An Internet Content Integration System: the Mediator and Wrappers.

by

Nikolaos T. Ntarmos

Submitted to the
Department of Electronic and Computer Engineering
in Partial Fulfillment of the Requirements for
the Diploma of Electronic and Computer Engineering
at the Technical University of Crete.

Guidance Committee

Professor Peter Triantafillou (Supervisor)
Associate Professor Manolis Koubarakis
Assistant Professor Euripides Petrakis

April 2002

Acknowledgements

I would like to thank Professor Peter Triantafillou, for supervising this effort and for his guidance and support; they were more than indispensable to the completion of this thesis.

I would also like to thank in advance Assistant Professor Euripides Petrakis and Associate Professor Manolis Koubarakis for participating in my jury.

Special thanks also go to my house-mate, Vassilis Heliades, for his support and patience (sleeping through the day and working at night isn't something everyone could cope with...).

Abstract

Information integration and dissemination over the web is increasingly receiving more attention from academics and related industry. With such emerging enabling technologies as XML and all of it's descendants (XQuery/XML-QL, XPath, XPointer, XSL/XSLT, etc.), we seem to be moving into an era of relative data uniformity. However, the unstructured and heterogeneous nature and the immense proportions of data currently available through the web, make information integration a vital part of many modern data management systems and data warehouses.

The most wide-spread approach to automated data integration and dissemination utilizes a mediator and wrappers for the back-end sites. The former stands between the wrappers and the end-user, while the latter deal with data extraction from the back-end sites and data transformation into the mediator's internal data model.

In this text, we shall present the Content Integration Architecture (C.I.A.) - an approach to a domain-independent caching mediator system - as well as two applications based upon it: HyperHotel and HyperTV.

Contents

List Of Figures

Chapter 1

Introduction

1.1 Overview

The world wide web has evolved to the world's most massive database, but also to the most non-homogeneous one. Several attempts have been made to develop technologies that will integrate related data available online in an automated or semi-automated way and facilitate/provide uniform access to this data.

Some of them aim to implement a domain-based integrator; they make use of artificial intelligence and domain-based knowledge to automatically extract the structure of available data and integrate it.

Others try to create a more generic infrastructure on which numerous domain-based integration applications will be based; they rely on wrapper-generation toolkits and a custom internal design / data model.

What we try to do is to build an independent mediator model, using third-party off-the-self wrappers and/or wrapper generation toolkits to extract information from e-sites, and standard XML-based[1] technologies to integrate them with our data model. The purpose of this work is to automate the task of selectively querying multiple data sources on the web and presenting the results in a uniform way. In later stages of development, the system will make use of caching and distribution techniques for increased throughput and decreased response time to user queries.

The use of third party wrapper generation toolkits removes the burden of the development of this part of a mediator system and allows us to exploit available technology, making our work market-relevant, and to concentrate on the building of the mediator itself. By utilizing XML-based technologies, the only requirement the mediator must face is to deal with XML data, complying to a predefined set of DTDs. Queries can then be formulated in many ways (including XSL, XPath, XQuery and DTD-compliant XML documents).

We will demonstrate the functionality of this system with two applications: HyperHotel and HyperTV; two dynamic mediator-based information integration and dissemination systems for e-hotels and television program listings, respectively.

1.2 Necessity / Current Situation

Due to the diversity of data available online through the World Wide Web, should one want to retrieve information on a specific subject, one would have to search in many different sites, keeping track of search and comparison results while most of the time dealing with outdated and/or obsolete data. This situation calls for a new way of designing and implementing data retrieval systems.

Let's assume, for example, that an individual intends to purchase something online (e.g. a book, cdrom etc.). Using currently available solutions, the process is as follows:

1. Visit all relevant e-sites. This is an inherently inefficient task, since the set of available e-sites constantly changes. This usually results in the individual visiting only a small, random subset of them, therefore excluding important amounts of available data.

2. Search for the wanted item. This step forces the user to deal with many different interfaces across the different sites. Should an e-site not be very well designed or its search engine be stricter than usual, the user would probably fail in finding the desired information.

3. Compare and choose. The choice could be based upon such criteria as the price or the proximity of the e-shop, the available paying methods etc. In any case, the user has to

keep track of all relevant results from the previous step(s) in order to be able to compare and choose.

Let's assume, however, that there were a central e-site, interfacing with all back-end e-sites in a way that is transparent to the end-user. The user would then only have to search once for the desired information; the central e-site would undertake the task of querying all registered back-end sites and presenting the results in a uniform way, so that direct comparison would be made possible, if not completely automated.

The second scenario is obviously much more preferable than the first one, as far as ease of use is concerned, but what about efficiency? Suppose two subsequent users are looking for the same information. Asking the same question twice over the internet would be very inefficient due to the network overhead. There are cases in which identical successive queries should both be run online. However, for all the other cases, by using an appropriate caching scheme, we can reduce the network overhead and response times significantly. In any case, should there be no central e-site, the users would have to individually deal with the network overhead themselves.

The goal of the C.I.A. endeavor is to build a domain-independent, dynamic, mediator system with caching, with an emphasis on speed, platform independence and ease of deployment. For modularity and ease of maintenance, C.I.A. was built as three independent but cooperating parts:

1. a graphical user interface (GUI), based on Java Servlets([2, 3]) and/or JavaServer Pages([4, 5]), HTML forms and XML-based technologies,

2. an XML-enabled mediator, utilizing third-party wrappers and standard XML-derived technologies, and

3. a caching subsystem, based on off-the-self database management systems and novel cache management techniques.

The work reported here is primarily concerned with the second part: the Mediator.

1.3 Thesis outline

We will continue as follows: in Ch. 2 we will discuss the architecture of C.I.A. outlining it's building parts and the way they interface with each-other and with the end-users and back-end e-sites. In Ch. 3 we'll delve into the details of the Mediator's architecture and the technologies used therein, also describing the querying models supported by the overall system as well as the results obtained by extensively testing these models. We will then (Ch. 4) present two data integration applications implemented using our system: HyperHotel and HyperTV, integrating e-hotels and TV program listings from TV e-sites respectively. We shall conclude with a brief presentation of work related to C.I.A. (Ch. 5) and proposals for future work(Ch. 6). Appendices A and B give some more technical details on the implementation of the Mediator and C.I.A. as a whole.

Chapter 2

Architectural Overview

As we already mentioned, C.I.A. is a mediator-based data integration and dissemination application for diverse data on the web. Written totally in Java([6]), it guarantees maximum platform independence and portability. Briefly, C.I.A. is a client/server architecture, consisting of the following components:

- The user client.
- The user interface.
- The query generator.
- The mediator system.
- The data repository.
- The agents.
- The wrappers.
- The back-end e-sites.

A schematic view of this architecture is presented in figure 2.1.

2.1 The user client

This is usually a web browser. Any browser available today (even text-based ones, such as Lynx) can contact the C.I.A. server, using the HTTP/1.1([7]) or higher protocol, HTML 3.0 or

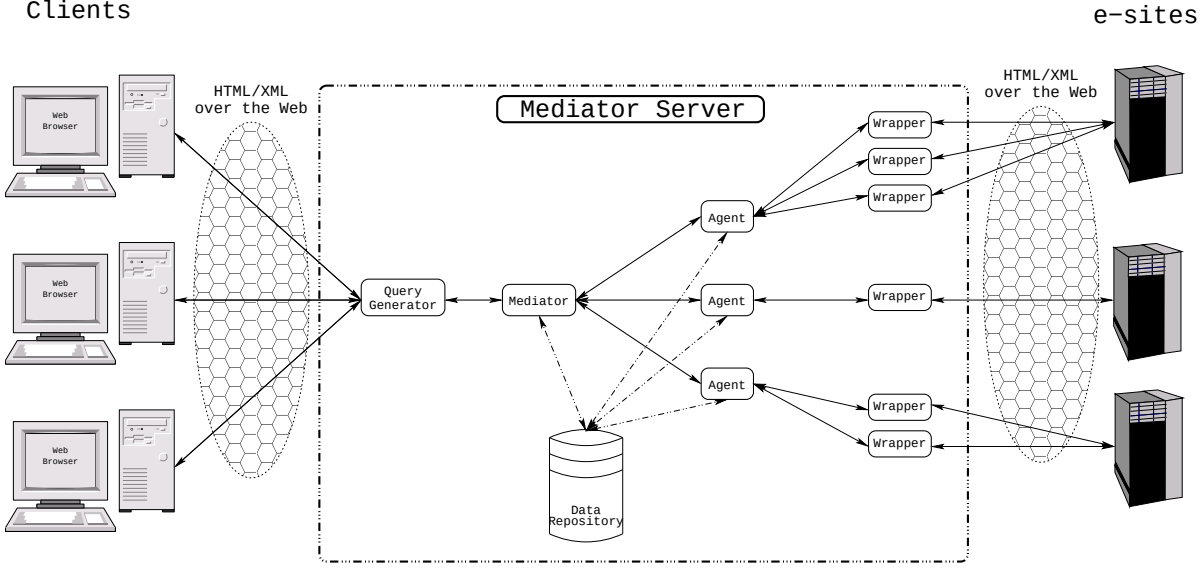


Figure 2.1: Mediator Architecture

higher and Javascript 1.0 or higher; the system uses server-side technologies, therefore moving the burden of almost all tasks into the server. This is a great advantage of the system, since it allows access from lightweight clients, such as PDAs and even WAP-enabled mobile phones.

2.2 The front-end

This is the server-side part of the user interface. A more detailed description of this part of the system is provided in [8].

2.2.1 The user interface

The user interface consists of HTML pages/forms and server-side technologies, such as JSPs and Java Servlets. This frontend is the only interface with the user. We have tried to make it as user-friendly as possible, given that this is an educational project and not a commercial one.

2.2.2 The query generator

This module validates the user query passed to it from the user interface, and generates a query in XML, according to a predefined grammar (DTD). It then forwards this query to

the next part: the Mediator. Alternatively the XML query can be forwarded to the local cache-storage system and the mediator will be engaged by it only in a cache (partial/full) miss.

2.3 The back-end [1]

2.3.1 The mediator

This part is named after its task: it stands between the user-interface and the back-end agents, selecting which, if any, agents to involve in each query, and formatting their output, according to a predefined grammar (DTD), in order to be presented to the user in a uniform way. The selection of the agents is based on meta-data, kept by the mediator, and on the user query.

2.3.2 The data repository

This includes the meta-data database, as well as a (optional) data-cache for e-sites. There are three approaches for the implementation of the data repository:

1. The RDBMS approach, which uses an off-the-self relational database (e.g. MySQL or Microsoft SQL Server). Data access is done through JDBC and XML. This is assumed to be the fastest approach, as far as runtime efficiency is concerned.

2. The native XML approach, which utilizes off-the-self native XML databases, such as Lore([9, 10]), Quilt([11]) and dbXML. Data access is done through vendor-specific APIs or the X-API. This approach is considered by the authors as the best of the three, since it can be platform-independent (if the database is implemented in Java) while using XML-derived technologies, making it easier to code and integrate with the rest of the system. It's only shortcomings are the low runtime efficiency and the lack of commercial support (the above mentioned implementations are far from being stable or ready for production

[1] **Though the mediator and the database could be considered middleware, they are so closely interconnected with the agents and the wrappers that are dealt with as parts of the back-end.**

bound to be extinct.

3. The DOM[2] approach, in which all data is kept in DOM trees, without the support of an underlying native XML database, and parsed at will. This is the worst of the three approaches, as it is memory-consuming (DOM trees are kept in main memory), requires extra time for the parsing of the XML data (should the desired DOM tree not be loaded at the time of the query) and execution times are much greater, since queries are executed on DOM trees, with no query optimization or scheduling. On the other hand, it is the easiest of the three to code for simple queries (we believe that the so-called simple queries, will be the vast majority of all queries that will run on the database).

2.3.3 The agents

Agents decide whether and what data will be drawn from the web or the local data cache. In the first case, agents make use of selected back-end wrappers to extract the desired information. Secondly, the execute a query on the local database. In both cases, an agent has to format its output according to a predefined grammar (DTD), so that the front-end modules will be presented with results in a uniform way.

2.3.4 The wrappers

Automatically generated or hardcoded, these are the most vital part of CIA. They extract information from the e-sites and transform it to the C.I.A. internal data model. Agents and wrappers will be discussed further in the following chapter. For a more technical overview of wrappers, see appendix 5.2.

Due to the authors' previous experience with such tools as Lex and Yacc and the demand for maximum control on the generated wrapper, grammar-based toolkits were preferred over learning-based ones. After extensively testing many available solutions, we ended up using

[2]For more information on DOM and DOM-related technologies, please refer to [12, 13, 14, 15, 16, 17].

the mediator has the ability to choose between Jedi and Minerva based wrappers at runtime.

Of course, grammar-based toolkits require the user to have knowledge on programming and grammar rules. This is no disadvantage at this stage of development. However, should C.I.A. ever go commercial, a more user-friendly toolkit would be preferable.

The integration of wrappers with the rest of the system is shown in Fig. 2.2 and 2.1.

