Intelligent Wrapping of Information Sources: Getting Ready for the Electronic Market

Sebastian Pulkowski
Research Assistant
University Library &
Institute for Program Structures and Data Organization
Universität Karlsruhe
Germany
E-mail: pulkowsk@ira.uka.de

Abstract:
Literature search and delivery in the World Wide Web becomes a rapidly expanding market. Up to now the search is mostly cost-free. But in the future we expect the appearance of more and more providers charging for their services. The main problems are finding the right provider and extracting the information. UniCats is a system for intelligent information search and extraction from multiple providers Web sites. One important part of the system is the so-called wrapper. This paper concentrates on wrappers and their generators. The wrappers' task is the translation of a customer's query into the source's syntax and the re-translation of the answer. Applying these wrappers in an electronic commerce environment needs additional functionality, e.g., navigation through a provider's site, collection of the information the customer desires or cost pre-calculation. Because of the variety of the source's functionality, we need a flexible and individually built wrapper. This could be achieved using a modular concept. For a wide acceptance of our system, easy wrapper generation is important. Thus, we develop a wrapper generator which enables laymen with no programming skills to build an individual wrapper for an information provider in an electronic commerce environment.

1 Introduction

The World Wide Web can be seen as one big virtual library. Information about documents or even the documents themselves in electronic format can be found for nearly every subject area. Hence, literature search and delivery becomes a rapidly expanding market. Today, almost all booksellers and publishers place their offers on the Internet, and intermediaries that catalogue and index documents for search purposes assist users in the retrieval of relevant information. Almost all of these do so in the hope of making profit and, consequently, charge users and/or providers for their services. A case in point are the existing so-called research-centers like the Fachinformationszentrum Karlsruhe [FIZ]. Here, a user has to pay for both the search and the delivery. It is expected that even university libraries, where the search currently is for free, will in the future charge for the service of literature search.

One problem that has to be addressed, is the supply of the user in locating, assessing and using information providers within a bewildering array of services. This is a new challenge for systems that provide the service of information search and acquisition of electronic documents. To be effective, such a system has to select, negotiate with and combine providers to obtain the best possible deal for the user. In this process cost control represents a particular challenge. Costs arise in parallel in all visited sources. Combine this with the fact that costs often become known only as they arise, and it becomes quite clear that the user may only be able to react when it is already too late, and may do so only at the price of less than satisfying service quality.

Nor is cost control the only consideration. Another is service quality. No matter how the various information sources present their individual services, the user should be faced with a uniform presentation. Sources change their offers quite frequently, and continuously update their products, and these should immediately be visible to the user. Finally, security issues play an important role due to privacy consideration, subscription facilities and payment systems.

All these concerns are the subject of the UniCats[1] project at the Universität Karlsruhe. To do so it employs a federation architecture based on user agents, traders and wrappers. User agents interpret, augment and execute search and retrieval requests of human users, based on policies and preferences kept in user profiles. Traders assist user agents in the selection of suitable information sources by maintaining metadata about these sources, such as content and service fees. The wrappers adapt information sources so that user agents and traders can work as desired. They are responsible for executing user requests by utilizing the source's query facilities and translating results back into the format expected by the user agent. All of these components are part of an electronic market and thus have to address cost accounting and security issues.

This paper concentrates on the wrapper components. We present the concept of a wrapper that is flexible and beyond search of literature, is intended especially for employment in an electronic commerce environment. To avoid unnecessary complexity in case of unneeded functionality (the sources may offer its services free of charge or it may only have very limited security concerns), we use a modular concept which allows the easy building of wrappers. Further we assume that sources offer their information in the form of HTML pages. Our goal is a wrapper generator that

[1] a UNiversal Integration of Catalogues based on an Agent supported Trading and wrapping System

allows even laymen with no programming skills to build a wrapper for every information source with an HTML interface.
The paper is organized as follows: We begin in Section 2 with introducing the UniCats system, its architecture and components. The following section describes the heterogeneity of information sources and the problems for wrappers and their generators with these sources. Thereafter, we show our concept of a powerful wrapper to access these providers before we present in Section 5 our wrapper generator which allows everybody to build a wrapper. Related work is covered in Section 6. We conclude with an outlook on future work in
Section 7.

2 The UniCats Environment

To cope with the open, heterogeneous, and non-transparent market, we need a flexible architecture. It should be open for modifications and extensions and independent from concrete computer platforms.
We achieve the flexibility by the tree-component type architecture illustrated in Figure 1.

