

# Towards Adaptive Knowledge Middleware

Raphaël MARVIE\*, Lev KOZAKOV, Yurdaer DOGANATA

IBM T.J. Watson Research Center  
IBM Corp.  
Hawthorne, NY, U.S.A.

## 1 Introduction

### Abstract

*The amount and the heterogeneity of the data produced by organizations are increasing at an unprecedented rate. The need to store, mine all this data and search and retrieve the information content is becoming a more critical task than ever since the success of the organizations depend on how effectively they can access this data. As the number of data sources increases so is their heterogeneity, and in this sea of knowledge retrieving relevant information becomes harder and requires designing and developing adapters to access the information with custom interfaces. This paper focuses on retrieving information from heterogeneous data sources without having to write code specific for each data source. It discusses the use of technologies such as XML, WSDL, and UDDI to achieve this goal, and has to be seen as a first step towards autonomous computing in data retrieving through the use of an Adaptive Knowledge Middleware.*

**Keywords:** Knowledge Middleware, Web Services, XML, Flexibility.

---

Raphaël Marvie is currently member of the *Laboratoire d'Informatique Fondamentale de Lille (LIFL), UPRESA CNRS 8022, University of Lille 1, France.*

The content of the organizations' *sea of knowledge* is increasing continuously with different data sources that are extensively used in everyday business. These data sources range from traditional data bases containing, for example, product stocks or customers, to more recently developed e-mail and intranet servers. Each of these data sources has its own internal structure of the data, which may be very simple, like an e-mail or an HTML page. To access this information, one has to use adapters specifically designed to be used interact with a given data source: a customized database data source client and a web browser application coupled with a specific search engine to retrieve and view HTML pages.

Advancement of the Internet technologies made a unified user interface to all these information sources possible, hence hiding from the users the complexity of the back end designs are. A single entry point to the system can easily be offered from a Web browser. Once the user requests are collected via a browser, however, passing those request to the data sources requires specific adapters for each data source with a specific structure, so responding to a query using retrieving information from several data sources is challenging. In other words, providing a unified access to various data sources, collecting and merging the results, and integrating to the front end to provide relevant information is not easy. Moreover, each time a new

data source has to be integrated in the information system, the application that provides access to the *sea of data* has to be modified, and its complexity increases.

Accessing and processing information is one of the most important tasks in an organization. This paper intends to present a means to provide for a single and unified access to an organization's *sea of knowledge* which contains various kinds of datasources that are often completely different in their internal structures. Thus, in most cases each datasource comes with its own search utility.

When accessing some disperse sections of the *sea of knowledge* is desired, various entry points to the system have to be used to collect the various elements of the required information. Thus, writing an application that searches, finds and delivers accurate information is not easy and, moreover, much of these applications are hard to maintain as new datasources appear. This paper aims to introduce a technique in building flexible adapters to be used in designing a knowledge middleware using standard technologies. While technical middleware provides a means for parts of applications to cooperate, a knowledge middleware provides access to information. RPC based middleware—like CORBA [4] and RMI [6]—has to comply to technical interfaces, a knowledge middleware has to comply to the activity ontology. The technique that is presented in this paper reduces the complexity of developing applications development process that need to access and process information contained in the *sea of knowledge*.

The purpose of the technique presented here is not to define a universal new search operation, but to unify the use of existing ones that are dedicated to specific datasources. Then, such a data access architecture becomes an entry point to the *sea of knowledge* that would be used by any application. The design of flexible adapters is intended to be generic in order to suit any application, and also configurable in or-

der to meet the applications specific requirements. In a longer-term vision these flexible adapters will also become self-tuning adaptive, meaning it they would be able to tailor itself themselves to the application use in order to improve performance of information retrieval. In all that, this paper tends to address issues related to scalability, system integration and evolution, as well as integration of knowledge, raised in [1].

This paper is organized as follows. Section 2 discusses the concept of adapters for data access in knowledge middleware and a method of design. Section 3 presents the design and architecture of the current implementation of our knowledge middleware that supports flexible data source adapters. Section 4 discusses the technologies that we use to implement our knowledge middleware. Finally, Section 5 concludes this paper and outlines some perspectives.