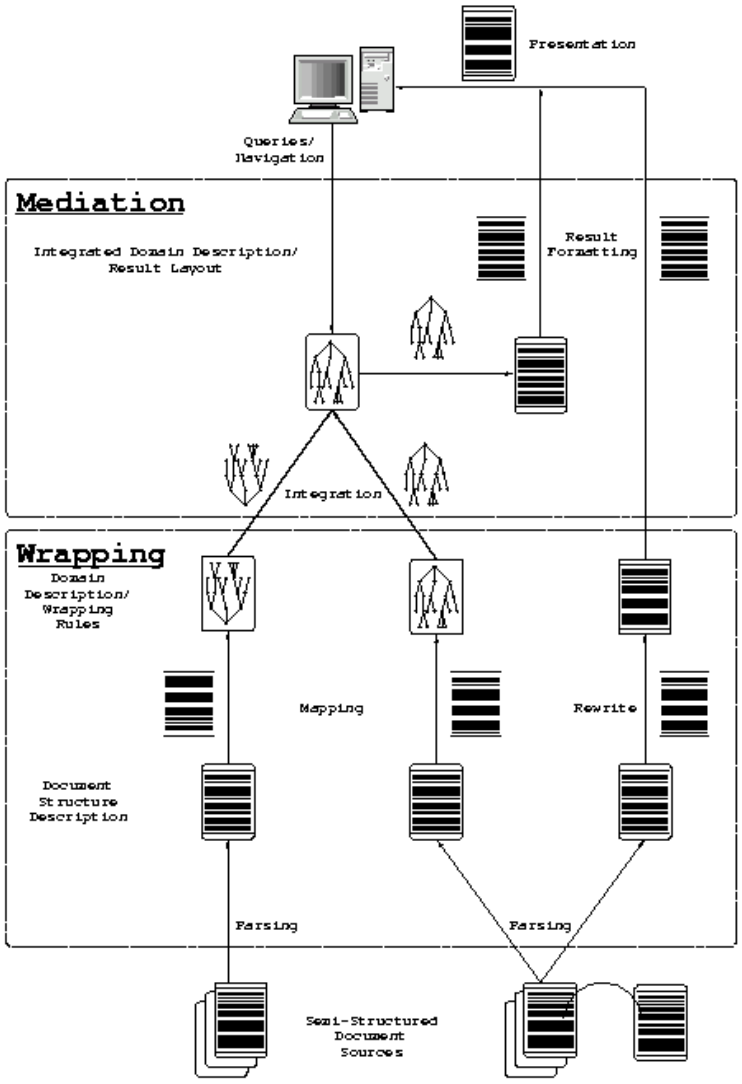


Figure 2.2: Integrating third-party Wrappers into a Mediator-Based System.

2.4 The e-sites

These are the original sources of information; the back-end e-sites that we query, in order to extract the desired information. They can be HTML or XML web servers, using any kind

of server side technologies, but only text based client side ones (for example, a Macromedia Flash-enabled web page can't be queried, since it is impossible to parse with our wrappers).

Chapter 3

HIT/CIA: The Mediator

The Mediator is the heart of the C.I.A. It is responsible for the retrieval of information from the back-end e-sites and the integration of data extracted by the latter ant its reformation according to its internal data model. For the shake of modularity, the Mediator is also implemented as a set of independent but cooperating components. These are:

1. the Front-End Interface,
2. the DOM-Tree Populator/Data Manager, implementing the local data repository, and
3. the Back-end Interface.

3.1 Front-End Interface (F.E.IN.)

To facilitate and modularize the communication between the mediator and the user interface modules, all data interchange is done using XML documents complying to a set of predefined DTDs (the naming scheme is <Application Name>-Query.dtd). Thus, the user interface modules need not know anything about the query execution methods or the internals of wrapping, while the mediator doesn't have to deal with HTML form elements and HTTP message parsing.

However, query execution is done using off-the-self technologies, such as XPath, XSL(T), XQuery or JDBC, depending on the implementation of the data repository. This calls for a translator from Query.dtd - compliant XML documents to the appropriate query language.

'feign'). FEIN consists of a set of translators, one for each application - query method pair. Since query DTDs are application - dependent, this part of the mediator is the only one with a need for domain-based knowledge.

In more details, FEIN functions in two levels:

1. Level 1: XML input manipulation, common to all translators within an application. At this level, the XML document given as input to the translator is parsed and all relevant information is extracted and stored. This information is then exposed to the second level via predefined Java methods.

2. Level 2: query generation, specific to the query language used. Using the information extracted during the first level of the translation, the modules functioning at this level generate the corresponding queries in the output query language. For the moment F.E.IN. supports XSL(T) and XPath, with XQuery support being under development.

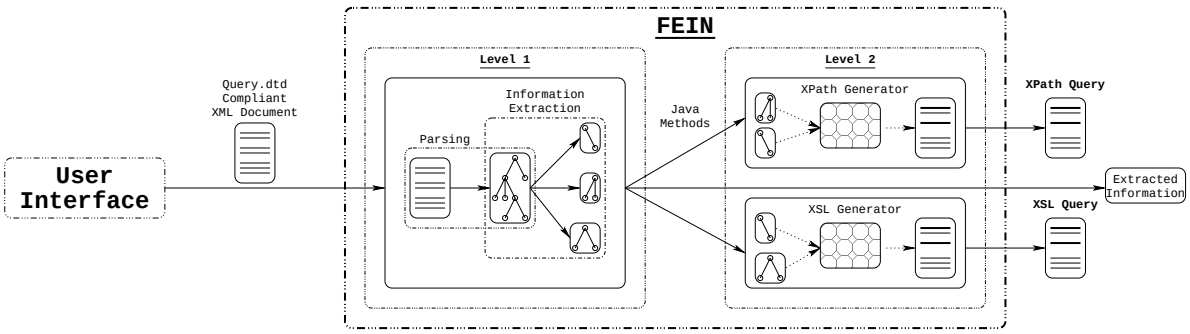


Figure 3.1: FEIN

As already stated, the use of the Query.dtd compliant XML documents as a means of communication between the user interface modules and the mediator, provides many advantages:

- it adds an extra level of abstraction between the user-interface modules and the mediator. After the query DTDs are defined, communication is done using DTD - compliant XML documents, independent of the actual query language used or the HTML form layout. This will be made clear at Ch. 4, where we use different query languages and HTML form layouts (actually one of the forms is static while another is generated on-the-fly)

for the different applications presented there, while keeping the same API between the user-interface and the mediator.

- it allows for transparent and easy addition of a caching subsystem between the user interface and the mediator, since the API used by the former to access the later is altered so that it redirects all queries to the cache.

- XML manipulation (generation, parsing etc.) is much easier than any other query language. The fully structured nature of XML makes it an ideal choice since manipulation is pretty simple while XML generation is straightforward once we have a result or query tree.

- addition of another query language doesn't affect the user interface at all, since what will actually be added is a Level 2 query generator for the corresponding language.

The only disadvantage of this method is that it requires parsing of the query DTD compliant XML document at all stages of the query execution (i.e. when the cache is added, the XML query is first parsed at the cache level, then (in the case of a cache miss or partial miss) regenerated and reparsed at the mediator level by FEIN). However, by using third-party XML parsers (namely Apache Project's Xerces parser, in deferred-node mode), the parsing-generation-reparsing overhead is reduced to a few milliseconds. As we have seen, the bottleneck of our system is the network transfer layer and not FEIN.

3.2 DOM-Tree Populator (DO.T.-POP.)

The output of FEIN is a query, executed on the data repository to extract the desired output. The mediator's repository is implemented as a forest of DOM trees, one for each back-end e-site. The system always does a cold start-up (i.e. the repository is empty when the mediator's server comes up). The population of these DOM trees, is done by the second part of the system: the DOM-Tree Populator (DOT-POP).

DOT-POP is domain - independent, so there is a unique DOT-POP for all implemented applications. The functionality of this part can be analyzed in the following stages:

1. Top-level selection: during this stage, DOT-POP selects the back-end sites that are involved in the executed query. This is done using the XPath output of FEIN to select all DOM tree roots that satisfy the top-level constraints defined by the user. At this stage of execution all DOM trees are almost empty; they contain only information included in the DomDB XML file (the exact functionality of this file will be further discussed in 3.2.1).

2. URL generation: at this stage, using the query constraints and a set of predefined rules, DOT-POP generates the URLs of the back-end HTML pages to be wrapped. The functionality of this stage is equivalent to the filling and submission of HTML forms in the back-end sites. We make heavy use of the capabilities provided by the java.reflect package to guarantee that this part of DOT-POP is also domain-independent. All information needed throughout this stage is extracted at runtime either from the user query or from corresponding metadata kept in external XML files.

3. BIN invocation: after defining the wrapped URLs, control is transferred to the Back-end Interface (the functionality of BIN will be further discussed in 3.3). What we need to know for the moment is that BIN returns a set of XML documents, complying to an application - specific DTD, representing the data extracted from back-end sites.

4. DOM tree population and normalization: at the final stage of DOT-POP, all data extracted at the previous stage is added to the DOM tree. The system offers the ability to normalize the populated DOM tree, using XSL transformations. Information concerning the normalization as well as the corresponding XSLT document, are defined at runtime.

Please note that by normalization we mean any kind of XSL transformation on the resulting tree. XSL queries generated by FEIN are actually executed at stage 4 of DOT-POP.

As we can deduce from the above, when all four stages of DOT-POP are over, what we have is a DOM tree containing the result of the user-defined query. Correctness of the result

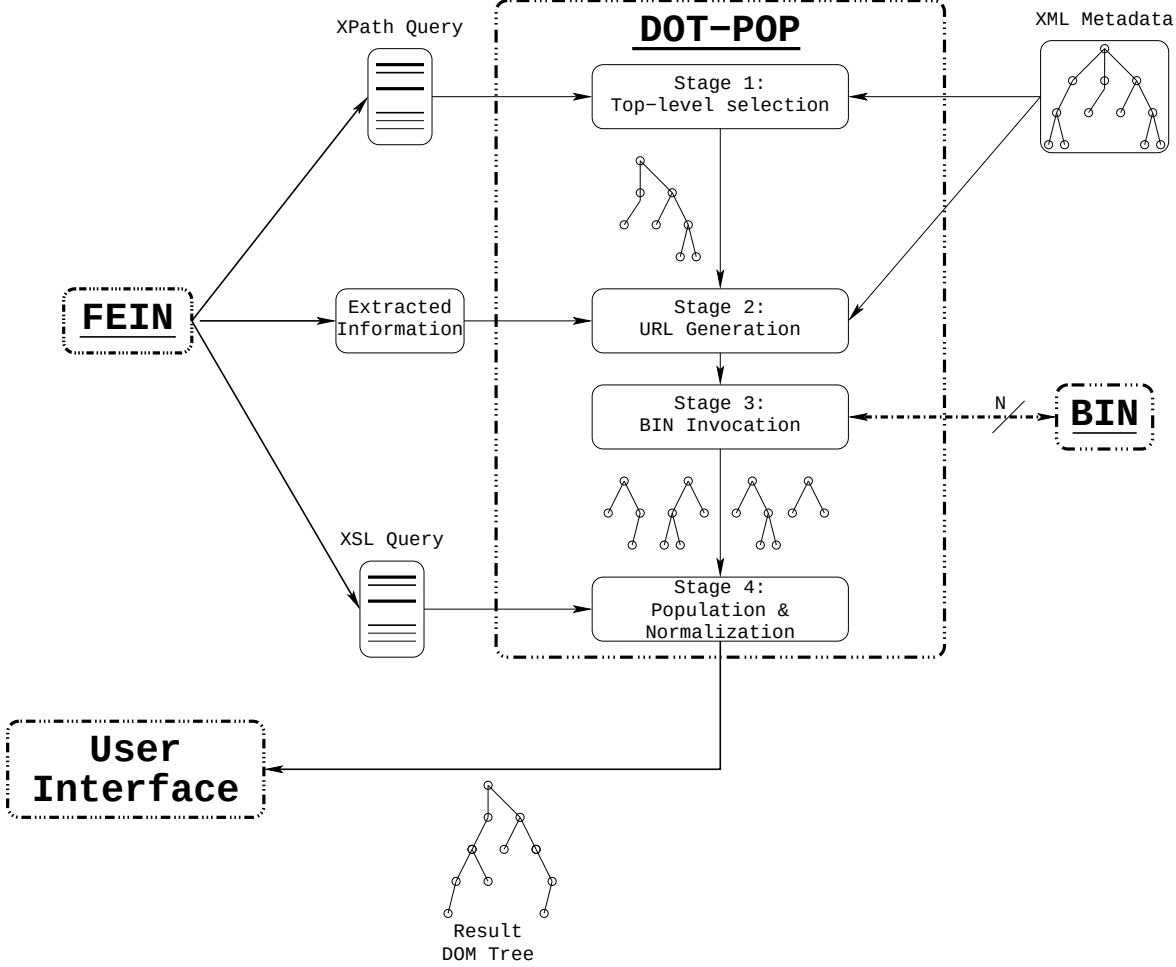


Figure 3.2: DOT-POP

set is guaranteed by the operations leading up to DOT-POP as well as by the four stages of the latter. Completeness however is a completely different story; since data is fetched over the web, we sometimes deal with network timeouts, web server misconfigurations or back-end e-site downtime. To account for these cases, DOT-POP characterizes DOM trees that are empty, as a result of data transfer errors, as incomplete. This prevents the caching and storing of these parts of the result set, so the data repository is always in a consistent state.

3.2.1 The DomDB XML File

To avoid any unnecessary back-end e-site access and reduce network overhead, we need a means to select only those sites whose result-set will contain answers to the user query. Since such knowledge is not available a-priori, what we do is to deselect those sites whose result-set will certainly NOT contain any useful information. To do this, we need some extra data about

kept in XML files (one for each C.I.A. application), namely the DomDB XML files.