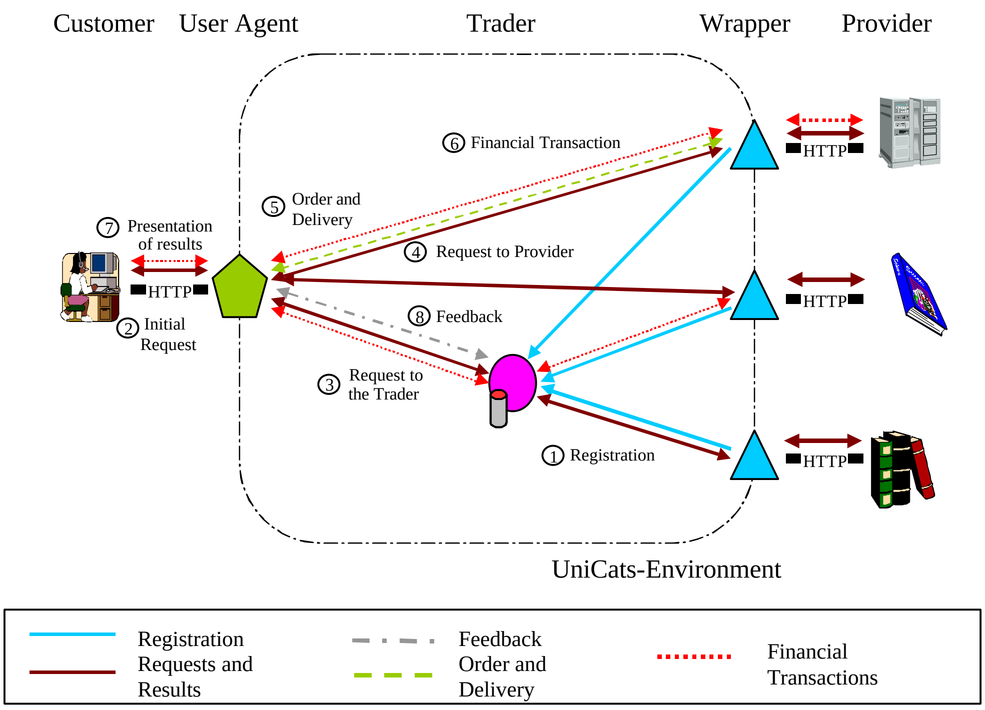


Figure 1 : The UniCats Architecture

2.1 Components

The three main components of the UniCats system [Chri98, Chr99a] are the *user agents* as proxies at the customer side, the *wrappers* as proxies for the information providers and finally *traders* for the intermediation between the two. Their functions and the interactions between them are as follows:

The user agent offers a uniform interface to the customer for a transparent access to the heterogeneous information providers. Another task of the user agent is to develop a search plan minimizing the search costs and executing this plan. The plan may involve by integration with the traders. As part of the execution individual queries are sent to the wrappers. Finally, the user agent has to collect the results, clean them, and display them to the user.

The task of the wrappers is the translation of queries into the special format of the associated information source and the transformation of results back from the source format into the result format of the UniCats environment. However, wrappers are much more than just simple translators. Wrappers are built individually for an information provider and present the whole spectrum of functions available to the customer, i.e., its user agent. Another task of the wrapper is the collection of metadata and their transmission to the trader.

The traders [Chr99] mediate between the two components mentioned above. They store information about the available service providers. For an incoming user request the trader tries to find the most suitable providers. To achieve scalability, traders may be organized into federations where information is exchanged between participating traders when ever necessary.

2.2 Interaction of the Components

Figure 1 illustrates the communication principle and information flow inside the UniCats environment. After a wrapper has been built for an information provider, the wrapper registers (1) with one or more traders and gives them a profile in the form of metadata. From the other end the customer submits its query to the user agent (2) which adjusts the query to the user profile and contacts the trader (3) using this information. The trader returns a list of recommendations for service providers together with additional information available. The user agent then constructs – perhaps dynamically – a query execution plan. While executing it, the agent addresses particular wrappers (4, 5). This can be done in parallel to more than one service provider. The wrappers transform the query into the native query format of the information source and send the re-transformed results back to the user agent.
The results from all the wrappers have been collected and integrated in one overall result they are displayed to the customer (7). Finally, the user agent can give a feedback (8) to the trader about the success of the query, so that the trader can take the feedback into consideration for further user requests.

In the future, this open market will include financial transactions (6). We designed the platform in a way that offers a simple way to add components for certifying, accounting, billing and electronic payment.

To guarantee the homogeneity of the system towards the user in the face of widely heterogeneous information sources and large-scale networks, we need powerful wrappers and generators for wrapper construction. Before we introduce our concepts of wrapping and wrapper generation, we show some problems and challenges which can be expected in an open market for literature search in the next section.

3 Challenges for Wrappers and their Generators

Heterogeneity of Source Functionality

Apparent obstacle to large-scale wrapping of information sources is the wide divergence of functionality in the different sources. We may classify the providers on the literature market in four categories with the following characteristics:
Providers which offer online information and documents without charging money. Examples are search engines of university libraries or other university institutions which collect large amounts of papers and other documents. In this case, the search and all the information is free and can be used or downloaded without the requirement of a login or charging for it. Examples for such servers are the NCSTRL [NCSTRL] or the Liinwww servers [LiinWWW].
Another class of providers offers some of their services only to a restricted group of people, e.g., a university library for its students: While everybody can search in the catalogues of the university library of Karlsruhe [UBKA], reservations or lending of books is restricted to students. The service is still free.
An further class of providers require to login before accessing information. The customer has to pay an annual flat fee for this login. After logging in the customer can download whatever he or she likes. An example for that sort of source is, e.g., the online-server of the Springer publishing company [Springer].
Finally, a source may offer on-demand accounting, i.e., offer a login. The search for information itself costs money. The costs depend on the actions performed to get the requested information. An example for this class is the Fachinformationszentrum-Karlsruhe [FIZ].

Generation of wrappers must take place on an individual basis with the functionality adjusted to their particular requirements like login, payment etc. A uniform wrapper for all the providers is out of the question. Hence a compromise must be found between individuality of a wrapper suited for an information source and easy creation of the wrapper.

Site Navigation

Wrappers for information sources in an electronic commerce environment differ approximately from existing wrappers for single HTML pages. The page-wrapping and the extraction of content off the HTML page is only one small part of the wrapper. Usually, the information needed is not located on a single page but must be collected from several pages by navigating through the source. The task of the wrapper is to plan and perform this navigation and to decide which information to collect.

Search Costs

Cost accounting is non-trivial in case of demand accounting. It depends on the actions performed, the number of documents shown to the customer, the amount of time for the search, or even the size of downloaded files. It is true that some of the providers offer the possibility to obtain cost information, but this information only covers the costs incurred so far. Actually, no provider will calculate the costs before the action is performed. However, this is required if we search in parallel in multiple information sources. A pre-calculation of costs would help the user or his agent to decide whether to cancel or to continue a search request.

Wrapper Location

Still another problem is the wrapper location. Because of the mass of information sources, we cannot expect that every source is willing or even capable to create a wrapper for our UniCats system. In general, there may be wrappers that run at the provider’s site, at the customer’s institution or even as an independent third party somewhere in the World Wide Web. Clearly, these differences raise novel challenges with regard to security, trustworthiness and accounting.