## 2 Towards an Adaptive Knowledge Middleware

### 2.1 Data access in Knowledge Middleware

The goal of a knowledge middleware is to provide a single and unified access to multiple data sources, ability to pre-process to queries, post-process the results, merge, rank, group and display them. To do so, it collects unstructured data from available datasources and provides it as structured information to applications. One of the most important components of a knowledge middleware is the adapters to access providing access to datasources. In the design of such adapters, there are two important considerations. First, the heterogeneity of various data sources must be hidden by the middleware, hence the knowledge management application ought not to be modified every time a new source needs to be accessed. Second, the integration of new datasources should not in-

roduce complexity in the application itself. As soon as a data source becomes part of the *sea of knowledge*, the middleware should be able to use it and to provide its content to applications. Consequently, the sharing of information is increased through by integrating new data sources rapidly.

## 2.2 Static *vs* Flexible Adapters for Knowledge Middleware

The adapters for knowledge middleware may be designed by using two approaches: A static one and a dynamic one.

### 2.2.1 Static Adapters for Knowledge Middleware

In the context of a static data adapters, the lower layer of the middleware, which is connected to the datasources, is written specifically for the datasources it uses. Thus, this lower level is strongly dependent of the datasource, and has to evolve with it. Then, it may be complex to write and the complexity of the middleware increases with the number of datasources it uses. Moreover, each part of the lower level may not be used for any other datasource than the one it has been designed for. In that, the knowledge middleware in its static approach is a day-to-day answer to a global fast evolving problem—the growth of the *sea of data*—and it becomes less open for evolution and more complex to manage each time a new datasource appear. The problem is only shifted from the application- to the middleware level.

### 2.2.2 Flexible Adapters for Knowledge Middleware

While static data adapters do not require anything from datasources, flexible knowledge middleware introduces the use of ‘scope’ for datasources. A data source is not required to have a specific interface (meaning reduced to a particular technology) but has

to comply with a pre-defined scope - the ontology of the application domain. Meaningful ontologies have been and will be defined for every information source to designate given concepts of its domain (see Semantic Web Activity [11]). As long as the information provided by the datasource complies to this the pre-defined ontology, the knowledge middleware may use it. In that, first the datasource related code is more general and may be used in various contexts, and second the integration of new datasource in the system is easier and faster: once the datasource is available and known to the middleware, it is usable for applications.

### 2.2.3 Adaptive Knowledge Middleware

Flexible knowledge middleware is an improved answer compared to static ones, however it only represents a first step. As the need to write dedicated pieces of software no longer exists, the integration of datasources could no longer be performed manually. Adaptive knowledge middleware’s goal is to automate the discovering and the use of datasources as they are introduced in the system. While the *sea of data* would grow transparently to the applications, they would be able to benefit from it.

An adaptive knowledge middleware has to be highly configurable and reflexive. First, dynamic configurability is required for critical software (i.e. which cannot be stopped for administration purpose). Each time a datasource is introduced, the knowledge middleware has to serve application request while integrating the new datasource. Second, the use of reflexivity in a knowledge middleware reifies important information about the knowledge middleware. Then, it is possible for the middleware to tailor itself according to the context. When a new datasource appears, the strategy (in the sense of the strategy pattern [2]) of using the set of datasources may evolve. This is important, for example, if two datasources provide similar information: should the knowledge

middleware use both of them merging the results or one at a time?

Finally, the behavior of the middleware would be able to evolve according first, to the various datasources, and second, to the use of these datasources by the applications. (This last point will not be discussed here as it is part of the perspectives.)

### 3 A Knowledge Middleware Implementation

#### 3.1 Overview

The current implementation is mainly made of two parts: A runtime and XML schema.

The runtime is provided as a library that is to be used by applications requiring access to the *sea of data*. It provides a simple interface to send a query and to retrieve structured information. This structured information is returned as a Document Object Model (DOM) [8] tree that points to documents matching the query. Then, this tree is easily usable by the application. Basically, the runtime is quite generic and do not perform application level related processing. However, the runtime is also highly configurable, and as an instance of the runtime is used by one application only, it may be customized to perfectly match the application requirements. As an example, the behavior of the runtime may be tailored for the needs of the application.