The DomDB files are static XML files. They contain metadata (i.e. data about data) and any static (i.e. not changing) information concerning the back-end sites. They are loaded by the DOT-POP on-demand at runtime, every time a new query arrives. Data kept therein consists of all top-level information about the back-end sites (e.g. site name, URL, snail-mail address etc.), plus directives for the wrapping phase (e.g the class name of the wrapper, the class name of the URL generator etc.).

The use of DomDB files greatly improves the system's overall performance, by allowing selective querying of back-end sites with the (tiny) extra cost of loading and parsing the XML data they contain. An excerpt from the DomDB file used by one of C.I.A.'s applications - HyperHotel - is shown in fig. 3.3.

```
...

<Hotel Location="Lasithi" Category="A" Name="Kalimera Kriti"
Address="Elounda" SpecificLocation="Sissi"
URL="http://www.sterlinghotels.com/" HasConfRoom="TRUE" HasSwimPool="TRUE"
HasSea="TRUE" HasRestaurant="TRUE">
        <Wrapper Online="True" WGT="Minerva" ScriptName="TravelWeb"
        WrappedURL="URLGenerators.KalimeraKritiGenerator" />
</Hotel>

<Hotel Location="Chania" Category="C" Name="Alkion"
Address="Kato Stalos" SpecificLocation="Stalos"
URL="http://www.hotelalkion.com" HasConfRoom="FALSE" HasSwimPool="TRUE"
HasSea="TRUE" HasRestaurant="TRUE">
        <Wrapper Online="False" WGT="Jedi" ScriptName="hotelalkion.new.jedi"
        WrappedURL="file:Html/Remote/hotels/www.hotelalkion.com/Rooms.htm" />
</Hotel>

<Hotel Location="Chania" Category="B" Name="Halepa"
Address="164 El. Venizelou str." SpecificLocation="Halepa"
URL="http://www.halepa.com" HasConfRoom="TRUE" HasSwimPool="FALSE"
HasSea="TRUE" HasRestaurant="TRUE">
        <Wrapper Online="False" WGT="Minerva" ScriptName="Halepa"
        WrappedURL="http://www.halepa.com/HalepaHotel.html" />
</Hotel>

...
```

Figure 3.3: Sample DomDB XML File

3.3 Back-end Interface (B.IN.)

So far we have seen how we manipulate the user query, how we select the back-end sites to query and how we put together the resulting XML document. However, we haven't discussed

the way data is integrated with the mediator's internal data model. This is done by the third part of the mediator, the Back-end Interface (BIN).

BIN is responsible for fetching the data corresponding to the URLs defined at stage 2 of DOT-POP, extracting useful information and converting it to a predefined data model. BIN operations execute in two stages, to compensate for the network delays, while maintaining maximum parallelism for improved efficiency:

1. Stage 1: data fetching. During this stage BIN accesses the back-end sites, using the URLs generated by DOT-POP. It fetches and stores this information for use by the next stage. All retrieval operations are done in a parallel and thread-safe way, so that, given enough network bandwidth on the mediator's side, the overall network delay equals the maximum of the set of delays for each of the transfers.

2. Stage 2: wrapping. This is were wrappers are deployed. Wrappers are constructed using third-party wrapper generation toolkits (WGTs), usually available free of charge for educational and non-commercial purposes. Wrappers are responsible for parsing the data fetched during stage 1 of BIN's execution, using a set of predefined grammar and output rules. They usually generate forests of DOM trees, corresponding to the document fragments they parsed and wrapped. Since these operations usually involve numerous back-end sites, wrappers are deployed in parallel, using the multithreading mechanisms provided by the Java programming language. However, the WGTs used to implement C.I.A. suffer from certain inefficiencies as far as thread-safe execution is concerned. In order to accomplish maximum parallelism, the thread-unsafe parts of the wrapping process have been isolated and serialized using synchronized Java methods. Thus, the overall wrapping delay equals the sum of the maximum of the set of delays for the thread-safe parts of wrapping, plus the sum of delays from the thread-unsafe parts of wrapping.

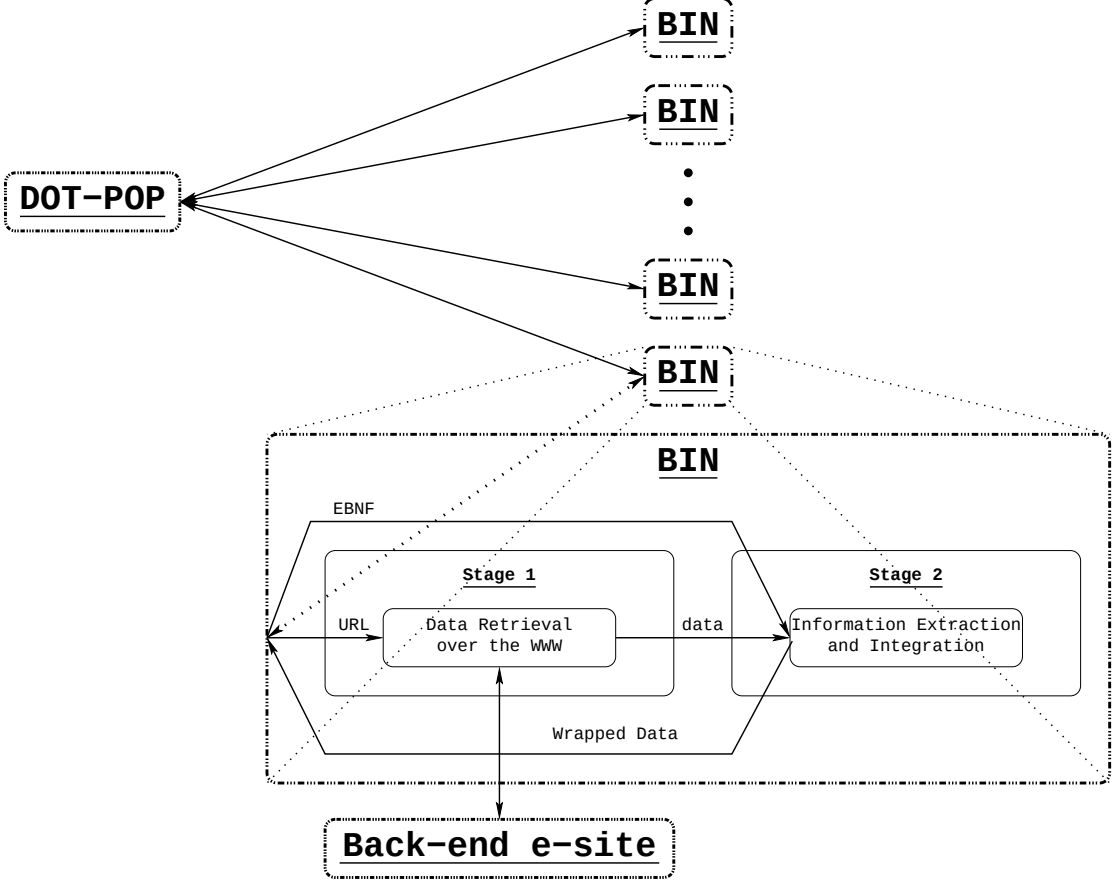


Figure 3.4: B.IN.

3.3.1 Wrapper Generation Toolkits

The wrapper generation toolkits used in the implementation of C.I.A. take as input a wrapping definition file. This file is a mixture of EBNF style grammar rules, Java or Java-like instructions and method calls, and output formatting definition instructions (see fig. 3.5 and fig. 3.6 [1]). For a more technical and detailed description of all examined wrapper generation toolkits, refer to Ch. 5.2.

The use of EBNF-supporting wrapper generation toolkits was the result of the following factors:

- the high grade of acquaintance with (E)BNF-based tools (e.g. Lex/Yacc etc.) minimizes learning overhead (the best-case scenario would be one in which the implementer would know a-priori exactly how to use the available tools). For example, of the two presented

[1] All Minerva figures and files were taken from the Minerva distribution. The Jedi file shown in fig. 3.6 was taken from the Jedi distribution. For licensing details, refer to the corresponding web sites (see App. 5.2 for more info on Minerva and Jedi).

```
  REMOVE_BLANKS
  TABULATOR("[","]|[","]\n")
}
PAGE AllConferences

//   This wrapper refers to the DBLP site at Trier
//   (http://www.informatik.uni-trier.de/~ley/db/).
//   It extracts data from the page containing
//   the list of all conferences in the site
//   (conf/index.a.html), mirrored in this distribution
//    in file araneusWTK\examples\html\conf-index.html

$AllConferences: *<hr> ( $ConfWithInitial )+ ;

$ConfWithInitial:
    <h3>                                // see X,Y,Z
      <a[ ]name="[a-z]">$Initial</a>(,<a[ ]name="[a-z]">[A-Z]</a>)*
    </h3>
    (
      <ul>
        (<li><a[ ]href="$ConfURL">$Acronym</a> $TmpConfName
         { $ConfName.replaceAll("[()*[]]",""); } $TP1
        )*
      </ul>
    )? ;

$Initial: [A-Z];

$TmpConfName:    - (\s)? $ConfName;
EXCEPTION()
{
    $ConfName.reset();
    $Acronym.cutAll();          // Append $Acronym ..
    $ConfName.paste();
    $TmpConfName.cutAll();      // ... and $TmpConfName documents...
    $ConfName.put(" ");         // ... separated by a space...
    $ConfName.paste();          // ... to form $ConfName
}

$TP1:
[
    $Initial    char(1),
    $Acronym    char(100),
    $ConfName   char(255),
    $ConfURL    char(255)

]
END
```

Figure 3.5: Minerva Sample Definition File

wrapper generation toolkits, we tend to use Minerva more than Jedi, since it fully supports the Java programming language rather than the syntactically Java-like language of the latter.

- grammar-based WGTs allow for maximum control on the input manipulation and output generation. Since this is a research project, we are more interested in using tools that allow us a great degree of control; user-friendliness is not required.

- the ability to mix grammar rules and programming language instructions gives maximum flexibility as to the integration and formatting capabilities of the generated wrapper.

- because of their deterministic nature, EBNF-based wrappers execute faster and require less memory than their learning-based counterparts, due to the simplicity of the data-structures required for their implementation.

The sample input and the corresponding output for the Minerva file shown in fig. 3.5, are depicted in figures 3.7 and 3.8.

```
// ********************************************************************
// rule BibEntry
// This rule matches the data of one bibliography record.
// It extracts relevant data by assigning data portions to
// variables which are used in a 'do' ... 'end' embedded code
// block which prints the data nicely formatted to stdout
// ********************************************************************
rule BibEntry : result is
        // Repeated author entries may appear that start with a '!'
        // These are collected in variable 'authors'.
        // The += assignment operator creates a sequence which
        // contains each match (.+)
        ('!' authors += .+ )+

        // an affiliation is optional and starts with '@'
        ('@' affil = .+ )?

        // The title starts with '*' and ends with '*'
        '*'  title = .+ '*'

        // skip irrelevant data
        .+

        // References start with '/' and are further seperated by '*'
        ('/' refs += .* )+

        // Rarely used comment entry
        ('=' comment = .+)?

        '>' // This char ends any comment or ref section

        do
                // Printing of formatted record data to stdout.
                // This code block is only executed if the rule matches.
                println(
                        "<hr><b>",title,"</b><p>",
                        authors.join(", "),


                        affil   ? text("(",affil,")") : "",
                        refs    ? text("<ul><li>",refs.join("<li></li>"),"</li></ul>") :
"",
                        comment ? text("<I>",comment,"</I>") : ""
                );
        end
end


// ********************************************************************
// rule BibList
// BibList matches a list of bibliography records.
// ********************************************************************
rule BibList : result is
        result = BibEntry()+
                // This production takes implicitly advantage of Jedi's
                // built-in failure tolerant parsing.
                // Looking at the 'BibEntry' rule, a record starts with a '!'
                // and ends with a '>'.
                // However, in the source a '!' does not follow immediately
                // after a '>'. These characters are skipped automatically
                // by the fallback of rule 'BibEntry'.
end

// ********************************************************************
// Script statements
// ********************************************************************

# Generation of HTML headers ...
println("<html><head><title>RESULT!</title></head><body>");
println("<hr><H2>Bibliography Rewrite Result:</H2>");

# Rules are objects with methods for parsing.
# We use BibList as an entry point to the grammar,
# parsing the content of the URL given.
BibList.parse(URL("http://www.darmstadt.gmd.de/~huck/jedi/bibliography"));

# Completing HTML ...
println("</body></html>");
```

Figure 3.6: Jedi Sample Definition File

3.4 Querying Models

As we already mentioned, user queries are executed by DOT-POP in two stages:

1. Stage 1: filling-out back-end sites' forms using user-supplied values.

2. Stage 2: running a user-defined query on the outcome of the wrapping process.

The first stage is done internally by DOT-POP. However, the second stage can be implemented in a variety of ways:

- the DOM tree approach.