Wrapper Generation

If the customer’s institution or a third party wants to integrate multiple information sources with the help of wrappers, wrapper generation must be easy and quick. Wrapper generation must be assisted by the system if we assume that expert knowledge about wrapper generation is not widely available. For example, a librarian in a university library has the knowledge for literature search but for the most part no programming skills.

Metadata Collection

Traders can only decide whether a source is appropriate for a user request if they have information about the source and its content, so-called metadata. It is the task of the wrapper to collect these data and send it to the traders to make a provider public known. In the vein of automatic wrapper generation, this metadata should be collected automatically, for example, during wrapper generation.

Robustness of Wrappers towards changes

Modifications of HTML pages, which may occur quite often, could cause problems. For a wrapper in a cost-based environment, adjusting to the modifications is essential. Since the customer has to pay for the search, he has every right to get correct data. A wrapper has to recognize changes in the HTML pages, and quickly adapt to these changes.

In the remainder of the paper we develop a solution for wrappers and wrapper generators that meet these challenges.

4 The UniCats Wrapper

4.1 The Modular Concept

In UniCats we have to deal with both, conventional providers such as university libraries and providers of the electronic commerce environment. To cover the whole spectrum of conceivable functionality, we had to find an architecture which is on the one hand very flexible and on the other hand powerful enough to give satisfaction to the provider and the customer. This is best done by a modular approach, i.e., by dividing wrapper functionality into modules with different functionality.

During wrapper generation a module is inserted it its (specialized) functionality matches the desired functionality.
We distinguish two sorts of modules: *basic* and *supplementary modules*.

Basic Modules

Some functions are required in every wrapper, whether it s a wrapper for a commercial or a non-commercial provider. These so-called *basic modules* cover functionality like

- connection establishment and termination to the provider;
- validation of customer's search request to avoid incorrect requests;
- translation of searches into the source's syntax;
- re-transformation of the source's answer;
- checking the syntactical correctness of results to recognize changes on the HTML pages;
- collection of metadata during the search activity for a better pre-calculation of possible results.

These modules are sufficient to form a standard wrapper for a conventional information provider outside an electronic commerce environment. Any tailoring of source functionality is done by adding *supplementary modules*.

Supplementary Modules

These modules cover the following functions:

- cost control and administration;
- cost-optimal planning of a search;
- guaranteeing secure data transmission;
- informing a trader if the provider's metadata changes;
- session accounting and administration;
- extra login for a wrapper;
- extra charging of money for the wrapper's work.

4.2 The Architecture of a UniCats Wrapper

In Figure 2 the scheme of a UniCats wrapper for a source in an electronic commerce environment is shown.

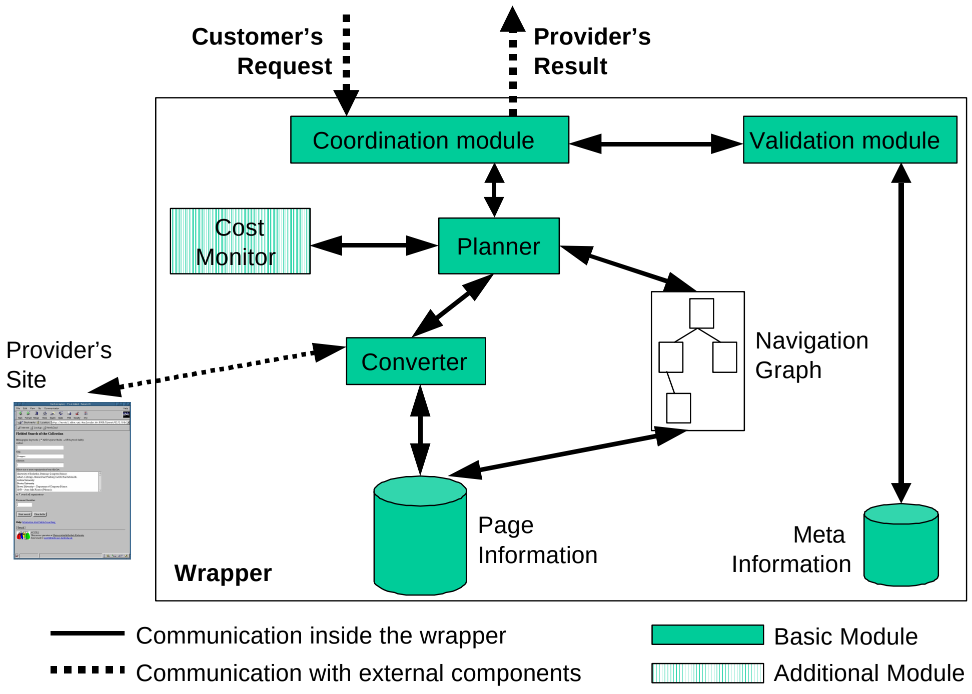


Figure 2: The Wrapper Architecture

The communication with the wrapper begins when the wrapper receives the user agent's *search request* in XML in the *coordination module*. First, the request is checked for syntactical correctness. This is done by the *validation module*. To perform this check, the module needs information about the contents of the pages and the possibilities of requesting information, such as truncation, possible connections of search attributes or required attributes. This *meta information* which is needed to fill out forms, is stored in a meta-database. If the request is not syntactically correct, the validation module tries to correct the request. If this fails, the wrapper informs the user and requests additional input. Otherwise the (perhaps modified) search can be given to the *planner*. The task of the planner is to search all relevant pages and construct a *navigation plan*. In this plan, all these pages are inserted in the right order. This is done by entering a *navigation graph*. This navigation graph is a sort of scheme of the source which includes all search relevant pages with their attributes and additional information, such as links between the pages. The planner now tries to find a path in this graph on which all required attributes can be found. To get this graph, the planner translates links into new actions, e.g., following a dynamically generated link to gather information about a special document. The planner uses on the one hand meta information about the source, the same which was used during the process of request validation. On the other hand, information about the pages and the content representation is required to extract the information from the result pages. This information is stored in a second database, called *page information*. Both databases are created semi-automatically during the process of wrapper generation. When the planner creates the *navigation plan*, the cost for this plan has to be calculated and compared with the user-given conditions or the maximal cost, respectively. This is done in the so-called *cost monitor*. If the cost is too high, the search is stopped, the customer is informed and the wrapper waits for the customer's decision. For example, this decision could be increasing the cost limit, reducing the required attributes, or canceling the search.

