A UML Variant for Modeling System Searchability

Axel Uhl[1] and Horst Lichter[2]

[1] Interactive Objects Software GmbH
Freiburg, Germany
uhl@io-software.com
[2] Aachen Technical University
Aachen, Germany
lichter@informatik.rwth-aachen.de

Abstract. Internet search engines today are facing problems in keeping up with the pace of web growth. Two facts are responsible: bandwidth bottlenecks due to central indexing; *deep web* (or *invisible web*) contents that are inaccessible for search engines. Powerful and flexibly extensible object-oriented frameworks are available that assist in the implementation of distributed search infrastructures, thus addressing the first problem. In order to address the second problem, searchability has to be designed into the online applications constituting the deep web, and integrations to the distributed search infrastructures have to be implemented. A model-driven approach to software construction can be used to specify an application's searchability. This paper presents an extension to the UML that can be used to specify an application's searchability in an efficient way. The resulting models can be used to generate large parts of the searchability implementation automatically.

1 Introduction

A fast growing share of the publicly available web content is no longer being served from static HTML documents but rather from online applications that are often database-driven. Given this trend it turns out that more and more highly relevant web content [1] appears in the so-called "deep web" that is not amenable to search engines' crawlers which are still based on a paradigm that assumes a static web.

Existing search infrastructures, e.g. [17,13], many of them leveraging the benefits of object technology and thus by far exceeding simple low-end protocols like HTTP / HTML in functionality and extensibility, can be used to make deep web contents searchable. The content providers have to contribute by means of providing an implementation that adapts their content to the search infrastructure, making their specific information searchable in the ways they want it to be.

An architecturally solid approach is to make *searchability* an integral part of the overall application architecture like it has become common for persistence, distribution, and transactionality. Using a model-driven approach to application

Z. Bellahsène, D. Patel, and C. Rolland (Eds.): OOIS 2002, LNCS 2425, pp. 199–210, 2002.

development, searchability can be integrated into application models, as is already done today for the other architectural aspects mentioned above [8]. Not only will this enable automatic integration with object-oriented, global Internet search infrastructures as will be shown in this paper, but also it will become possible to utilize this model information for automating the implementation of information retrieval support within the application.

We have organized this paper as follows. First we explain the concept of searchability in the context of Internet applications. We describe the New Wave Searchables framework offering an object-oriented infrastructure for distributed Internet search. Section 3 gives an overview on model-driven development and lists its benefits for application development. In section 4 we show how to combine both approaches: model-driven development and modeling searchability. Section 5 exemplifies these concepts by sketching the model-driven development of J2EE systems and then adding searchability support for a selected search infrastructure. After presenting existing and related work we finally evaluate our approach, summarize the main ideas and findings, and give an outlook on issues for future research.

2 Searchability

2.1 Definition and Problem Statement

The *searchability* of data and applications can formally be defined as a function taking a query as argument and producing a (potentially empty) set of results. Different searchability definitions may accept different kinds of queries and may relate the results to the queries in different ways, even for equal kinds of queries.

Something is said to be *globally searchable* if it can be searched by submitting a query to a general search engine that claims to search the whole Internet.

The largest share of web content today is brought online by complex, database-driven applications exhibiting a web front-end. Many of these applications are comparable in their functionality and complexity to usual desktop applications.

These web application architectures break the assumptions of typical web search engines of a statically linked and crawlable web. For example, URLs, which were used to identify *documents*, are abused by attaching information about the application's state, like a session identifier. HTML is no longer used only to represent documents containing the requested information, but instead is overloaded with presentation issues like frame layout, popup window instructions, JavaScript animations for menu or tree displays and input validation, etc. Instead of providing hyperlinks to all information that the application makes available online, in many cases HTML forms are used that, when submitted, dynamically produce HTML documents.

Therefore, search engines are usually unable to index the contents of these types of Internet applications. Instead, Internet applications have to define explicitly which data are searchable in which ways, and they have to implement their specified searchability.