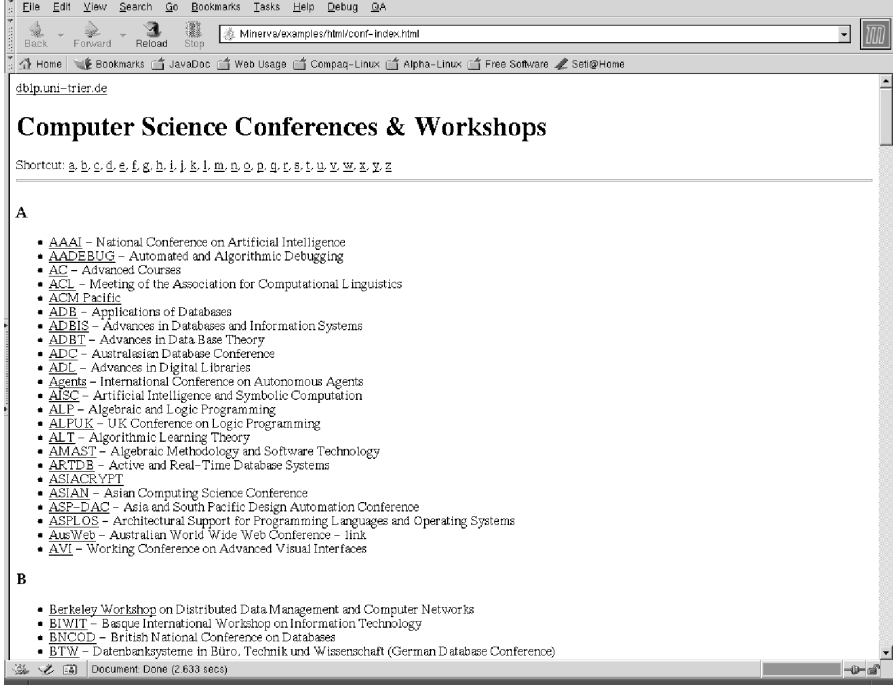


Figure 3.7: Minerva Sample Input

- the native XML database approach.
- the relational database approach.
- the fully-online approach.
- a hybrid approach combining two or more of the above methods.

3.4.1 DOM-Tree Approach

In this approach, all data from the e-sites being integrated, is prefetched and kept in a DOM tree. All queries thereafter are executed against this populated DOM tree, using some kind of XML-based query language, such as XSL(T).

This approach offers several advantages:

- It's extremely easy to program. Since wrappers return data in the form of XML documents or DOM trees, the population of the main DOM tree is straightforward and blazing-fast.
- It allows for easy querying. Java, through XML packages (such as Xalan) provides a variety of methods to execute XPath or XSL(T) queries against a given DOM tree; all

```
[A][AADEBUG][Automated and Algorithmic Debugging][aadebug/index.html]
[A][AC][Advanced Courses][ac/index.html]
[A][ACL][Meeting of the Association for Computational Linguistics][acl/index.html]
[A][][ACM Pacific Meeting of the Association for Computational Linguistics][pacific/pacif
ic75.html]
[A][ADB][Applications of Databases][adb/index.html]
[A][ADBIS][Advances in Databases and Information Systems][adbis/index.html]
[A][ADBT][Advances in Data Base Theory][adbt/index.html]
[A][ADC][Australasian Database Conference][adc/index.html]
[A][ADL][Advances in Digital Libraries][adl/index.html]
[A][Agents][International Conference on Autonomous Agents][agents/index.html]
[A][AISC][Artificial Intelligence and Symbolic Computation][aisc/index.html]
[A][ALP][Algebraic and Logic Programming][alp/index.html]
[A][ALPUK][UK Conference on Logic Programming][alpuk/index.html]
[A][ALT][Algorithmic Learning Theory][alt/index.html]
[A][AMAST][Algebraic Methodology and Software Technology][amast/index.html]
[A][ARTDB][Active and Real-Time Database Systems][artdb/index.html]
[A][][ASIACRYPT Active and Real-Time Database Systems][asiacrypt/index.html]
[A][ASIAN][Asian Computing Science Conference][asian/index.html]
[A][ASP-DAC][Asia and South Pacific Design Automation Conference][aspdac/index.html]
[A][ASPLOS][Architectural Support for Programming Languages and Operating Systems][asplos
/index.html]
[A][AusWeb][Australian World Wide Web Conference - link][http://www.scu.edu.au/ausweb95/]
[A][AVI][Working Conference on Advanced Visual Interfaces][avi/index.html]
[B][][Berkeley Workshop on
Distributed Data Management and Computer NetworksWorking Conference on Advanced Visual In
terfaces][berkeley/index.html]
[B][BIWIT][Basque International Workshop on Information Technology][biwit/index.html]
[B][BNCOD][British National Conference on Databases][bncod/index.html]
[B][BTW][Datenbanksysteme in B&uuml;ro, Technik und Wissenschaft
][btw/index.html]
[C][][C++ ConferenceDatenbanksysteme in B&uuml;ro, Technik und Wissenschaft
][c++/index.html]
[C][CAAP][Colloquium on Trees in Algebra and Programming][caap/index.html]
[C][CADE][Conference on Automated Deduction][cade/index.html]
[C][CAiSE][Conference on Advanced Information Systems Engineering][caise/index.html]
[C][CAV][Computer Aided Verification][cav/index.html]
[C][CC][Compiler Construction][cc/index.html]
[C][CCL][Constraints in Computational Logics][ccl/index.html]
[C][CDB][Constraint Databases and Applications][cdb/index.html]
...
```

Figure 3.8: Minerva Sample Output

a programmer has to do is put together the query and pass it as an argument to the relevant Java method.

- It allows for easy transformation from and to XML/HTML and other human-readable forms. This means that the presentation of the query results can be done using a simple XSL stylesheet and off-the-shelf tools.

- It is reusable. By implementing the functionality required to support a DOM-based database, we can use the exact same components for even more tasks (e.g. information extraction from traditional databases, transformation of data into many different forms etc.).

It does however suffer from some very important disadvantages:

- In order to support updates at runtime, the DOM Tree approach requires the design and implementation of components performing complicated manipulations of the DOM trees. The easiest solution is to rebuild the whole DOM tree every time an update is performed. However, the inefficiencies of this method are quite obvious. On the other

hand, implementing DOM tree updates makes all source code level maintenance tasks a very difficult endeavor.

- Memory requirements can become prohibitive. For the proof-of-concept applications implemented for this thesis, the amount of memory used was not very big. However, we expect memory requirements to grow almost linearly to the growth of the number of e-sites wrapped. A possible solution for this problem would be to use some form of persistent DOM trees, saving all information on some external storage device. This could circumvent the memory consumption matter, but the query execution time would increase.

- There is no query optimization mechanism other than that provided by the default XML manipulation Java packages. A traditional relational database can execute the same queries in fragments of the time required by the DOM tree approach to execute its XSL transformations.

3.4.2 Relational Database Approach

As the header implies, in this approach all data is prefetched and kept in a traditional relational database. All queries thereafter are executed using JDBC and SQL query language ([18]).

This approach offers the following advantages:

- Off-the-shelf solution. Using third-party products removes the burden of maintenance and support of these parts of the system. This allows us to concentrate on the development of the data integration side of C.I.A., using the low-level database access as a black box.

- Industrial-strength quality. Since this is a third-party product, it has probably been thoroughly tested and debugged by the company that developed it. The importance of this fact may not be obvious at this time. However, consider the possibility of C.I.A. going commercial; everyone would prefer a system based on a well-known, highly optimized relational database over an experimental and not-so-stable DOM-Tree based one.

- Easy querying. These products usually support a variety of query languages through different interfaces. Just to mention a few, we generally can run queries written in SQL, XPath, XML-QL([19]) and XML Query([20, 21]), via ODBC, JDBC, CORBA and RMI([22]).

- Fast queries. The fact that these products are separately developed, allows their developers to delve into such details as query scheduling and optimization, transaction control, storage optimization etc. The outcome of all these: a blazing-fast query execution engine.

- Powerful queries, if JDBC or ODBC is used. XPath, XML-QL and XSL are not as powerful query languages as SQL and XML Query([23]). Therefore, the use of a relational database for our metadata repository also allows us to easily execute powerful and complex queries in a straightforward manner.

- Easier updates. Of course a relational database management system (RDBMS) offering all of the above, could do nothing else than to also offer a powerful and highly optimized update mechanism. This solves the greatest inefficiency of the DOM-Tree approach in the best possible way.

It's main disadvantages include:

- Extra bindings and extra costs, due to the use of third-party commercial software.

- No source code is available since these are full-blown commercial products. This may prove really annoying in cases where a certain functionality is not implemented by the software in use (in which case we should either change the RDBMS used, or redesign the rest of the system, both of which being very expensive).

3.4.3 Fully On-Line Approach

All solutions presented so far, are based on some kind of prefetching and cache-storage technologies. If we completely remove this part of the system, what results is the Fully On-Line

approach, no data prefetching or caching is done on the server side. The only information the server has is the initial metadata, as made available by the DomDB XML files.

The advantages of this approach include:

- Extreme reusability. Since all tasks are done on-line, this approach needs only the DomDB XML files and the corresponding wrappers to function properly. If these two requirements are met, the Fully On-Line approach can act as a full-fledge, domain-independent mediator[2].
- It allows for selective queries and/or updates. Of course, the real update will be done against a server-side database, as those described earlier. However, the Full On-Line approach makes it possible to query only a portion of the back-end sites, thus allowing for selective execution of user or system-defined queries.
- It allows for better caching algorithms. Since no caching is implemented in the database, the task of caching is moved outside the Mediator, where more complicated and proprietary algorithms can be used.
- Low memory requirements. Since no caching is done on the Mediator server's side, the memory requirements of this approach are extremely low, compared to the requirements of the approaches presented so far.

However, this solution has the worst run-time efficiency of all previous approaches; each and every query takes extra overhead (network transfer, metadata parsing, data wrapping, wrapping output manipulation, etc.), making query execution a time-expensive process.

3.4.4 Hybrid Approach

C.I.A. uses none of these approaches as they were presented above. We rather invented a hybrid approach, combining the advantages of all of the mentioned approaches, while avoiding most of their shortcomings:

[2]As we shall see, the Fully On-Line part of the Mediator was used in an as-is basis in both of the sample applications of C.I.A.

- Fully On-Line queries, as an interface with the back-end sites,
- an internal DOM-Tree database, for the static or slowly changing parts of the wrapped information, and
- an external Relational database for the caching of wrapped data.

This approach offers all advantages of the individual approaches, with the following differentiations:

- It doesn't suffer from the run-time inefficiency of the Fully On-Line approach, since the back-end sites are contacted less frequently.
- The memory requirements are decreased to a minimum, since only static data is kept in memory.
- It provides ultra-fast execution of queries concerning cached or static data.
- The use of an external Relational database as a cache, moves the development and maintenance cost from the Mediator to the external Cache. Modularity is one of the developers' main goals.

3.5 Putting it all together.

Our Mediator is the assembly of all of the above. A schematic view is depicted in fig. 3.9.

Information integration, as presented above (Ch. 3.2), suffers from a serious inefficiency. Although all data is stored and manipulated as DOM trees, there is no specific method for all the parts and stages of mediation to share the same DOM tree in a thread-safe way. This means that data is converted from and to XML whenever the processing context changes (i.e. from DOT-POP to BIN, from BIN to DOT-POP etc.).

The main reason why this occurs is the lack of access to the source code of the wrapper generation toolkits as well as the lack of appropriate Java methods which would export processed data as DOM trees. However, for the time being, with C.I.A. still in the 'proof-of-concept'

stage of development, the delay presented to the system by this fact is negligible, since the number of back-end e-sites is not great. If efficiency ever becomes critical to the evaluation of C.I.A. as a whole, this is a section whose improvement would provide an important overall performance gain.

The Mediator and C.I.A. as a whole were designed with tree main factors in mind:

1. Platform independence: the final outcome should be easily portable among different software and hardware platforms. The use of Java and XML derived technologies guarantees maximum platform independence.

2. Architecture openness: modules should be easily alterable, while the set of wrapped back-end sites should be as dynamic as possible. This is achieved by using the DomDB files and in conjunction with the overall architecture design.

3. License freedom: all parts used should be either open/free source or freely available for educational reasons. The Mediator was developed in Java, using Sun's publicly available JDK, and non-commercial wrapper generation toolkits.

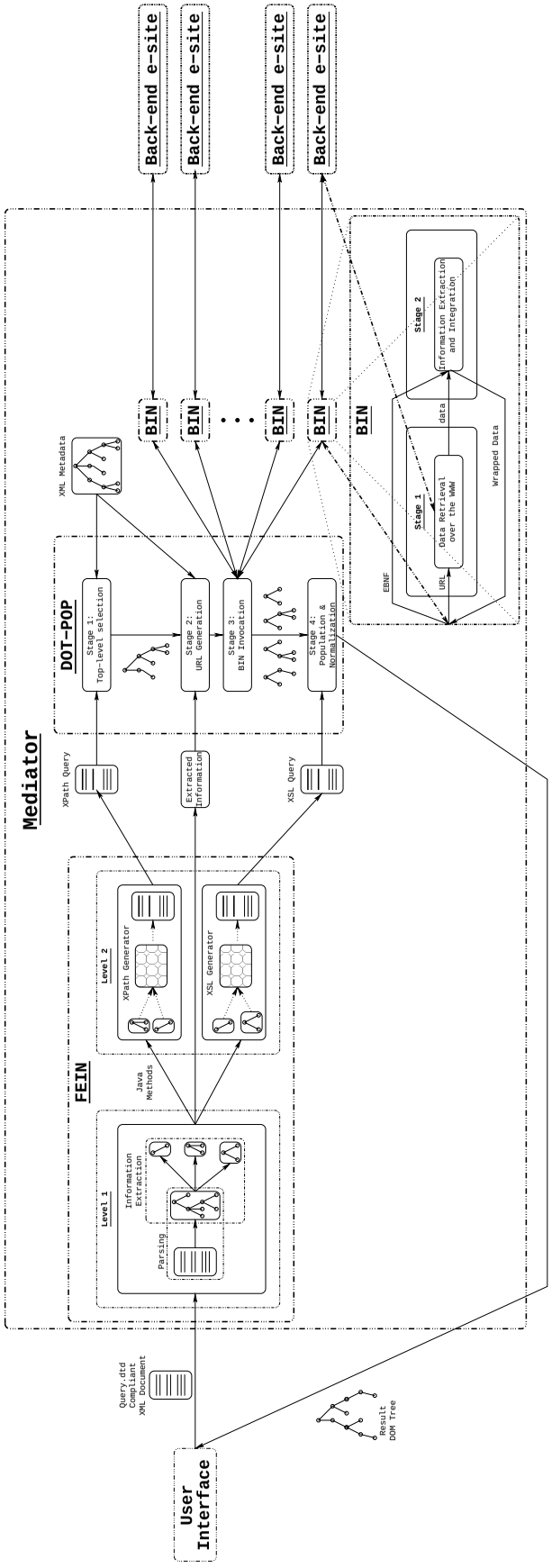


Figure 3.9: The Mediator

Chapter 4

Champion Applications.

