. G. G. Catell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann Publishers, 2000.

[CFP99] S. Ceri, P. Fraternali, and S. Paraboschi. Design Principles for Data-Intensive Web Sites. *SIGMOD RECORD*, 28(1), 1999.

[Cor01] Vignette Corporation. Vignette Content Management Server. White Paper, February 2001.

[FLM98] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3), 1998.

[Inc00] Interwoven Inc. Application Development using Interwoven : Version 1.1. White Paper, December 2000.

[JDO00] Java Data Objects, JSR 000012, Version 0.8. Technical report, Sun Microsystems, www.sun.com, June 2000.

[KN00a] A. Kobler and M. C. Norrie. OMS Java: A Persistent Object Management Framework . *L'Objet*, 6, November 2000.

[KN00b] A. Kobler and M. C. Norrie. OMS Java: An Open, Extensible Architecture for Advanced Application Systems such as GIS. In *International Workshop on Emerging Technologies for GEO-Based Applications*, Ascona, Switzerland, May 2000.

[Kob01] Adrian Kobler. *The eXtreme Design Approach.* Phd thesis, Department of Computer Science, ETH, CH-8092 Zurich, Switzerland, 2001.

[Mos98] Karl Moss. *Java Servlets.* McGraw-Hill, 1998.

[Nor93] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *12th Intl. Conf. on Entity-Relationship Approach*, pages 390–401, Dallas, Texas, December 1993. Springer-Verlag, LNCS 823.

[Nor95] M. C. Norrie. Distinguishing Typing and Classification in Object Data Models. In *Information Modelling and Knowledge Bases*, volume VI, chapter 25. IOS, 1995. (originally appeared in Proc. European-Japanese Seminar on Information and Knowledge Modelling, Stockholm, Sweden, June 1994).

[PAD+97] T. Printezis, M. Atkinson, L. Daynès, S. Spence, and P. Bailey. The Design of a new Persistent Object Store for PJama. In *The Second International Workshop on Persistence and Java*, 1997.

[Ree97] George Reese. *Database Programming with JDBC and Java.* O'Reilly & Associates, 1997.

[SGN01] B. Signer, M. Grossniklaus, and M. C. Norrie. Java Framework for Database-Centric Web Site Engineering. In *Proc. 4th Workshop on Web Engineering*, Hong Kong, May 2001.

[SKN98] A. Steiner, A. Kobler, and M. C. Norrie. OMS/Java: Model Extensibility of OODBMS for Advanced Application Domains. In *Proc. 10th Conf. on Advanced Information Systems Engineering (CAiSE'98)*, Pisa, Italy, June 1998.

[XML00] Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, http://www.w3.org/xml/, October 2000.