2.2 Approach to Improved Application Searchability

It is specific to each application which types of queries the application may answer and how these queries are applied to the content and processes that are brought online through the application. Regarding the architecture of the search infrastructure, centralized approaches to Internet search have repeatedly been reported to fail regarding scalability issues [10,18]. Instead, architectures that distribute index information and query processing can be implemented in much more scalable and efficient ways.

Such architectures consist of protocol and interface specifications that govern how queries are transmitted, received, and routed to the searchable sources, how results are retrieved, ranked, and merged, how queries can be transformed, and how query capabilities of searchable sources are formally described. Several different such architectures have been conceived and implemented over the course of various research and industry projects. Section 2.3 will present one that will be used as example for the remainder of this paper.

2.3 The New Wave Searchables Framework

In [17,16] the *New Wave Searchables* framework for object-oriented, distributed Internet search has been presented. It combines best practices from many research projects in the field of distributed search technology and implements them using Java technology which lends itself well to the implementation of a distributed object-oriented infrastructure. It constitutes a search architecture in the above sense.

Its key abstraction is the interaction between four object types: *Searchable*, *Query*, *Production*, and *SearchResult* (see figure 1). Searchables specify their query capabilities using Production objects. Productions can tell if they match a Query object. Query objects that are understood by a Searchable are sent to it, and the Searchable can produce zero or more SearchResult instances.

Using inheritance, new *Query* subtypes as well as specialized *Searchable* and *Production* implementations can be plugged into the framework in intuitive

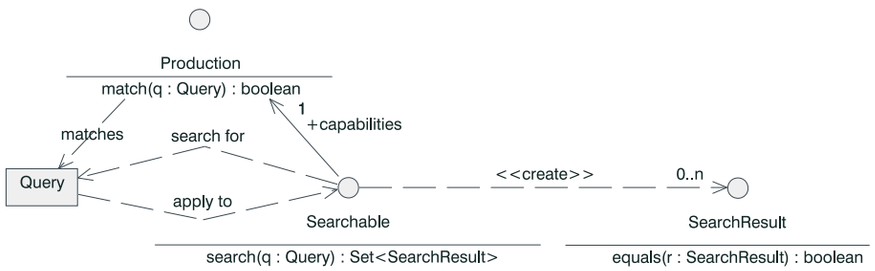

Fig. 1. UML model of the top-level abstractions of the *New Wave Searchables* framework

ways. This allows developers to integrate new ways of searching information and searching new types of media at any time. Java's RMI subsystem transparently manages all relevant issues regarding polymorphic remote method calls and even the transmission of implementation byte code for specialized value-type classes over the network.

The framework supports query transformation. Each query type may implement transformation algorithms that receive as input a *Production* instance describing the search capabilities of a searchable source to which to apply the query. The query can tell whether or not it may transform itself such that the resulting query or queries can be processed by the searchable source.

Query routing, result ranking and merging is implemented in the New Wave Searchables framework by so-called *Traders*. A trader implements a *Composite* pattern on searchables.

At `http://www.NewWaveSearchables.com` a prototypical implementation is online that demonstrates how the concepts can be integrated with existing web technologies. Queries can be created and submitted using a web frontend, searchable web sources are presented as wrapped New Wave Searchables objects, and search results are displayed in HTML documents. The New Wave Searchable framework is also maintained as an active open source project on SourceForge (`http://search.sourceforge.com`).

3 Model Driven Development

In 2001 the Object Management Group (OMG) started an initiative named *Model-Driven Architecture* (MDA) [14,9]. It suggests using *models* to describe software systems at various levels of abstractions, where the models are always held consistent with each other. The software development process benefits from the easier specification at appropriate abstraction levels (see figure 2, left), the increased portability of system specifications, and the improved readability of specifications that serve as additional up-to-date documentation of the system. Figure 2, right, illustrates the increase in portability.

Precise specifications of *mapping techniques* that describe how to transform models between the different abstraction layers ensure that all models for one system are mutually consistent and non-contradictory. A *mapping* is the actual execution of a mapping technique. It may use input in addition to the source model(s), called *annotations*. These make it possible to mark-up a model for a specific mapping technique without making the model itself specific to this technique.