If there is no conflict after cost calculation, the planner executes the query plan transmitting the search request to the *converter*. The converter fills out the forms, follows links and finally extracts the information from the provider's result pages. This is done with the above mentioned *page information*.
The provider result returned is compared to the navigation plan, and unless all the required information has been collected, the planner begins to re-calculate the plan, checks the expected costs again and finally executes the next part of the plan. If the whole request is performed, the result is transmitted back to the user by the *coordination module*.

In cost-based sources both the provider and the customer have an interest in a secure data exchange. On the one hand, the user often has to transmit personal information such as logins, addresses, or even credit card data. On the other hand, the provider sends information about documents or even the documents themselves. We inserted a security module in our wrapper controlled by the *coordination module*. This module has different security levels which can be negotiated between both components (see [Frü99] for details).

Of course, the quality of navigation planning and the following plan execution can differ from one wrapper to another. This can have different reasons: Normally, the provider itself possesses more metadata about the source than an external institution. Thus, the provider has more possibilities to use these metadata for the pre-calculation of the request. But a pre-calculation of costs and user information in case of exceeding this limit is rather implausible, because the provider also wants to earn money with the service it offers.

There are good reasons not to restrict the wrapper generation to the information provider. Instead, the customer's institution like a university library or a third party may build their own wrappers implementing different search strategies. For a university library, e.g., it would be possible to restrict the number of results to reduce the costs. Further cost reduction could be achieved by demanding that a request have at least a given number of attributes so that the search request becomes much more precise.
Another restriction could concern the interaction with the user: A defensive wrapper could ask the customer before performing an action, e.g., ordering a document. A rather offensive wrapper could, once instructed, navigate through the source and collect all the information it can get, regardless of the costs. With the existence of wrappers with different search strategies a competition among wrappers in the information market becomes possible.

5 Wrapper Generation

In Section 3, we described some problems concerning wrapper generation. Here now, we want to show a concept of a wrapper generator which enables a simplified wrapper generation. For a wide spread acceptance, this simple generation of a powerful and individual wrapper is the most important requirement. It must be possible even for a layman who has little programming skills to build a wrapper for an information source. Therefore, we use the approach "wrapping by example".

5.1 Wrapping by Example

Figure 3 shows the principle of wrapper generation: The person building the wrapper, whom we will call administrator, begins with one page, most often the first page in the search process: an initial search form. This page is loaded into the generator and parsed by the generator. The administrator defines the page's content and the way the information is displayed with the help of the generator (see the next subsection). This data is stored in the *source specific database*. Doing this, the generator tries to extract as much metadata as possible automatically. But sometimes not all the information which is necessary can be extracted automatically, so the administrator has to add information manually guided by the wrapper generator. During the process of content specification of the page, a meta-database is created automatically. This metadata include the kind of page, e.g., form, result list, or specific document information, links on the page or the attributes on this page, for example, the title, author, etc.

Thereafter, the administrator follows the link to the next step of the search, and the process of page specification re-starts until all search relevant pages of the provider's site have been added to the wrapper. The actions of following a link from one page to another is registered by the wrapper generator and an internal scheme of the source in form of a graph is build automatically (see below).

To make the existence of the wrapper public, some of the metadata will be sent to traders. In Figure 3, the metadata is divided into two parts: One part is *public data* and serves a trader for its profile calculation. The other part is *private* and the access is restricted to the wrapper only. Here cost information, detailed page information or data of previous requests could be stored for a use during the wrapping process, e.g., to calculate the query plan.

When the administrator has finished the specification of all search relevant pages, the generation process concludes with the generation of the wrapper. During this generation all modules needed are combined and merged into the wrapper.