4.1 Overview

We shall now present two proof-of-concept novel data integration applications, implemented using the C.I.A., as it was described in the previous chapters:

- HyperHotel: a data integration application, bringing together the vast amounts of information available online by real-world hotels. Users will have the ability to search for their residence-of-choice in multiple e-hotels, without having to visit the web sites of each and every one of them, and
- HyperTV: an application integrating data available on the world-wide-web, concerning television channels' programs. Users will eventually be able to search and select the TV programs they'll watch, by selectively querying TV stations' web sites, in a uniform, centralized and user-friendly manner.

4.2 HyperHotel

A classic example of data freely available through the world-wide-web in unstructured or semi-structured form, is data about accommodation facilities, such as hotels and rent-rooms. The common practice for such companies is to have a web site offering location-related information (such as address, transportation, proximity to known sites etc.), room rates, availability checking, online booking etc. Since tourism is a quickly growing sector of modern economies,

the demand for a centralized way of uniformly querying all such facilities is becoming greater. HyperHotel's goal is to answer to this demand in a semi-automated way, surpassing currently available paradigms utilizing data-entry and/or proprietary communication protocols between the integrator and the back-end sites.

Being an application of C.I.A., HyperHotel has the following advantages:

- It uses standard world-wide-web technologies and protocols (i.e. HTTP/1.1, SSL etc.) to communicate with the back-end sites and with the clients. This means that a hotel needs only have a web site to be a candidate for integration, while all a user needs is a web browser and access to the Internet.
- It is written totally in Java, hence guaranteeing maximum platform independence on the server side.
- Since all data is drawn from publicly available sources (i.e. the hotels' web sites), it is free of possible copyright and licensing issues.

To achieve better performance, the HyperHotel Mediator prefetches, integrates and stores all "static" data (i.e. data available through static HTML pages) at start-up, while still offering the capability of online dynamic querying through HTML form interfaces.

HyperHotel makes it possible for a user to query multiple hotel websites, based on such criteria as:

1. locality (currently more than 40 hotels from more than 5 areas in Greece are dynamically included in queries).
2. hotel category (e.g. A/B/C class etc.)
3. amenities available (e.g. swimming pool, conference room, restaurant etc.)
4. closeness to the sea.
5. proximity to the nearest city.
6. room size (i.e. number of beds).

7. actual dates of accommodation.

8. final room rates (including support for ranging rates, e.g. spanning multiple seasons).

A demonstration of HyperHotel is available online at http://hit.softnet.tuc.gr/Applications/HyperHotel.

4.3 HyperTV

HyperTV is another proof-of-concept application based on C.I.A.. It deals with the integration and uniform querying of television channel programmes, available online through the corresponding television channel website.

Currently almost all television channels have a website, mainly providing information about the channel's programme. Moving into the era of digital television, users will have access to hundreds of channels, through the corresponding digital platforms. The diversity and large amount of programmes will then make printed TV guides and individual channel websites obsolete or unusable. HyperTV covers this gap, by providing a "centralized" way of selectively querying multiple programmes, transparently to the end-user.

HyperTV allows for queries based on such criteria as:

1. channel name (currently more than 10 channels are integrated).

2. channel type (i.e. satellite, cable, digital, subscriber-based etc.)

3. program start/end hour.

4. program name.

5. program type and/or description.

6. predefined queries, concerning high-volume high-frequency user queries (e.g. "What movies are there tonight?" or "What football matches are on tomorrow?").

To achieve maximum query execution performance and minimum response time, HyperTV prefetches, integrates and stores all relevant data, for a predefined amount of time (usually

seven days)[18]. This process is done during low activity time periods (e.g. at night), so that users will seldom see any degradation in performance. The stored data is then deleted and / or updated in a circular way, on a per-day basis: at the end of each programme day (when the programme for a day becomes obsolete), the system prefetches the programme for the seventh day (i.e. the programme of the first day of the next week, starting from the current day). This technique guarantees completeness and correctness for the prefetched and stored data.

A demonstration of HyperTV is available online at http://hit.softnet.tuc.gr/Applications/HyperTV.

Chapter 5

Related Work

We proceed with the description of related work and its comparison to C.I.A.. We classify relevant projects in two major categories, as to their functionality: full-scale integration and wrapper generation.

5.1 Full-Scale Integration

Data integration has been a hot-spot of information technology for many years. Many attempts have been made to create a full-blown mediator system. Due to the size of relevant work, we choose only to refer to the most well-known among them[1]: TSIMMIS([48, 49]), DISCO([50, 51]), Enosys Markets([52]) and ShopBot([53]).

5.1.1 TSIMMIS

TSIMMIS stands for The Stanford-IBM Manager for Multiple Information Sources. As its name implies, TSIMMIS was developed by the Stanford University in cooperation with IBM.

TSIMMIS is a classic representative of a full-scale mediator system, consisting of the following components:

[1]For more information on the subject, please refer to [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47].

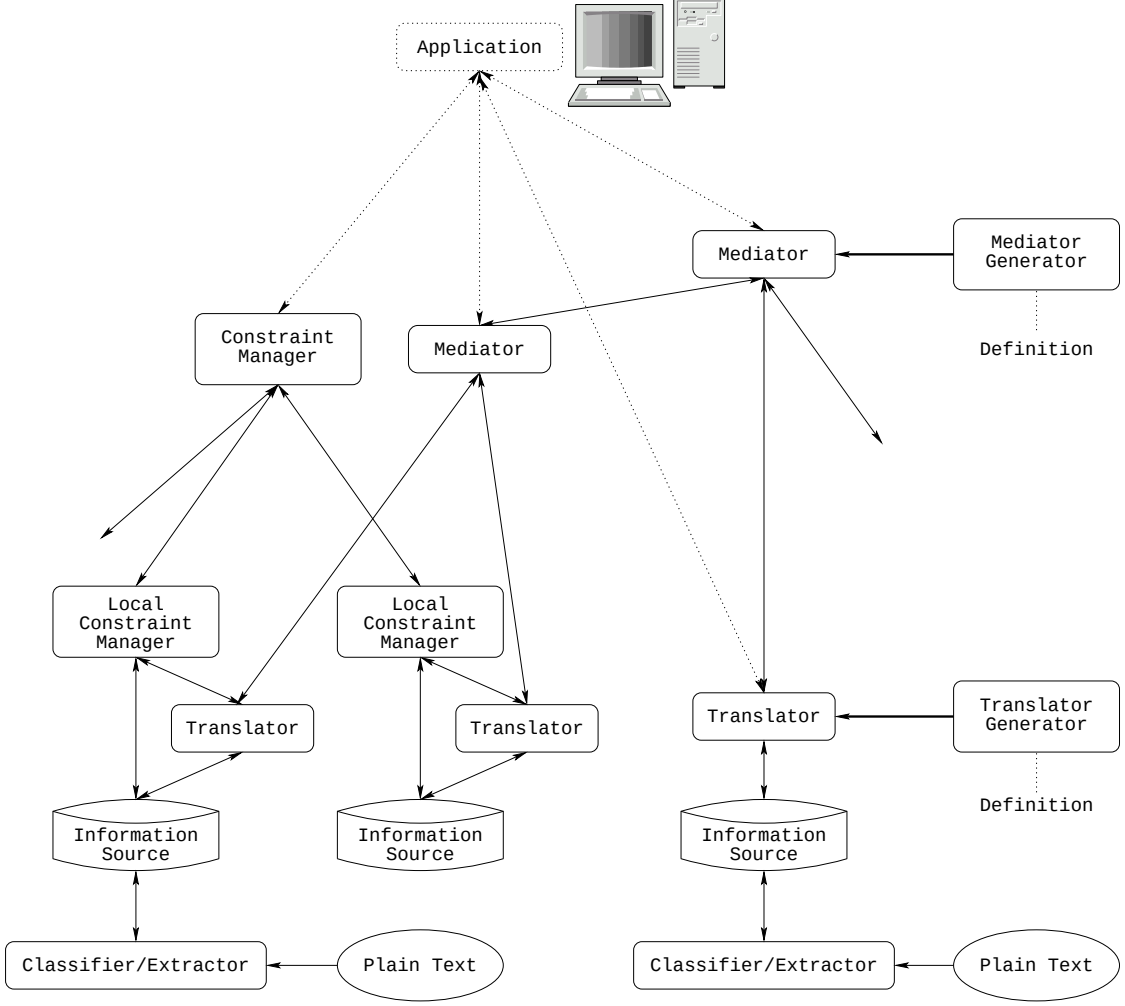


Figure 5.1: The TSIMMIS System.

- The *Translator Generator* and the generated *Translators.* Back-end sites are accessed through wrappers (referred to as "*Translators*" in the TSIMMIS literature), semi-automatically generated by the *Translator Generator*, a wrapper generation toolkit. *Translators* deal with the tasks of converting queries to a form executable on the back-end sites and then converting the extracted data to the TSIMMIS object model.

- The *Mediator Generator* and the generated *Mediators. Mediators* are semi-automatically generated super-wrappers, dealing with the task of selectively including sets of *Translators* in query execution time, based on semantic criteria (e.g. query semantic category and *Translators* available for relative back-end sites).

- The *Constraint Managers.* These components deal with integrity constraints imposed on the integrated data by front-end applications.

- The *Classifiers/Extractors*. These components deal with the extraction of useful attributes from unstructured data sources (e.g. plain-text files), so that such information could be used by a *Translator* in subsequent queries.

- The data model. TSIMMIS uses a self-describing (tagged) object model, called the *Object Exchange Model*, or *OEM*. *OEM* allows simple object nesting, thus being a very simple object model. Queries against *OEM* repositories are issued in *OEM-QL*, and SQL-like language, specific to the TSIMMIS project.

The design of TSIMMIS is very close to that of the C.I.A.; both platforms use wrappers, semi-automatically generated using wrapper generation toolkits, to access back-end e-sites. In the C.I.A., constraint management is done during stages 2 and 4 of the DOT-POP execution, while there is no need for *Mediators*, since multiple C.I.A. instances are deployed for different semantic categories.

Moreover, both platforms use self-describing object models. However, TSIMMIS's *OEM* calls for a model-specific proprietary query language, while XML is an industry standard, with querying capabilities controlled by the W3C XML group. Another major difference between TSIMMIS and the C.I.A. is the use of *Classifiers/Extractors*, since C.I.A. currently doesn't support integration of unstructured data. This is due to the fact that currently publicly available wrapper generation toolkits don't support efficient wrapping of unstructured data.

5.1.2 Disco

DISCO stands for Distributed Information Search COmponent and was developed by Inria Rocqencourt and the University of Maryland.

DISCO is another representative of a full-scale mediator system. It features:

- The *Catalog*: a collection of data concerning the *Mediators*. The *Application* uses data available in the *Catalog* to select which *Mediators* to include in every query execution.

- The *Mediators*. DISCO's Mediators deal with the tasks of selecting which wrappers to use for every query and the conversion of wrapped data to the system's data model.

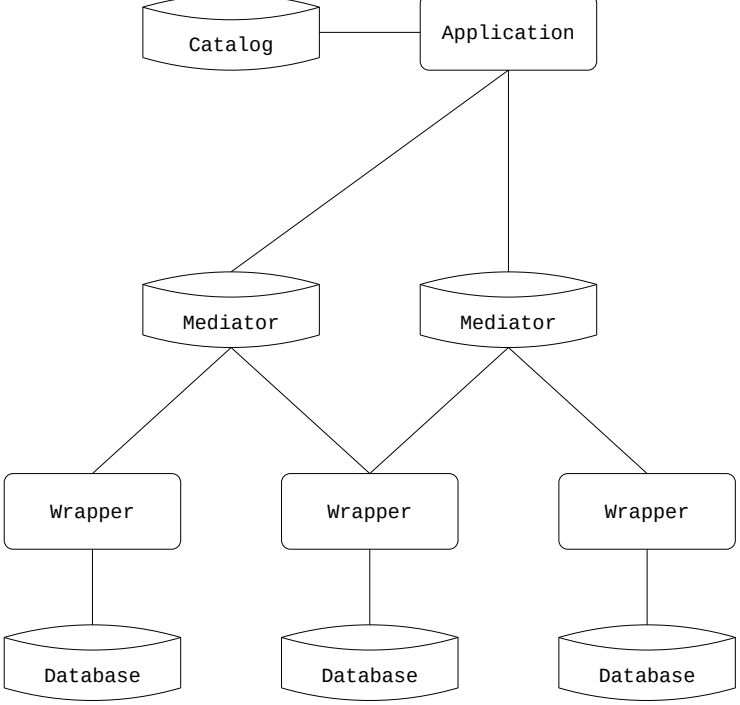


Figure 5.2: The DISCO System.