A chain of sets of metamodels leading from abstract to detailed specifications of a system, together with the corresponding mapping techniques is called a *modeling style*.

Models are instances of metamodels. The work on the Meta Object Facility (MOF) [5] provides definitions of models, metamodels, and their mutual relations. Metamodels may be arranged along abstraction and refinement relations and describe aspects of a platform. For example, the Java programming language

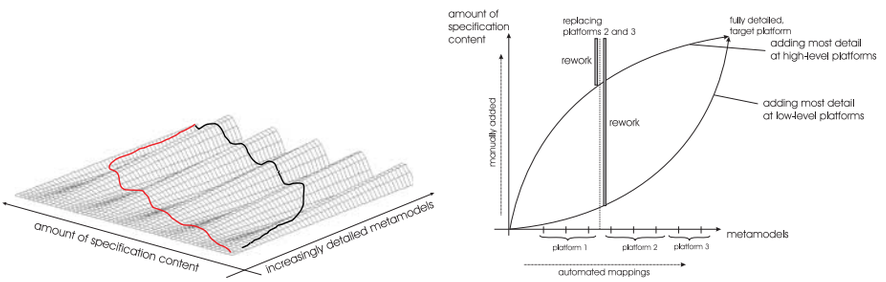

Fig. 2. Left: Development efficiency. The same amount of specification content usually can be provided much easier at higher levels of abstraction while transformations into more detailed levels can be automated, working in favor of the left path, "crossing the hills where they are lower". Right: Specifying as abstract as possible (above curve) increases portability and reduces the amount of rework in case of changing a platform decision or developing for multiple platforms. Too much too detailed specifications (lower curve) result in increased porting effort

specification together with the set of standardized APIs form the metamodel for the Java platform.

It is an important achievement of the MDA initiative to abstract from existing automated model transformations like programming language compilers, and extend this notion into the realm of more abstract models of a system, like Unified Modeling Language (UML) [12] models or Class-Responsibility-Collaboration (CRC) card models [19]. Within MDA, generation of source code and other text-based artifacts of a software system becomes merely a special case of more general transformations between arbitrary models. However, text-based documents can themselves be regarded a model in the context of MDA, and hence can be used as input or output of model transformations.

4 Combining Search Architecture and Application Architecture

Distributed search infrastructures define a technical architecture for parts of a software system: the *search architecture*. It defines the protocols, interfaces, and semantics that can be used to implement an application's *searchability* (see again section 2). For example, the New Wave Searchables framework defines a search architecture by means of its top-level types and interfaces *Searchable*, *Query*, *SearchResult*, and *Production* and the protocol it uses for communication between the distributed components (Java RMI).

Typically, a search architecture cannot stand alone but needs to be integrated with the architecture that an application is built with: the *application architecture*. The application architecture defines the nature of the entities that are to be made searchable using the search architecture.

According to IEEE Std. 610.12-1990, an architecture defines the *organizational structure (the static and the dynamic view) of a system*. In the context of software development, an *application architecture* specifies a template structure or a blueprint for a class of applications in terms of e.g. layers, components and their interrelationships, and the corresponding development infrastructure. Application developers have to ensure that concrete architectures conform to the application architecture.

A search architecture and an application architecture can be combined, resulting in a *platform* in the sense of MDA as described in section 3. Hence, common metamodels can be found, making it possible to create models of searchable applications, serving as specification of both the core application *and* its searchability.

A technology platform supporting searchability with a given search infrastructure typically consists of the *core application* implemented in the application architecture, e.g. a component-based J2EE or .NET environment, that is extended by an adaptation layer that mediates between the core application and the search infrastructure. Figure 3 illustrates this setup. The adaptors access the application's core and adapt it to the search infrastructure.