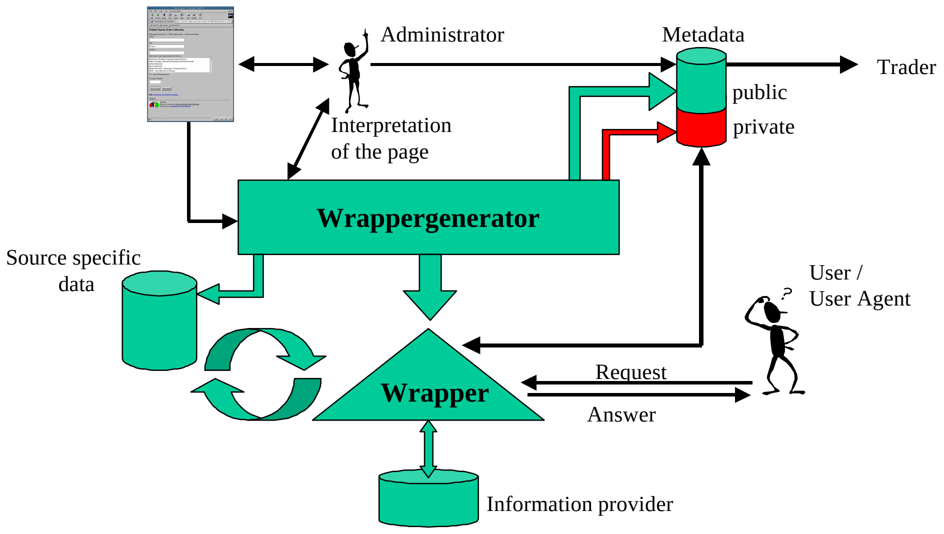
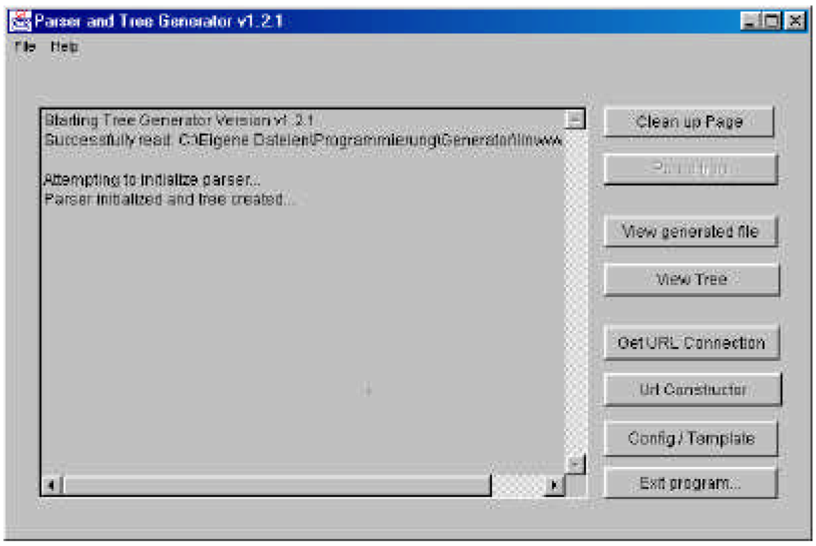


Figure 3: The process of wrapper generation

Then, the user can begin to query the source. After receiving a user request the wrapper contacts the *information provider* using the *source specific data* for the transformation of requests and results and the *metadata* for query planning.

5.2 The generation process

In this subsection, we want to show some details of wrapper generation: In Figure 4, we show the interface with the buttons for loading and parsing the pages.



```
Starting Tree Generator Version v1.2.1
Successfully read: C:\Eigene Dateien\Programmierung\Generator\Unwww

Attempting to initialize parser...
Parser initialized and tree created...
```

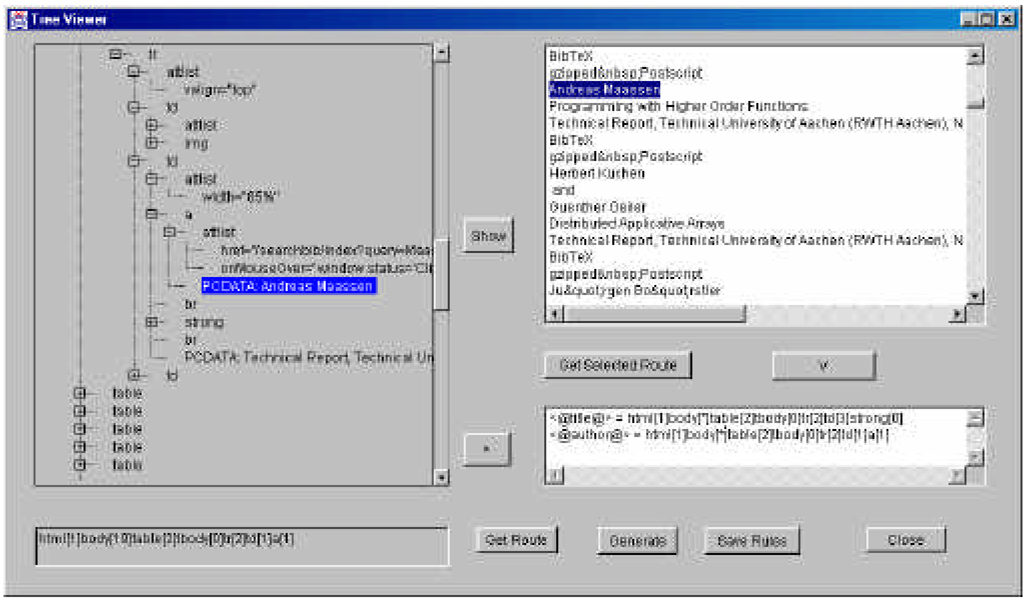
Figure 4: The user interface for wrapper construction

In the window we see a text window for messages and buttons for the generation process. The actions important for an administrator are parsing the tree, viewing the generated tree, the URL-constructor and viewing the generated configuration file.
The process of wrapper generation for a page begins with loading a page from the Web. This can be done by two ways: first, the administrator can insert an URL with the help of the URL constructor and the generator receives the page directly from the internet. The alternative way is loading the page from a file. Normally this is the more convenient way because the user can use the browser and knows that the URL is correct.

After loading a page into the wrapper generator the generator checks, if the page's syntax is correct. This check became necessary because we noticed that almost all pages are incorrect. Often a begin-tag exists but the end-tag is missing or the page contains an incorrect interlocked combination of begin- and end-tags, e.g., in a tabular environment. Normally a Web-browser can handle these problems using compensation algorithms, but these algorithms are browser specific and not public available, so we had to enlarge our wrapper generator with this functionality.
After this check, the generator tries to correct errors by editing the page automatically. We made the experience that in about eighty percent of all cases this correction ends successful. In the remaining cases the administrator has to help the generator by inserting the tags by hand.