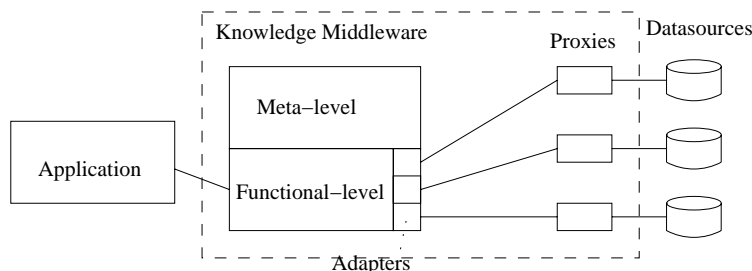
The XML schema, associated with the runtime, is related to the domain of the application, in our context, retrieving documents. Its purpose is simply to define an ontology shared by the application and the datasources. In addition to defining basic types — such as document title, url of the document, or summary of the document — it provides a base definition to express a query and a hit — the information describing a document that matches the query. These

two types are provided as base type that may be extended — like when datasources provide more information than the one described in the base type — but only according to the definitions contained in the XML schema. At the moment, only one XML schema may be used by an application at a time. The runtime will behave relying upon this schema to make responses from the datasources understandable to the application.

#### 3.2 Runtime Architecture

The runtime architecture is composed of two levels: the functional one and the meta one (see Figure 1). The functional level is the one used by the applications. It provides the operations to perform the search. This level does not hard code any behavior of the runtime. It only knows where the behavior is defined (at the meta level) and uses it to process the requests of the application. On the datasource side, a proxy for the data source has to be made available. This proxy is dependant of the data source internal format and is intended to provide information according to the XML schema definitions. The point in having this proxy on the datasource side is that it may be used in several contexts (not only by the knowledge middleware). On the runtime side, an adapter is connected to the datasource proxy on a one-to-one basis. This adapter will process part of the query sent by the runtime: The request is splitted between several adapters to multicast to the datasources. Some processing, like filtering, may be performed by the adapter too.

The meta level does not only provide define the behavior to of the functional level at runtime, it also makes it available for introspection purpose. Then, it is possible to discover the configuration of the knowledge middleware while it is running. For example, one can discover which strategy is used by a given adapter, which datasource it is connected to, or some technical information like the time it took for processing the last re-



**Figure 1. Architecture of the Knowledge Middleware Runtime**

quest. Moreover, without any impact on the application—which is unaware of the architecture of the knowledge middleware—it is possible to change the KM configuration dynamically at runtime. Thus, even the application becomes flexible and can evolve easily.

An administrator can change the behavior of the knowledge middleware while in use by the application. It can add new datasources, or change the configuration of adapters. In order to avoid non-coherent state of the knowledge middleware, when an adapter is performing a request, it asks the meta-object to define its behavior according to the meta-level instructions. In the meantime, the administrator can create a new behavioral meta-object to replace the existing one. The adapter will not be aware of this shift until the next time it requests its behavior from the meta-level.

### 3.3 Prototype & Example

The runtime prototype has been developed in Java using IBM Web Service Toolkit (WSTK) [3] as a support to use WSDL and UDDI technologies (see Section 5). It is a small library to be linked to any application that needs to access the public datasources of an organization. It has been evaluated with a simple application providing a search interface to users. This simple application do not perform any processing on the results of the search, it asks the users for keywords and some parameters for the search and then

simply displays the results found in several datasources.

In the meantime, a basic administration oriented console has been developed. This console allows one to know the actual configuration of the knowledge middleware. It also provides operations to change its configuration: Looking for new datasources, changing the behavior of the knowledge middleware, changing the behavior of a specific adapter, and so on. All these operations are performed dynamically, without stopping the application or suspending the service. Nevertheless, they may introduce little delays perceptible by users.

## 4 Use of Open Technologies and Standards

### 4.1 XML

The eXtensible Markup Language (XML) represents one of the most open ways to structure information. It is relying on standardized recommendations of the World Wide Web Consortium (W3C). It allows applications to exchange information in a standard way, regardless of the way they internally structure this information. Moreover, XML is easily understandable and usable as many XML parsers are nowadays available for most popular programming languages. Finally, XML Schemas [10] provide a means to define the structure of an XML document: as long as two applications share this knowl-

edge, they will be able to produce and process compatible documents.

In the context of our knowledge middleware, XML Schema are used to define in an easy manner a simple ontology used in the context of document searching. It defines the structure of a query as well as the structure of matching hits and sequence of hits. As XML Schemas permit the use of inheritance, the definition of queries and hits are not strongly fixed, they may vary according to the datasources characteristic. However, the scope of variation is also defined in the XML Schema. So, enriching the hit structure will not break it for applications that are not aware of the extended elements.