The combination of the application architecture, the search architecture and the *search adaptor micro-architecture* form the platform on which the search-enabled application gets deployed. The benefit in having specified this platform is that it becomes possible to define MDA support for it. This includes the definition of metamodels that allow for the creation of models for this platform at appropriate levels of abstraction and corresponding mapping techniques that can be automated. By being able to use appropriate abstraction levels for specifying an application's searchability and having automated mapping techniques handle the transformation into source code, the search-related functionality can be developed much more efficiently.

5 Defining a Modeling Style for Searchability

In this section, excerpts from an existing modeling style for J2EE systems are presented. It is then shown how this modeling style can be extended to support modeling searchability in such a way that an adaptor layer for the New Wave Searchable framework can be generated mostly automatically from the models.

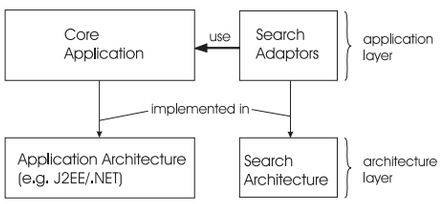

Fig. 3. Extending an application architecture by searchability

5.1 A Modeling Style for J2EE Systems

The modeling style for developing J2EE systems described here is a small subset of the style used by *ArcStyler* [4], limited to those aspects that are required to demonstrate the integration of searchability support with this style.

At the most abstract level, the system is described using CRC cards with responsibilities [19] which, due to space limitations, is not described in any more detail here.

An automated mapping technique transforms the responsibility-driven models into technical UML models. Each card becomes a UML class, and the generalization relationships between the CRC cards are mapped to UML generalizations. Responsibilities and collaborations are, based on model annotations, mapped to one or more of the following UML metamodel elements: attributes, operations, and association ends (roles).

In the UML model several technical properties of these model elements can be specified, for example the multiplicities and navigability of association ends or solely technically motivated inheritance relationships.

Next, the UML model is annotated for a mapping technique transforming it into a set of Java source files, deployment descriptors, Java IDE project files, ANT (see `http://jakarta.apache.org/ant/`) build support scripts, and SQL scripts. This includes the specification of a component's transactional behavior.

Eventually, the annotated model is used as input to a mapping technique producing all of the abovementioned output. The generated Java IDE project files can be used to modify the generated source code in order to insert "if-then-else" logic in marked areas.

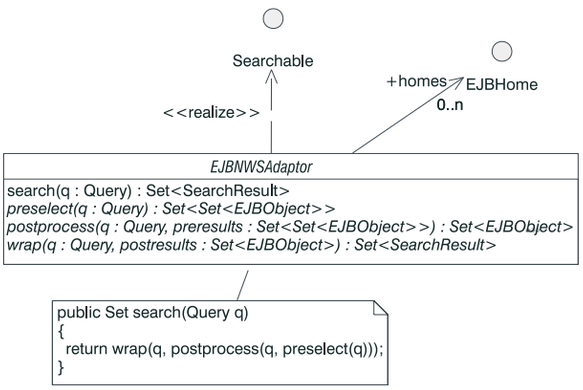

Fig. 4. Adaptor between New Wave Searchables *Searchable* interface and J2EE/EJB architecture. Angle brackets denote parameterized type instantiations

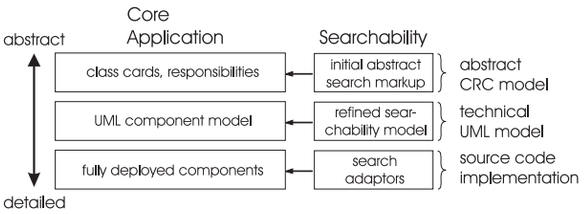

Fig. 5. Modeling a searchable system at different levels of abstraction

5.2 Combining J2EE and New Wave Searchables Architectures

For clarity and brevity of this example, a simple micro-architecture is chosen for the adaptors that mediate between the New Wave Searchables and the J2EE architectures, shown in figure 4. The abstract class *EJBNWSAdaptor* implements the *Searchable* interface provided by the New Wave Searchables framework. At the same time it references one or more *home interfaces* of EJB components that it makes searchable.