When the page is syntactically correct, a parser constructs a corresponding parse tree. This tree begins with the <HTML> (begin HTML) tag of the page. Then the next element in the hierarchy, for example the <HEAD> (begin HEAD) tag is inserted as a sub-node of the tree. The tree ends

with atomic nodes such as strings. With the help of the tree it is possible to identify each element on an HTML page.
In Figure 5 the window with the tree on the left hand side is displayed.



```
⊟─ tr
   ⊟─ attlist
   │  └─ valign="top"
   ⊟─ td
   │  ⊞─ attlist
   │  ⊞─ img
   ⊟─ td
      ⊟─ attlist
      │  └─ width="85%"
      ⊟─ a
      │  ⊟─ attlist
      │  │  ├─ href="/search/oib/index?query=Maa
      │  │  └─ onMouseOver="window.status='Cli
      │  └─ PCDATA: Andreas Maassen
      ├─ br
      ⊞─ strong
      ├─ br
      └─ PCDATA: Technical Report, Technical Un
   ⊞─ td
⊞─ table
⊞─ table
⊞─ table
⊞─ table
⊞─ table
```

```
BibTeX
gzipped&nbsp;Postscript
Andreas Maassen
Programming with Higher Order Functions
Technical Report, Technical University of Aachen (RWTH Aachen), N
BibTeX
gzipped&nbsp;Postscript
Herbert Kuchen
 and
Guenther Geiler
Distributed Applicative Arrays
Technical Report, Technical University of Aachen (RWTH Aachen), N
BibTeX
gzipped&nbsp;Postscript
Ju&quot;rgen Bo&quot;rstler
```

```
<@title@> = html[1]body[*]table[2]tbody[0]tr[2]td[3]strong[0]
<@author@> = html[1]body[*]table[2]tbody[0]tr[2]td[1]a[1]
```

```
html[1]body[1 0]table[2]tbody[0]tr[2]td[1]a[1]
```

Figure 5: The generated tree and the rule extractor

In the environment of literature search the HTML pages are generated dynamically. In particular, during a provider request the number of results cannot be foreseen. So, we need a wrapper which is independent of the number of results on a page. We decided to solve that problem by defining templates for information extraction. In these templates a path scheme from the root node to the desired node can be defined for each element. This procedure is only possible, because of the uniform structure of dynamically generated HTML pages. An example for such a path description from the root to a table entry is the following:

html[1]body[3]table[2]tbody[0]tr[2]td[1]a[1]

This representation means, that the element can be found by following the second element after the<HTML>-tag, the fourth element in the <body>-tag etc. (The first element has the number [0].)

Of course, such a representation could be hardly defined by an administrator without help. So we developed a tool which allows the specification of these paths using a graphical and textual representation of the page. The tool is shown in Figure 5. On the left hand side of the window the administrator sees the automatically generated tree with all search relevant elements. For a better clarity, tags which are not relevant for a search are not displayed. On the right hand side of the window the HTML page can be seen in text format. Only those parts of the HTML page which are also visible in the Web browser are shown. Now, the administrator can mark elements in the HTML page using both the tree representation or the text field. The corresponding part in the alternative representation will be displayed immediately. When the administrator finds a relevant part of a page, e.g., the author’s name, the rule for extracting this element is automatically calculated and displayed in the window at the lower left side. Then the administrator specifies the name of the element, in this case the author and adds the rule to the template by pressing the *get route*-button. Because we don't want to access only one element of a page, we need a definition to tell the wrapper to extract all authors. So, we use an ambiguous letter (*) in the template, which means that the information of all paths matching this structure have to be inserted into the result. Thus, the rule for extracting all authors from a page could be defined this way:

```
<@author@> = html[1]body[*]table[2]tbody[0]tr[2]td[1]a[1]
```

Similarly, all the other attributes of one data set, for example, the title, the year etc., must be defined as well. Altogether they form the template for the result extraction. To allow a structuring of the result we inserted a blocking-operator: (<@block@>). This operator tells the wrapper how to extract the elements during the wrapping process. For example the block for one result could be

```
<block>
        <@author@>
        <@title@>
        <@abstract@>
</block>
```

This blocking operator would not be necessary, if we had only simple structures on a page. But often the abstract on a page is not on a one string basis but consists of two or more strings because of the length. So we have to define another block inside the result block for the abstract. These interlocked blocks are also very useful if a page contains a table in a table environment. Using this blocks, even very complex structures on an HTML page can be defined.
With the help of this template, the wrapper generates the result, which is send to the customer. We use a data-format which is XML-like, because the usual exchange format between the user agent and the wrapper is XML. But the wrapper is also capable to generate results in alternative formats like a text-file, a latex or even an HTML file. With this format independence the wrapper gets interesting for other systems, too.

When a page has been fully specified, the administrator can select a link to the next page in the search progress and continue the process with specifying this page. The wrapper generator recognizes the move to the next page and generates the template for the specified page and stores it in the so-called *navigation graph*. This graph is a scheme of the source including all pages with their attributes and the links between the pages.

```
link ::= (n_link, n_page_from, n_page_to, cond, l_sort)

cond ::= (cost, sort_next_page, av_load_time, serv_change)

page ::= (n_page, n_templ, attr_list, p_sort)

attr_list ::= (attribute, value; ... ; attribute, value)

p_sort ::= {form, result list, docu_information, online_doc}

online_doc ::= (name, size, format, add_cost)
```

Figure 6: The scheme definition

In Figure 6, the definition language for the navigation graph is shown. We have two main elements: Links and pages. A page has a number, a belonging template, the kind of page and a list of attributes which could be found on that page.
A link has a number, a starting page and a target page, conditions for following this link and a sort (see below). The conditions include costs, the kind of the next page, an average load time for pre-calculation of the overall search time and a flag whether the link causes a change of servers. The latter attribute is also important for the pre-calculation, because in case of a server change there might be a delay if the target server is down.
We distinguish four different types of pages: forms, pages with more than one result which is a point of decision in the planning process, a page with detailed information to one document and finally if the target page is an online document. If it is a document, we can characterize it by a name, the size of the document, the format and additional cost for delivery like postage and packing material.

5.3 Navigation graph construction

With the help of this definition language we build a graph for all search relevant pages of a provider’s Web site. A sample graph for the Fachinformationszentrum Karlsruhe is shown in Figure 7. Insertion of additional back-links into the graph makes a navigation possible. The functionality is similar to the back button of a web browser. So, previous results can be stored in the wrapper and can serve as a starting point for the next search operation.