## 4.2 WSDL

The use of Web Service Definition Language (WSDL) [9] intends to define web services regardless of their implementation, even of the technology used for their implementation. WSDL relies upon XML to describe a web service: The operation it provides, the argument and return values of these operations, as well as the location of the web service (the port where to connect) and the technology solution one needs to use in order to benefit from the service. Most of the time, web services are used through the Simple Access Protocol (SOAP) [7], but this is not mandatory. Like XML, WSDL may be used in any environment and with any programming language.

WSDL is used in the current context to define and easily produce datasource proxies that will give access to the datasource information, providing it according to the XML Schema definition. In that, the datasource will be usable through the knowledge middleware, but the proxy may be used by other applications also. Then, the web service could become the only entry point to access the datasource.

## 4.3 UDDI

The Universal Description, Discovery, and Integration (UDDI) [5] is a new industrial initiative to ease the discovering and integration of business web services in order to create Business-to-Business (B2B) applications. The UDDI registry is a trading function that permits service providers to publish their services, and users to search / discover services using their characteristics — like using the yellow pages — in order to use them. Compared to name-based search — like using white pages — this brings flexibility and dynamic use of web services: It is no longer necessary to know what is the name or the url of a service to use it.

The UDDI registry is used in the context of our knowledge middleware to publish and to search the datasource proxies. When a datasource has to go publically available, its proxy is published in the UDDI registry. Each time the knowledge middleware runtime is initialized by an application, it searches in the UDDI registry all the datasources available (potentially according to criteria) and try to connect to them. For each proxy found, it launches an adapter to make this datasource part of its known *sea of knowledge*. Then each time a query is sent to the knowledge middleware runtime, the datasources found are used to process the query.

## 5 Conclusion

This paper has presented a solution to find structured information from a *sea of knowledge* in an easy way using a knowledge middleware. It has focus on the fact that using a knowledge middleware, applications become more flexible with a potential to evolve. It is no more necessary to modify the application each time a new datasource is introduced in the *sea of knowledge* in order to use it. Also, the use of open and standardized technolo-

gies for building the knowledge middleware has also been discussed. This allows the integration of heterogeneous datasources, the furniture of a unified entry point, as well as the environment being interoperable with third-parties applications.

At the moment, our knowledge middleware provides a unified entry point to a *sea of knowledge*. It is flexible and highly configurable. Most of the basic mechanisms of such a middleware have been evaluated and implemented. Relying upon these mechanisms, an administration console has been build to provide dynamic reconfigairiton of the knowledge middleware. A simple application to search documents from several datasources has also been developed. Thus, the knowledge middleware is right now flexible but used in non-autonomous fashion.

The basic mechanisms developed are an investment for the future. They represent the basis of forthcoming work which will try to address the issue of automation. How to define and to implement rules that will be automatically applied by the middleware to adapt itself to the application requirements: which datasources are most frequently used, how the adapters should behave, what filters should be applied to results from a given datasource, and so on. In that, this work should be seen as a first step towards an adaptive knowledge middleware.

## References

- [1] A. Aho *Accessing Information from Globally Distributed Knowledge Repositories* In *Proceeding of PODS'96*, ACM, Montreal, 1996.
- [2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Westley Professional Computing, 1995. ISBN: 0-201-63361-2
- [3] IBM *Web Service Toolkit 2.3* IBM Alphaworks technologies, May 2001 <http://www.alphaworks.ibm.com/tech/webservicestoolkit>
- [4] OMG *CORBA/IIOP 2.4.2 Specification*. Object Management Group. OMG TC Document formal/01-02-01, February 2001.
- [5] uddi.org *UDDI Technical White Paper*. UDDI Initiave, September 2000. <http://www.uddi.org>
- [6] A. Wollrath, R. Riggs, and J. Waldo *A Distributed Object Model for the Java System* USENIX Computing Systems, 9, November/December 1996.
- [7] W3C *Simple Object Access Protocol (SOAP) 1.1* World Wide Web Consortium. W3C Recommendation, May 2000. <http://www.w3.org/TR/SOAP/>
- [8] W3C *Document Object Model (DOM) Level 2 Core Specification* World Wide Web Consortium. W3C Recommendation, November 2000. <http://www.w3.org/DOM/DOMTR>
- [9] W3C *Web Services Description Language (WSDL) 1.1* World Wide Web Consortium. W3C Recommendation, March 2001. <http://www.w3.org/TR/wsdl>
- [10] W3C *XML Schema Part 0: Primer* World Wide Web Consortium. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-0/>.
- [11] W3C *Semantic Web Activity* World Wide Web Consortium. 2001. <http://www.w3.org/sw/>.