The default implementation of the `search` operation performs the search in three steps:

pre-selection Zero or more finders on any of the home interfaces that the adaptor references are called (basically a query on the corresponding component's extent, exposed as an operation of the life-cycle-managing home interface). The parameters for the finder calls are retrieved from the passed query object. The collections of EJB remote references that each finder call produced are returned.

post-processing The results returned by the finders may then be processed further. This may be necessary because the set of available finders may not be sufficient to implement the desired query semantics. Furthermore, if more than one finder was called in the pre-selection step, the results have to be combined, e.g. by intersecting (*AND*-semantics) or uniting (*OR*-semantics) the result sets.

wrapping In most cases, the `wrap` method will produce *SearchResult* instances that contain only those pieces of information from the found instances that are supposed to appear as visible and accessible part of the results.

5.3 A Simple UML-Based Metamodel for the Combined Platform

Searchability can be supported by corresponding metamodel extensions at all levels of abstraction provided by the J2EE modeling style, as depicted in figure 5. The extensions for the UML level are provided as an example.

Figure 6 shows the extensions to the UML-based metamodel used for specifying the system at the technical level. *SearchableClass* is a metaclass whose instances model a single search adaptor, each. For each adaptor a single query type must be specified. Furthermore, a set of finders to be called when receiving

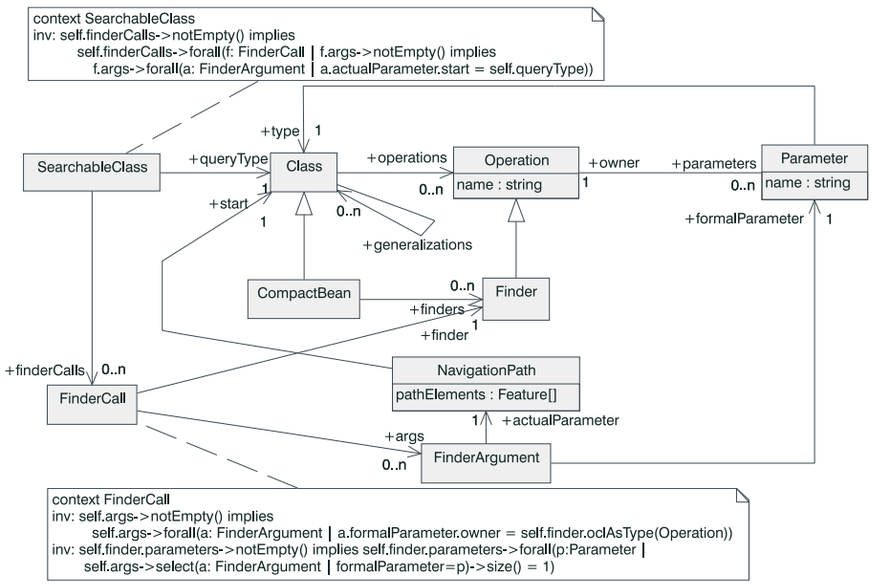

Fig. 6. Example: metamodel for describing searchability of a J2EE architecture

a query of the specified type has to be associated with the adaptor. For each finder the model must specify how the finder arguments are retrieved from the query. It does so by providing *NavigationPath* instances that describe how to retrieve the value to be used as finder argument, starting on the query object, and navigating along features (attribute, associations, operations) of the query.

Specialized physical components can be used to group multiple *SearchableClass* instances together that are assigned as residents of the component.

5.4 Source Code Level

An instance of the extended UML metamodel sketched in section 5.3 can be used to generate corresponding source code that provides sections where the developer has to add more detail. Each *SearchableClass* instance is transformed into a Java class extending the *EJBNWSAdaptor* class (see again figure 4). An implementation of the `getSupportedQueryTypes` operation is generated that returns a *Production* that matches queries of the query type specified in the model for the *SearchableClass*.

In the constructor of the generated class, the home interfaces of those EJBs whose finders are used by the *SearchableClass* are resolved and stored in the `homes` role. An implementation for the `preselect` method is generated that calls the finders on the homes as specified by the model, retrieving the arguments from the passed query and returning the results returned by the finders.