- The *Wrappers*. These components deal with the conversion of queries, from a subset of the general query language used by the *Mediator*, to the particular language used by the corresponding data source, and with the conversion of data thus acquired to the format expected by the corresponding *Mediator*.

- The data model. DISCO's data model is based on the ODMG-93 standard[54], consisting of an object data model (*ODM*), an object definition language (*ODL*), a query language (*OQL*) and a language binding.

DISCO's approach to mediation is very similar to that of TSIMMIS, to the extent that both systems use wrappers, controlled by mediators, controlled by a "super-mediator" (i.e. the *Catalog*). However, DISCO mediators are not semi-automatically generated, as is the case with TSIMMIS's mediators. As already mentioned, the C.I.A. makes no use of "super-mediators", since multiple C.I.A. instances are deployed for equal semantic categories. Moreover, DISCO, by extending the ODMG standards, uses a less proprietary data model than TSIMMIS's *OEM*, but it's still far more complicated than the flat XML data repositories of C.I.A.

Another major difference between the DISCO approach and that of the TSIMMIS and the C.I.A. systems, is the intended back-end data sources; DISCO is designed for the integration of DBMS-based information systems. The issues that arise in the integration of such systems are very similar to those arising in the integration of data sources over the web (as those dealt with by TSIMMIS and the C.I.A.), with the exception of the semistructured nature of web-based sources; data retrieved by directly querying a DBMS-based data source is always structured, while data transfer is done through proprietary APIs, specific to the DBMS. As such, we consider the DISCO system to be more of a data wrapping service than a full-scale mediator.

5.1.3 ENOSYS Markets

ENOSYS is the commercial offspring of the research made for the TSIMMIS project. As such, the ENOSYS system also deploys wrappers (*XMLizers*) for the extraction of information from the back-end data sources, and mediators (*XMediators*) for the integration of extracted data. It also features an XML cache database, much like the C.I.A. platform.

The main building blocks of the ENOSYS platform comprise of:

- The *XMLizers*: ENOSYS Markets' *XMLizers* (wrappers) deal with the extraction of data from back-end data sources and the transformation of extracted data to the mediator's internal data model. *XMLizers* exist for various types of data sources, such as RDBMSs, XML files etc. *XMLizers* are semi-automatically generated, using user-friendly visual tools and a declarative source definition language.

- The *XMediator*: all information extracted by the *XMLizers* is then passed to the *XMediator*, ENOSYS Markets' mediator. The XMediator then exports this information in the form of "views" on the integrated data, thus resembling a database. As a matter of fact, the server-side components of the ENOSYS system is called Virtual Integrated XML Database (*VIX Database*) in the ENOSYS literature, thus the corresponding views on integrated data are called *VIX Views*.

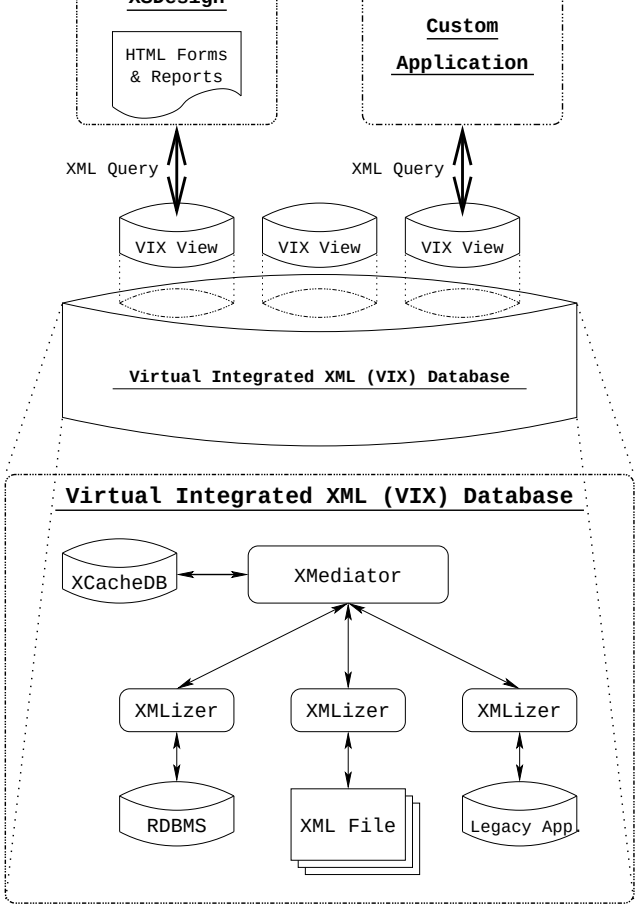


Figure 5.3: The ENOSYS System.

- The *XCacheDB*: when the wrapped data sources are either static or slow, their XML views are cached in the XML Cache DataBase, the *XCacheDB*. Of course, caching is done transparently, considering the rest of the system's components.

There is great resemblance between the ENOSYS Markets' system and the C.I.A., apart from both being classic representatives of full-blown mediator systems; both systems utilize caching techniques to compensate for high-reaction-time or static data sources. However, the C.I.A. takes caching one step further, by caching each and every query and it's corresponding response, using state-of-the-art caching techniques and partial hit/miss recognition and by tackling the integration problem in a distributed manner (through cache replication and multiple mediator instances).

On the other hand, the ENOSYS system, being a commercial product, is a much more complete data integration system than any of the systems presented here. In a nutshell, it also features visual tools to simplify wrapper generation and data acquisition and maintenance,

of the ENOSYS system to create web/HTML-based forms for the access of integrated data.

5.1.4 ShopBot

The ShopBot was developed by R.B. Doorenbos, O. Etzioni and D.S. Weld at the University of Washington. It was a World-Wide-Web shopping agent that enabled users to shop online for CD's and computer software, but was retired in 1998. However, it contributed to the development of more advanced shopping agents, such as the *Jango* (http://www.jango.excite.com).

The ShopBot utilized advanced artificial intelligence techniques to understand information published at back-end sites; information extraction was more-or-less automatic, based on limited domain-specific knowledge and certain assumptions as to the structure and content of the integrated data. It did however suffer from some major problems:

- Data analysis was not detailed enough and could lead to wrong output (e.g. upgrades of a program, being less expensive than the program itself, appeared higher in ShopBot's sorted list of available products).
- The rules on which ShopBot's decisions were based, were too strict, leading to incompleteness of the output (e.g. when the formatting of a web page wasn't within the limits recognized by the ShopBot, parts of the integrated data could be mistaken to belong to the useless content of the page).
- The ShopBot could only integrate sites with a searchable index. This not being the case for many all of the available e-shops, the ShopBot could only integrate a fraction of the set of e-shops.
- ShopBot's performance was linear in the number of integrated data sources, thus not scaling well for large numbers of back-end e-sites.
- Wrapper generation was heavily based on the assumption that all data sources export information in HTML form. This means that the ShopBot would never be able to integrate a data source embedding information in Java Applets, images etc.

5.2 Wrapper Generation Toolkits

We continue with a presentation of the major wrapper-generation toolkits available today, which where also candidates for adoption by the C.I.A.. All of them have the following characteristics:

- They are written in Java, therefore guaranteeing platform-independence and maximum integration capabilities with the rest of the system.
- They are available free of charge for educational reasons. Although C.I.A. might evolve to a commercial platform, in this stage of development no commercial products should be used.

We separate toolkits in two major categories, according to the way they interact with the user in order to generate the wrapper:

1. Grammar-based toolkits.
2. Learning-based toolkits.

5.2.1 Grammar-Based

These toolkits take as input a description of the grammar of the wrapped source and a definition of the output format. They then generate a wrapper that matches the given grammar rules to the e-site web-pages and returns the parsed data, according to the output format definition. Grammar and output definitions are made in toolkit-specific formats. The ease-of-use of this format plays a very important role in choosing one toolkit over another.

With these toolkits, wrappers are harder to code, since they require grammar rules induction by the user, but allow maximum control over the generated wrapper.

The most prominent representatives of this category are the Lex-Yacc parser generation suite, the GMD-IPSI's JEDI([55]), INRIA's YAT([56, 57, 58]) and the Universita di Roma's Minerva([59, 60, 61, 62, 63, 64]) wrapper generation toolkits.

5.2.1.1 Lex-Yacc

The Lex-Yacc parser/compiler generation suite has been around for quite a long time. Most compilers available today are developed and maintained using these very two tools or their various ports (e.g Flex/Bison for GNU/Linux etc.). As expected, they have also been ported to Java by various developers (e.g. Coco/Java, CUP, the JavaCC etc.).

Due to the low-level nature of these tools, they are the most powerful of the presented toolkits as to the features of the generated parsers. However, programming in Lex-Yacc can be very time consuming, while the advanced possibilities of this toolkit would surely be never used in the context of web-page data-source wrapper generation.

5.2.1.2 JEDI

JEDI stands for Java-based Extraction and Dissemination of Information. It was developed at the Integrated Publication and Information Systems Institute (IPSI) of the German National Research Center for Information Technology (GMD). Quoting from the JEDI Handout:

> JEDI adopts a lightweight approach to wrapping and mediation, requiring only basic web-browser technology. It has been entirely implemented in Java.
>
> JEDI's wrapper consists of a powerful and fault tolerant parser. Using attributed, nested rules that describe the source structure of documents, the parser segments them to any desired level, and collates the parsed data into a network of objects. Unlike parsers for formal languages, JEDI's parser can cope with incomplete and ambiguous source specifications. This is accomplished by a novel parsing technique that chooses always the most specific rule among several applicable rules. When finding no applicable rule for some document portion, it skips as little as necessary to continue with an applicable rule.
>
> Wrappers and mediators have been carefully designed to tolerate structural deviations and incomplete specifications without trading expressive power. The immediate advantage of this is that users can concentrate on what they want to reuse and

in addition, JEDI's fault-tolerance leads itself to applying machine-learning techniques that explore information spaces to generate recognition rules and mapping specifications semi-automatically.

JEDI's architecture is shown in Fig. 5.4. For a more detailed description of JEDI, refer to [55].

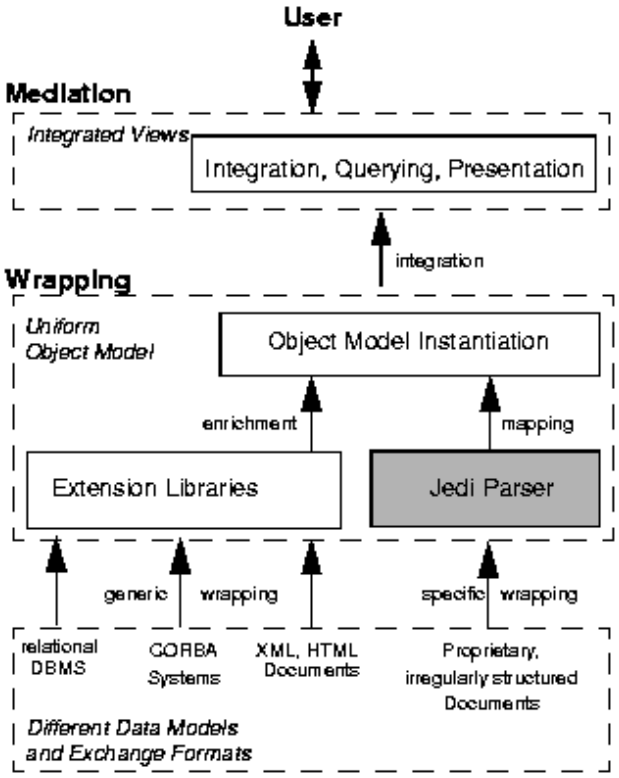


Figure 5.4: JEDI Architecture

JEDI and related demos and documentation, can be found at

- http://www.darmstadt.gmd.de/oasys, and
- http://www.darmstadt.gmd.de/~huck.

5.2.1.3 YAT

YAT was developed by INRIA, as part of the OPAL project. YAT stands for Yet Another Tree-based system. Its architecture is shown in Fig. 5.5.

Quoting from [57]:

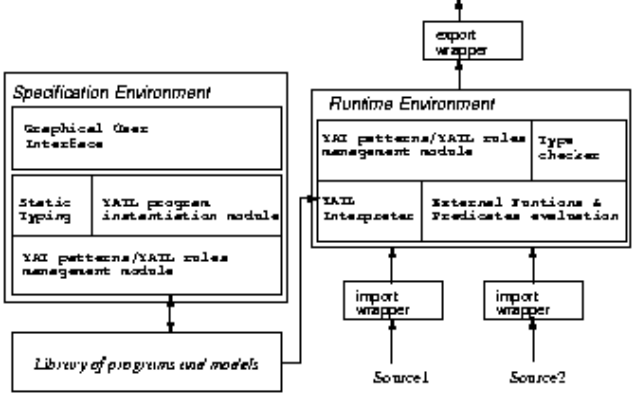


Figure 5.5: YAT architecture

It relies on a middleware model, a declarative language, a customization technique and a graphical interface. The model is based on named trees with ordered and labeled nodes. Like semistructured data models, it is simple enough to facilitate the representation of any data. Its main originality is that it allows to reason at various levels of representation. The YAT conversion language (called YATL) is declarative, rule-based and features enhanced pattern matching facilities and powerful restructuring primitives. It allows to preserve or reconstruct the order of collections. The customization mechanism relies on program instantiations: an existing program may be instantiated into a more specific one, and then easily modified.