This graph displays the site structure for the Fachinformationszentrum Karlsruhe (FIZ). The nodes represent the pages which have been specified in the process described above. Links display possible connections between the pages. In the figure we can distinct three types of links:

- links without conditions: These are links that will not cause costs or a change of servers
- links with additional information: these are links which are important for the calculation of the best operations to get the final result
- links that are not on the provider’s site but added by the wrapper-generator to allow a faster calculation: The target of these links are pages with intermediate results. Following such a link is equivalent to the use of the “back” button of a browser.

With this graph, we could calculate all actions which must be done to get the most information as cheap and fast as possible.

When the administrator has specified all search relevant pages and inserted them in the graph the wrapper generation ends with metadata generation. During this process all the attributes and cost information is collected and inserted in a trader provided form. With the delivery of this form to the trader the wrapper gets known to the search system and can be accessed by the user agents. Of course the volume and the quality of metadata depends on the provider of the wrapper.

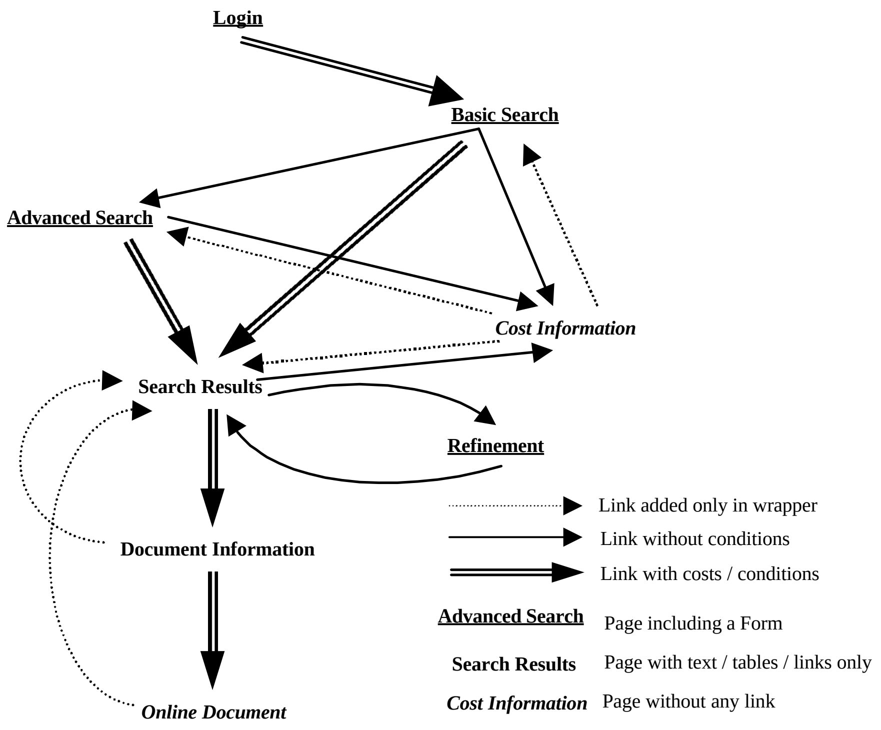


Figure 7: A navigation graph used by our wrapper

5.4 Adaptation of wrappers

Because a web page has an average lifetime of about three months a fast adaptation of the wrapper to the new page is very important. This gets even more important if the provider or the institution providing the wrapper charge for their service or the wrapper works upon a cost based provider. A customer paying for information he or she cannot use will probably use an alternative wrapper or information source next time. Thus, we need a concept for performing fast changes on the wrapper. The wrapper is built to always verify the result pages from the provider. If it cannot match the actual page with the template for this page, i.e., the corresponding tree for an HTML page, the wrapper stops the actions immediately and informs the administrator. Parallely this information is inserted immediately into the navigation graph. The page gets the status “invalid”. Further requests could the be calculated with this new graph. But, if a customer requires information, which could only be extracted from this page, the request will be rejected and the session ends.

The administrator has to adapt the wrapper to the new page layout and finally add this page into the navigation graph and replace the old page. The adaptation begins with loading a page into the wrapper generator. Then the parser generates the corresponding tree for the new page and builds a structural tree out of this page tree. The original tree cannot be used because of the dynamically generated results. Then the two basic structures are compared with each other and differences marked in the new tree. Then, the administrator can specify the content of new parts or delete invalid parts. After that, a new template file must be generated and the navigation graph updated. An advantage of this processing is, that the adminstrator does not have to re-build the whole wrapper in case of changes. Only the pages which changed must be specified again.

The same process takes place, if an existing wrapper has to be enlarged. So, a provider can built a first wrapper prototype for an information source and test the acceptance. Then, the existing wrapper could be enlarged, for example, with login, payment or security functionality.

In the next section we introduce other projects that build wrappers and other projects that use a sample approach for wrapper generation.

6 Related Work

Wrapping of semistructured information sources is the topic of several research projects. A good overview is given in [Flo98]. A lot of these projects developed query languages for Web pages such as WebSQL [Aro97], WebLog [Lak96], W3QL [Kon95] or Florid [For97]. These languages offer possibilities to formulate queries on semistructured pages and define rules for information extraction. To build wrappers with these languages a good knowledge of HTML and of the programming language and programming skills are required.

The Araneus [Atz97] project also offers a language to extract relational data from the Web. To do this, the page scheme must be defined in the Araneus Data Model (ADM). The language ULIXES is used to navigate within the link structure of the pages and to build relational views over the extracted data. But in this system, the wrapper creation expects knowledge of the language and the site structure as well. Functionality like query planning or access restriction to parts of the Web site, like in a university library (see above), are not provided.

Another system to present the data in Web sites in a graph representation is Strudel [Fer98]. But this system also offers neither electronic commerce functionality nor a generator to construct these wrappers for Web sites.