In the `postprocess` operation, a default implementation is generated that intersects all finders' result sets. The developer may modify this default to meet special needs. The same customization is possible for the `wrap` operation, where the generated default implementation returns the references to the EJBs, wrapped as `SearchResult` objects.

From the specialized physical components containing *SearchableClass* instances as their residents, source can be generated that instantiates the *Trader* class from the New Wave Searchables framework, adding one instance of each *EJBNWSAdaptor* that was created from each of the residents to it. By default, the resulting trader supports the combined set of query types and can be used to register the application with a search engine.

6 Related Work

Concepts for a distributed search infrastructure based on CORBA have been developed, e.g., in the *InfoBus* [13] project. This and similar projects contained many excellent approaches, prototyping wrappers for existing information sources and demonstrating how CORBA helps in solving the challenges of distributed systems development leveraging the benefits of object technology. However, they have not addressed other important aspects like showing how different query types can be applied in the presence of heterogeneous data sources that support different sets of query types.

Garlic [3] is a project that has conducted research in the area of information retrieval on heterogeneous multi-media data sources. Garlic uses an object-oriented model to represent data and queries. One task was query rewriting in the context of an extensible query type set [7], using search capability descriptions and query execution cost models for the participating searchable data sources. Given these descriptions and models a query could be mapped to a cost-optimal execution plan using standard planning algorithms. Garlic does not address issues like integerating the query type framework with web frontends and the problem of handling large numbers of searchable collections.

Another approach to heterogeneous and distributed search has been researched in the *DISCO* project [15]. In DISCO the search capabilities of the data sources are described as grammars for the queries. All DISCO-enabled sources have to be capable of delivering *all* their retrievable instances which can be prohibitive for huge data collections. DISCO, like Garlic, also uses a cost model for query execution. A special feature of DISCO is that it can reasonably deal with temporary unavailability of data sources. Web integration of the framework was not discussed.

Other existing approaches that define distributed search infrastructures, and that cannot be discussed in detail due to space limitations are *Lexibot* (`http://www.lexibot.com`), Apple's *Sherlock* [11], and *Grub* (`http://www.grub.org`).

The general ideas of model-driven software construction are combined in the OMG's work on MDA [14,9]. Many application- or domain-specific modeling

styles have been created, e.g. for multimedia or real-time applications. [2] provides an overview. Examples can also be found in [6].

7 Conclusions and Future Work

In this paper we have presented a model-driven approach for search-enabling Internet applications that constitute parts of the deep web. This was achieved by combining an application architecture and an object-oriented search architecture into a platform that can then be supported by the Model-Driven Architecture (MDA). A chain of metamodels and corresponding mapping techniques have been presented that enable developers to specify an application's searchability in convenient and portable ways.

The automation of mapping techniques reduces the implementation effort to a minimum. As a result, making deep web applications searchable using a powerful search infrastructure that leverages the benefits of distributed object technology becomes straightforward, easy, and intuitive, which in turn gives rise to hopes that an increasing share of deep web information will be globally searchable in the future.

Future research will have to focus on improving the metamodels with regard to their applicability to a wide range of combinations of application and search architectures. The example presented here assumed that a set of query functions (*finders* in the case of EJB) are available that the model can refer to. Intelligent target-technology-aware mappings may be found that create default sets of such query operations. We will investigate to what extent portable specifications of search logic details are possible by defining mapping techniques to other object-oriented search infrastructures.