A sample translation scenario is described in Fig. 5.6. For the time being, the YAT system isn't yet available to the public. Therefore, the authors have no experience on its functionality and coding facilities.

5.2.1.4 Minerva

Minerva was developed at the University di Roma Tre, in cooperation with Universita della Basilicata, as part of the Araneus project. It builds on the idea of dealing with exceptions caused by the parsing of a document. It allows for both an EBNF grammar approach and a procedural manipulation of document data.

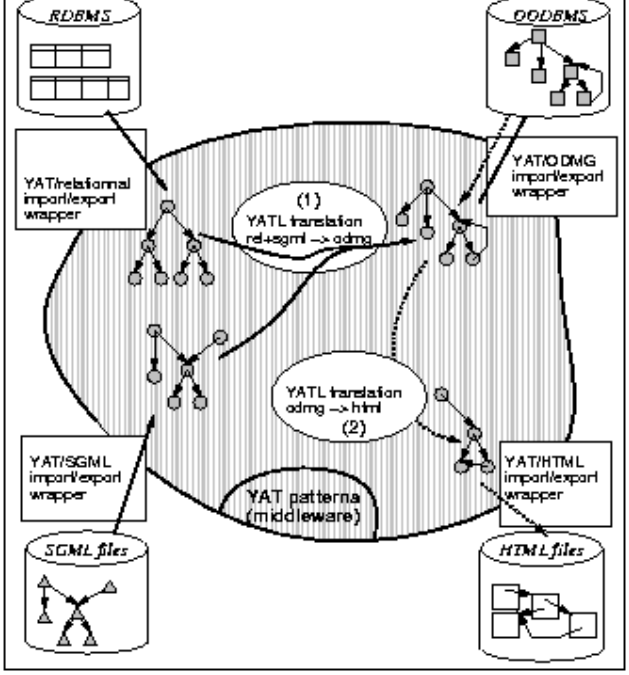


Figure 5.6: YAT translation scenario

Being an EBNF-based wrapper generation toolkit, Minerva allows for maximum flexibility and powerful wrapper generation. However, it's greatest advantage over the rest of the EBNF-type wrapper generation toolkits, is it's support for inline Java code. Thus, it combines the ease of use of the Jedi toolkit, with the power and robustness of the Lex-Yacc suite. Moreover, it allows for execution-time manipulation of malformed input, through the support for parsing exceptions([65]).

Sample input and output files of the Minerva toolkit were presented earlier.

5.2.2 Learning-Based

The toolkits that belong to this category use machine-learning algorithms and AI concepts in order to extract the grammar rules used to generate a wrapper. They usually interact with the user through a GUI. In order to assure that the rules extracted are correct, they prompt the user for suggestions and corrections, through which they "learn" what parts of the wrapped site the user is interested in. The output format is also defined by the user, through the same GUI.

These toolkits require little or no coding. However, peculiarities in wrapped web pages make learning very difficult, so they don't always succeed in inducing the correct grammar rules, even after several suggestions and corrections from the user.

NoDose([66, 67]) and XWrap([68]) are two of the most well-known toolkits of this category.

5.2.2.1 NoDose

NoDose stands for Northwestern Document Structure Extractor. It was developed at the Computer Science Dept. of the Northwestern University. Quoting from [67] and [66]:

> NoDose allows non-programmers to build components that can convert data from the source format to XML or another generic format. Further the generated code performs a set of statistical checks at runtime that attempt to find conversion errors before they are propagated back to users ([67]).
>
> Using a GUI, the user hierarchically decomposes the file, outlining its interesting regions and then describing their semantics. This task is expedited by a mining component that attempts to infer the grammar of the file from the information the user has input so far. Once the format of a document has been determined, its data can be extracted into a number of useful forms ([66]).

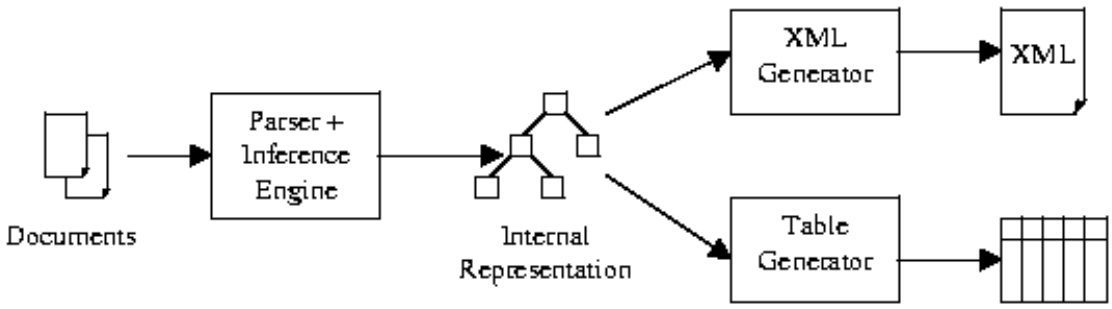


Figure 5.7: The steps of using NoDose

According to the authors' experience from using this toolkit, wrapper induction with NoDose is a trivial task when the wrapped source is relatively semi-structured. However, when the source was highly unstructured, NoDose required a larger time frame to teach, since it learns by example.

5.2.2.2 XWrap

XWrap is another wrapper generation toolkit that builds on the idea of interacting with the user through a GUI and generating wrappers through learning. Its architectural outline is shown in Fig. 5.8. The main idea behind XWrap is separating common wrapping tasks from source-specific ones; the wrapper generation process is done in two steps:

1. The user defines the regions of interest in the wrapped source, using the GUI. Behind the scene, the user's selections are translated into declarative information extraction rules.

2. The XWrap system then combines these rules with the XWrap component library and constructs a procedural wrapper program (in Java).

The system also provides the ability to come back and tune the generated wrapper at run time.

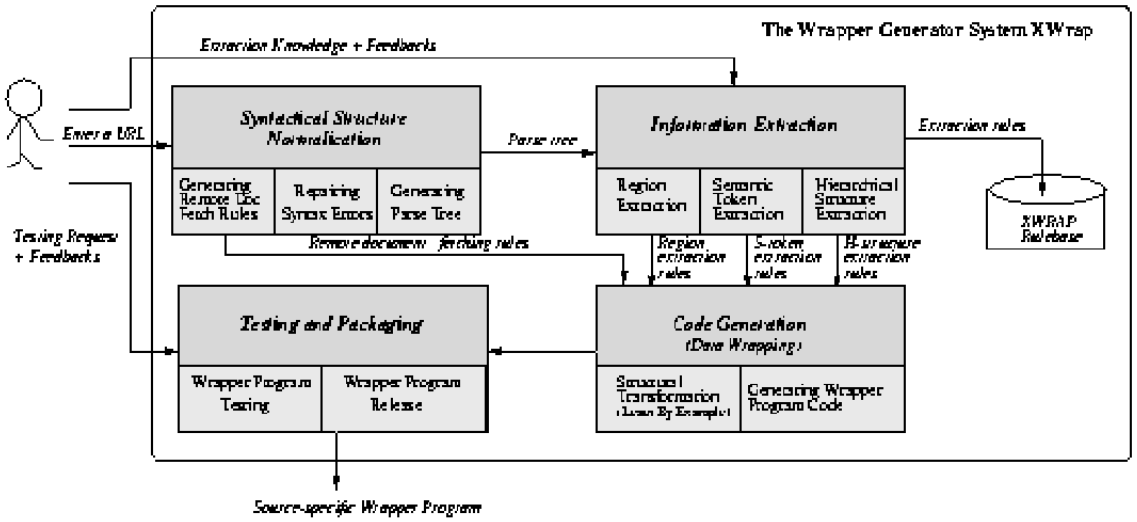


Figure 5.8: XWrap architecture

For the time being, XWrap is available as an online resource (i.e. XWrap is not available for download. Wrapper generation is made through an HTML interface on the XWrap web host).

Chapter 6

Conclusions and Future Work

The heterogeneity and unstructured form of data available online through the world wide web has recently evolved to a hot subject of research by academics and relevant industry. The most wide-spread solution to the problems posed by these attributes and by the vast amounts of available information, is the deployment of data integration schemes, mainly in the form of mediators and relevant wrappers.

In this thesis we have described the design and implementation of an independent mediator model, using third-party off-the-self wrappers and/or wrapper generation toolkits: the Content Integration Architecture (C.I.A.). Emphasis has been given on speed, platform independence and ease of deployment. The purpose of this work is to automate the task of selectively querying multiple data sources on the web and presenting the results in a uniform way. We have also demonstrated the functionality of this system with two applications: HyperHotel and HyperTV; two dynamic mediator-based information integration and dissemination systems for e-hotels and television program listings, respectively.

Throughout the design and implementation stages of this work, we have reached the following conclusions:

- Despite the emergence of XML and XML-related technologies as the preferred means of data exchange of the web, the majority of web content is still available in its traditional unstructured or semi-structured form. We consider that this situation is likely to change in the next few years. However, due to the amount of data to be converted to XML format, we believe that this transition will take quite some time.

- Wrapper generation toolkits are still unusable by non programmers. Since data integration is still the subject of research, user-friendliness is not a requirement. Thus, almost all available wrapper generation toolkits require advanced knowledge of computer science fields by their prospective users.

- The currently available wrapper generation toolkits are not usable in production environments, since minor changes in the formatting of the source documents require wrapper rewriting, a not-so-easy task.

- Since wrapper generation toolkits haven't been around for enough time, they lack basic system integration capabilities; they usually include non-thread-safe parts and export wrapped data in custom formats.

- Currently available HTTP-related Java classes don't provide the extended support for HTTP handshaking required by data integration applications. For example, support for HTTP connection timeouts, HTTP proxies and cookies, features supported by most web-browsers, have to be coded explicitly by the designer/implementor of the data integration applications.

Future plans include:

- Extensive caching/prefetching, using novel algorithms for storage, retrieval and full/partial hit/miss recognition[1].

- Moving the system to a distributed environment, where multiple C.I.A. servers will cooperate to answer to user queries in a locality-based, distributed, fault-tolerant way.

- Development of a thread-safe wrapper generation toolkit, exporting real DOM tree structures instead of XML data.

- Development of a faster query execution engine, since we expect XSL not to scale well for large DOM trees.

[1]For a detailed description, please refer to [18].

- Use of an XML native database, with a Java based X Query interface.

Appendix A

Technological & Software Choices

As already mentioned, the ease of integration of our application in many different environments and across many platforms was one of our primary design targets. That's why we used technologies and software that are either platform-independent or available for a great deal of software-hardware combinations.

A.1 Technologies Used

As far as technologies are concerned, we used either solutions based on the Java programming language, or on general-purpose communication protocols featuring implementations on mostly all known platforms.

In more details, we used the following technologies:

- the Java Programming Language, as implemented by the Java Development Kit v1.3 and v1.4 (Java 2 SE platform) specification.
- JavaServer Pages (JSPs) v1.1 (final).
- Java Servlets v2.2.
- Enterprise JavaBeans (EJBs), according to the JavaBeans Development Kit v1.1 specification.
- Java DataBase Connectivity (JDBC) v2.0.1.

- HTTP v1.1.
- Secure Socket Layer (SSL).

A.2 Software Used

The software used includes:

1. JDK v1.3.1 and v1.4.0
2. JBDK v1.1.
3. Tomcat Application Server v3.4.
4. Apache Web Server v1.3.19.
5. JDBC v2.0.1.

All of the above choices were preferred so that they satisfy the following requirements:

- They are based on the Java programming language and are therefore portable across all platforms for which the Java Run-Time Environment (JRE) is available.
- Alternatively, they are distributed under the terms and conditions of the GNU General Public License (GPL) or its modifications and are therefore available in source code in the public domain .
- They have been tested under real circumstances and work loads by reliable internet sites, companies and organizations etc.

Appendix B

Models of Database Connectivity

The main difference between these models is the way we connect to the database management systems (DBMS). In any case, the client-user connects to a World Wide Web server, according to the HTTP/1.1 protocol, also using SSL (Secure Socket Layer) for increased security. Furthermore, the final connection to the DBMS server is always done via a JDBC driver. The following models differ, then, in the kind of intermediates between the initial web server and the DBMS server. The resulting models are:

- The Direct JDBC Connection Model.
- The Connection through Web Server Model.
- The Connection through Specialized Application Server Model.

B.1 Direct JDBC Connection

As we can see in the following diagram, this model is very simplistic. The client sends a request to the initial server (called DSWS - Department Store Web Server), using the HTTP protocol. DSWS then connects directly to the DBMS (DataBase Management Server) using a Java Servlet utilizing a JDBC driver.

The main characteristics of this model are:

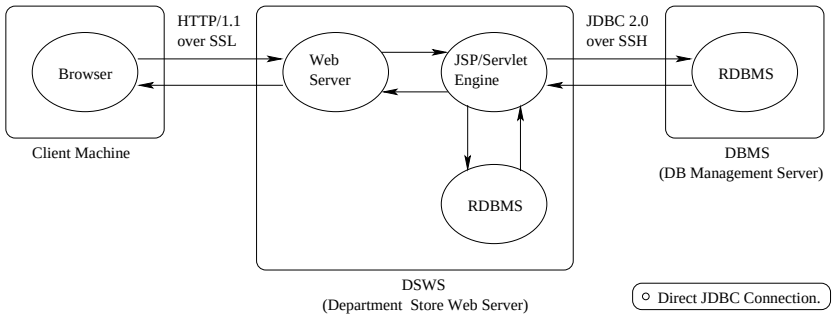
- (+) Efficiency due to the lack of any mediating parts.

by the DBMS's capabilities.

- (+) Simple design and easy implementation.
- (-) Increased parallelism can overload the DBMS to the point of a system crash (when the number of concurrent clients is overwhelming).
- (-) To achieve a satisfactory security grade for the system, we must use pure SSL or SSL-tunneled connections, to encode the connection elements. This mechanism, apart from being slow, is not one of the standard connection methods and therefore there is a possibility that it won't be available for some platforms.
- (-) Mixing the programming logic and the presentation makes the code complex and unreadable, and therefore hard to maintain.

Due to the above complications, this model was used only in the initial development stages.

Figure B.1: Direct JDBC Connection.



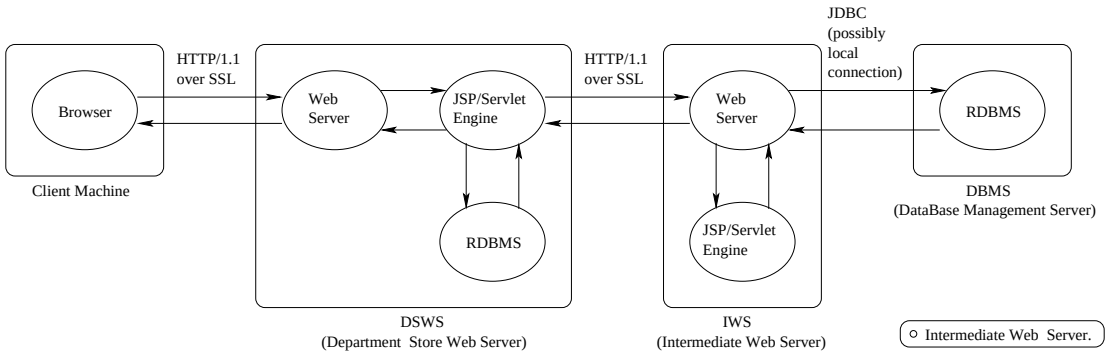
B.2 Web Server Model

This solution is more complex than the previous one but is also the most generic of the tree connection methods discussed here. We see below a diagram, in which the DSWS communicates with the DBMS via an intermediate web server. The main characteristics of this method are:

many and different hosts and the capability to support many different database management systems, provided that there exists a corresponding JDBC driver.

- (+) By using properly configured network architecture (i.e. existence of a firewall, installation of the web server and the DBMS on the same computer and configuring the latter to only accept connections from the former, etc.), this solution can prove to be secure against a wide range of know attacks.
- (+) The concurrency level is furthermore limited by the web server, which is a much efficient method, since web servers have been extensively developed and tested in production environments.
- (+) By using such techniques as connection pooling, we can improve overall efficiency and achieve better throughput than with the first method.
- (+) Limited concurrency reduces the risk of a successful denial of service (DOS) attack.
- (+) An e-shop can exist independently of an electronic department store.
- (-) The intermediate web server, can be a bottleneck for the overall performance.
- (-) The existence of many different web servers can complicate porting the application to many platforms.

Figure B.2: Intermediate Web Server Connection.



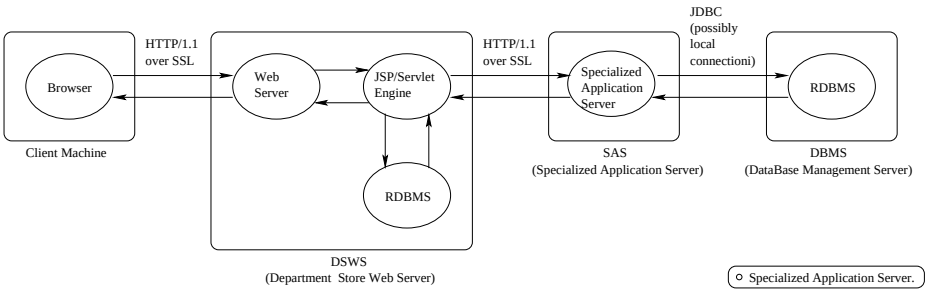
This was the solution-of-choice for the implementation of this application.

B.3 Specialized Application Server Model

This solution, since it is also based on the use of an intermediate part (the SAS (Specialized Application Server)), has all the advantages and disadvantages of the previous one, with the following differences:

- (+) The SAS allows for further optimizing the system's efficiency.
- (+) Security can be further hardened by using certain techniques in the SAS.
- (-) The SAS probably would also call for a specialized communication protocol.
- (-) The development time is further augmented by the amount corresponding to the development and testing of the SAS.
- (-) No e-shop can exist and function in an independent manner.
- (-) Specialized client-side software is demanded.

Figure B.3: Intermediate Specialized Application Server Connection.



Reference List

[1] World Wide Web Consortium, http://www.w3.org/TR/1998/REC-xml-19980210. *Extensible Markup Language (XML) 1.0*, February 1998. W3C Recommendation.

[2] J.D. Davidson and D. Coward. *Java Servlet Specification*. Sun Microsystems, Inc., v2.2 (final release) edition, December 1999.

[3] M. Hall. *Core Servlets and JavaServer Pages*. Sum Microsystems / Prentice Hall PTR, 2000.

[4] E. Pelegri-Llopart and L. Cable. *JavaServer Pages Specification*. Sun Microsystems, Inc., v1.1 edition, November 1999.

[5] K. Avedal, D. Ayers, T. Briggs, C. Burnham, A. Halberstadt, R. Haynes, P . Henderson, M. Holden, S. Li, D. Malks, T. Myers, A. Nakhimovsky, S. Osmont, G. Palmer, J. Timney, S. Tyagi, G. Van Damme, M. Wilcox, S. Wilkinson, S. Zeiger, and J. Zukowski. *Professional JSP*. Wrox Press Ltd., 2000.

[6] Sun Microsystems, Inc. *The Java 2 Enterprise Edition Developer's Guide*, May 2000.

[7] et al. R. Fielding. *HyperText Transfer Protocol (HTTP)*. Network Working Group, 1.1 edition, 1999. RFC 2616.

[8] K. Mpletsas. An Internet Content Integration System: the Front-End. Diploma Thesis, Technical University of Crete, 2002.

[9] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997. http://www-db.stanford.edu/ widom/pubs.html.

[10] S. Abiteboul. Query semi-structured data. In *Proceedings of the ICDT*, 1997.

[11] J. Robie, D. Chamberlin, and D. Florescu. Quilt: an XML query language for heterogeneous data sources. In *Proceedings of XML Europe*. Graphic Communications Association, 2000.

[12] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113. *Document Object Model (DOM) Level 2 Core Specification*, November 2000. W3C Recommendation.

[13] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113. *Document Object Model (DOM) Level 2 Views Specification*, November 2000. W3C Recommendation.

[14] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113. *Document Object Model (DOM) Level 2 Events Specification*, November 2000. W3C Recommendation.

[15] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113. *Document Object Model (DOM) Level 2 Style Specification*, November 2000. W3C Recommendation.

[16] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Html-20001113. *Document Object Model (DOM) Level 2 HTML Specification*, November 2000. W3C Recommendation.

[17] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113. *Document Object Model (DOM) Level 2 Traversal and Range Specification*, November 2000. W3C Recommendation.

[18] J. Giannakopoulos. An Internet Content Integration System: the Cache Manager. Diploma Thesis, Technical University of Crete, 2002.

[19] World Wide Web Consortium, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819. *XML-QL: A Query Language for XML*, August 1998. W3C Note.

[20] Z.G. Ives and Y. Lu. Xml query languages in practice: an evaluation.

[21] World Wide Web Consortium, http://www.w3.org/TR/2001/WD-xmlquery-req-20010215. *XML Query Requirements*, February 2001. W3C Working Draft.

[22] D. Chang and D. Herkey. *Client/Server Data Access with Java and XML*. Wiley Computer Publishing, 1998.

[23] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. In *Proceedings of the ACM SIGMOD Conference*, volume 1 of *29*, pages 68–79, 2000.

[24] N. Ashish and C. Knoblock. Semi-automatic wrapper generation for internet information systems. In *Proceedings of Cooperative Information Systems*, 1997.

[25] N. Kushmerick, D.S. Weld, and R. Dorenbos. Wrapper induction for information extraction. In *Proceeding of IJCAI*, 1997.

[26] C. Quix and M. Schoop. Metadata management for facilitating data integration in electronic marketplaces. Informatik V, RWTH Aachen, 2001.

[27] A. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference*, 1996.

[28] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. STRUDEL: a web-site management system. In *Proceedings of the ACM SIGMOD Conference*, 1997.

[29] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with STRUDEL: experiences with a web-site management system. 1998.

[30] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Overview of STRUDEL - a web-site management system. 1998.

[31] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Web-site management: the STRUDEL approach. 1998.

[32] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Declarative specification of web sites with STRUDEL. In *Proceedings of VLDB*, 2000.

[33] H.P. Schnurr, S. Staab, and R. Studer. Ontology based process support. Institut AIFB, Univ. Karlsruhe, 1999.

[34] M. Erdmann and R. Studer. Ontologies as conceptual models for XML documents. Institut AIFB, Univ. Karlsruhe.

[35] D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: the very high idea. In *Proceedings of the 11th International Flairs Conference*, 1998.

[36] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.P. Schnurr, S. Staab, R. Studer, and A. Witt. On2broker: Semantic-based access to information sources at the www. Institut AIFB, Univ. Karlsruhe.

[37] S. Decker, M. Erdmann, D. Fensel, and R. Studer. *ONTOBROKER: Ontology-based Access to Distributed and Semi-Structured Information.* Kluwer Academic Press, 1998.

[38] V.R. Benjamins, B. Wielenga, J. Wielemaker, and D. Fensel. Towards brokering problem-solving knowledge on the internet.

[39] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the IEEE Data Engineering Conference*, pages 251–260, March 1995.

[40] Y. Papakonstantinou and P. Velikhov. Enhancing semistructured data mediators with Document Type Definitions. In *Proceedings of the IEEE Data Engineering Conference*, 1999.

[41] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker: A mediation system based on declarative specifications. In *Proceedings of the IEEE Data Engineering Conference*, pages 132–141, March 1996.

[42] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of the Deductive and Object-Oriented Database Conference*, pages 161–186, December 1995.

[43] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proceedings of the ACM SIGMOD Conference*, 1999.

[44] L. Gravano and Y. Papakonstantinou. Mediating and metasearching on the internet. *Bulletin of the IEEE Computer Society, Technical Commitee on Data Engineering*, 21(2):28–36, 1998.

[45] B. Ludascher, Y. Papakonstantinou, P Velikhov, and V. Vianu. View definition and DTD inference for XML. In *Proceedings of the Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.

[46] K. Konopnicki and O. Shmueli. Information gathering in the world-wide web: the W3QL query language and the W3QS system. Computer Science Dept., Technion - Israel Institute of Technology.

[47] A. Saguguet and F. Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *Proceedings of VLDB*, 1999.

[48] J. Hammer, H. Garcia-Molina, S. Nestorov, and R. Yerneni. Template-based wrappers in the TSIMMIS system. Department of Computer Science, Stanford University.

J. Widom. Information translation, mediation and Mosaic-based browsing in the TSIMMIS system. SIGMOD Demo Proposal (final version).

[50] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databased and the design of Disco. INRIA.

[51] A. Tomasic, L. Raschid, and R Valduriez. A data model and query processing techniques for scaling access to distributed heterogeneous databased in Disco. In *IEEE Transactions on Computers, special issue on Distributed Computing Systems*, 1997.

[52] Enosys Markets. Enosys Markets: Architecture and product overview. Enosys Markets, Inc., 2000.

[53] R. Dorenbos, O. Etzioni, and D.S. Weld. A scalable comparison-shopping agent for the world wide web. In *Proceedings of the 1st International Conference on Autonomous Agents*, 1997.

[54] et al. R. Catell. *The Object Database Standard - ODMG 93.* Morgan Kauffman, 1993.

[55] G. Huck, P. Fankhauser, K. Aberer, and E. Neuhold. Jedi: Extracting and synthesizing information from the web. GMD-IPSI.

[56] S. Cluet, S. Jacqmin, and J. Simeon. The new YATL: Design and specifications. Technical report, INRIA, 1999.

[57] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[58] J. Simeon. *Integration de sources de donees hegerogenes ou comment marier simplicite et efficacite.* PhD thesis, L' Universite Paris XI, January 1999.

[59] The ARANEUS Project Home Page. http://www.dia.uniroma3.it/araneus.

[60] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. 1999.

[61] G. Mecca and P. Atzenti. Cut and paste. *Journal of Computing and System Sciences*, page 85, 1999.

[62] G. Mecca, P. Atzeni, P. Merialdo, A. Masci, and G. Sindoni. From databases to webbases: the ARANEUS experience. D.I.A. - Universita di Roma Tre, May 1998.

[63] G. Mecca, P. Atzeni, P. Merialdo, A. Masci, and G. Sindoni. The ARANEWS web-based management system. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[64] V. Crescenzi and G. Mecca. The ARANEUS wrapper toolkit: A tutorial. Adapted from [65], July 1999.

[65] V. Crescenzi and G. Mecca. Grammars have exceptions. *Journal of Information Systems*, 1998.

[66] B. Adelberg. NoDoSE - a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[67] B. Adelberg and M. Denny. Building robust wrappers for text sources. Computer Science Dept., Northwestern University.

[68] L. Liew, C. Pu, and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. Oregon Graduate Institute of Science and Technology.