References

1. Michael K. Bergman. The deep web: Surfacing hidden value. July 2000. URL `http://128.121.227.57/download/deepwebwhitepaper.pdf.` 199
2. Margaret Burnett. Visual language research bibliography. URL `http://www.cs.orst.edu/~burnett/vpl.html.` 209
3. M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The garlic approach. In *Research Issues in Data Engineering,* pages 124-131. IEEE Computer Society Press, Los Alamitos, Ca., USA, March 1995. ISBN 0-8186-7056-8. URL `http://www.almaden.ibm.com/cs/garlic/ride-dom95.html.` 208
4. Interactive Objects Software GmbH. ArcStyler User's Guide. URL `http://www.io-software.com/products/docu/Users-Guide.pdf.` 205
5. The Object Management Group. The MOF specification version 1.3, March 2000. URL `http://www.omg.org/cgi-bin/doc?formal/00-04-03.` 202
6. John C. Grundy and John Hosking. High-level static and dynamic visualisation of software architectures. In *IEEE Symposium an Visual Languages, Seattle, WA,* USA, September 2000. URL `http://www.cs.auckland.ac.nz/~john-g/papers/v100.ps.gz.` 209

7. Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing querfies across diverse data sources. In *Proceedings of the Twenty-third International Conference an Very Large Databases,* pages 276-285. VLDB Endowment, Saratoga, Calif., Athens, Greece, August 1997. URL `http://www.almaden.ibm.com/cs/garlic/vldb97opt.ps.` 208
8. Richard Hubert. White Paper: Convergent Architecture & The ArcStyler Tool Suite, 2000. URL `http://www.io-software.com/products/docu/iO_CA_ArcStyler_Whitepaper.pdf.` 200
9. Thomas Koch, Axel Uhl, and Dirk Weise. Model-Driven Architecture, January 2002. URL `http://cgi.omg.org/cgi-bin/doc?ormsc/02-01-04.pdf.` 202, 208
10. Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science,* 280(5360):98, 1998. URL `http://www.neci.nj.nec.com/~lawrence/science98.html.` 201
11. John Montbriand. Extending and controlling sherlock. November 1999. URL `http://developer.apple.com/technotes/tn/tn1141.html.` 208
12. The Object Management Group (OMG). The unified modeling language, version 1.4, September 2001. URL `http://www.omg.org/cgi-bin/doc?formal/01-09-67.pdf.` 203
13. M. Roscheisen, M. Baldonado, C.-C. K. Chang, L. Gravano, S. Ketchpel, and A. Paepcke. The stanford infobus and its service layers: Augmenting the internet with higher-level information management protocols. In *Digital Libraries in Computer Science: The McDoc Approach, LNCS,* volume 1392. Springer, 1998. URL `http://www-diglib.stanford.edu/diglib/WP/PUBLIC/DOC148.pdf.` 199, 208
14. The Object Management Group (OMG). Model Driven Architecture: The Architecture of Choice for a Changing World, 2001. URL `http://cgi.omg.org/cgibin/doc?ormsc/01-07-01.` 202, 208
15. A. Tomasic, Louiqa Raschid, and Patrick Valduriez. A data model and query processing technique for scaling access to distributed heterogeneous databases in DISCO. *IEEE Transactions an Computers, special issue an Distributed Computing Systems,* 1997. URL `ftp://ftp.umiacs.umd.edu/pub/ONRrept/IeeeTOCS96.ps.` 208
16. Axel Uhl and Horst Lichter. New Wave Searchables: Changing the paradigm of Internet scale search. In *International Conference an Advances in Infrastructure for Electronic Business, Science, and Education an the Internet.* SSGRR, L'Aquila, Italy, August 2001. ISBN 88-85280-61-7. URL `http://shipping.accesscable.net/uhl/SSGRR.PDF.` 201
17. Axel Uhl. The future of Internet search. In Roberto Baldoni, editor, *DOA'01 International Symposium an Distributed Objects and Applications, Short Papers,* September 2001. ISBN 888665811-7. URL `http://shipping.accesscable.net/uhl/DOA2001Short.PDF.` 199, 201
18. Axel Uhl. A bandwidth model for internet search. In *Proceedings of the 28th International Conference an Very Large Data Bases (VLDB '02).* Morgan Kaufmann, Orlando, September 2002. URL `http://www.vldb.org/conf/2002/P687.pdf.` 201
19. Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing ObjectOriented Software.* Prentice-Hall, Englewood Cliffs, NJ 07632, 1990. 203, 205