Another project which is the closest to ours and which offers a wrapper generator is W4F [Sah99]. They have a language, HEL, to describe the HTML page as a tree. They built an editor for wrapper construction. The extraction rules are generated when the user clicks on a Web browser on the piece of information he wants to collect. But to offer this functionality the system is restricted to the internet Explorer from Microsoft. The user of the wrapper generator also has to insert the data in the editor by hand. Because of the restriction to this single Web browser, the generator only works under Microsoft Windows. In the often very heterogeneous environment of operating systems like AIX, Solaris or Linux the wrapper generator can not be used. The generator also offers no functionality to model navigation in a source or electronic commerce functionality like payment, cost calculation or security.

We restricted our wrapper to the data search and conversion in an electronic commerce environment with no additional mediation functionality. But we believe that the wrapper can be used outside our project and that it can easily be included in existing integration systems like Garlic [Haa97] or TSIMMIS [Ham97].

7 Conclusion and Future Works

In this paper, we have presented the architecture of the UniCats wrapper which has the functionality to include information providers in an electronic commerce environment. The location of this wrapper is not restricted to the provider's side. On the contrary, with the wrapper generator each institution can build their own wrappers for an information source. The only requirement is the existence of an HTML Interface to the information which exists for nearly all providers.
Because of the simple interface to the user agent with data exchange in XML we believe that the wrapper is not restricted to the UniCats system, but it can easily be included into existing integration systems like Garlic or TSIMMIS. Of course, the wrapper can only work with full functionality such as secure data transmission, administration of logins or pre-calculation of costs, if all participant components use the UniCats protocol.

Using such a wrapper in Germany causes a lot of legal effects. For example, we have to solve problems like an assurance of solvency of a wrapper, because lots of private data such as credit card information, logins or personal addresses are administered by the wrapper. At the same time, the wrapper can extract lots of information out of the actions a user performs while searching. This knowledge would be very interesting for companies selling products on this information market like booksellers or publishers. When we expand the scope of our system to an international platform, we have to deal with international rights in addition to these problems. Therefore we have to find a concept which allows to construct a wrapper which can deal with these problems and guarantee the legality of the delivered information.

In the future, we plan to extend the functionality of request pre-calculation based on metadata. The goal is to calculate costs before they arise with the help of data collected from previous requests. If we could estimate how many results a search request will produce, we can warn the customer. The request ends in the wrapper and could be refined and posed again with more detailed information. Of course, this pre-calculation is more appropriate for a customer's wrapper than for a wrapper built by the information source. With these differences in functionality, quality and perhaps prices of wrappers we could achieve a market where wrappers compete with each other.

Finally, we want to enlarge the wrapper generator with another module, so that the search strategies could be adjusted using different parameters. Therefore, we have to identify relevant parameters, identify relations between them and find a model for optimal parameter combination. So we can promote a market of competitive wrappers with individual search strategies on the information market.

References

[Aro97] Arocena G., Mendelzon A., Mihaila G. (1997). Applications of a Web Query language. *In Proceedings of the 6th International WWW Conference*, Santa Clara, April.

[Atz97] Atzeni P., Mecca G., Merialdo P. (1997). To Weave the Web. In *Proceedings of VLDB, Athens*, Greece, 206-215.

[Chr98] Christoffel M., Pulkowski S., Schmitt B., Lockemann P.C. (1998) Electronic Market: The Roadmap for University Libraries and Members to Survive in the Information Jungle. *SIGMOD RECORD, 27(4)*, 68-73.

[Chr99] Christoffel M. (1999) A Trader for Services in a Scientific Literature Market. In *Proceedings of the 2nd International Workshop on Engineering Federated Information Systems (EFIS'99)*, 123-130, Kühlungsborn, Germany, May 5-7.

[Chr99a] Christoffel M., Pulkowski S., Schmitt B., Lockemann P.C., Schütte C. (1999) The UniCats Approach – new Management for Books in the Information Market. In *Proceedings of the International Conference IuK99 – Dynamic Documents*, Jena, Germany, March 22-24.

[FIZ] Fachinformationszentrum Karlsruhe (FIZ), http://www.fiz-ka.de.

[Fer98] Fernandez M., Florescu D., Kang J., Levy A., Suciu D. (1998). Catching the boat with Strudel: Experiences with a web-site management system. In *Proc. of ACM SIGMOD Conference on Management of Data*, Seattle, WA, 414-425.

[Flo98] Florescu D., Levy A., Millstein T. (1998). Database Techniques for the World Wide Web: A Survey. *ACM SIGMOD Record, 27(3)*, 59-74, September.

[Fro97] Frohn J., Himmeroeder R., Kandzia P.-Th., Lausen G., Schlepphorst C. (1997). Florid – a prototype for F-logic. In *Proceedings of the 13th International Conference on Data Engineering*, April 7-11, Birmingham UK, 583.

[Frü99] Früauf, T. (1999). Konzeption und Entwicklung einer Verbindungs- und Koordinationskomponente für den Wrapper im UniCats-Projekt. *Diploma-Thesis, Institute for Program Structures and Data Organization, University of Karlsruhe*.
[in German]

[Haa97] Haas L., Kossmann E., Wimmers E., Yang J. (1997). Optimizing queries across diverse data sources. In *Proceedings of the 23rd VLDB Conference,* Athens, Greece, August
276-285

[Ham97a] Hammer J., Garcia-Molina H., Nestorov S., Yerneni R., Breunig M., Vassalos V. (1997). Template-Based Wrappers in the TSIMMIS System. In *Proceedings of the 26th SIGMOD International Conference on Management of Data*, Tucson, Arizona, May.

[Kon95] Konopnicki D., Shmueli O. (1995), W3QS: A query system for the World Wide Web. *In Proceedings of VLDB*, Zürich, Switzerland, 54-65.

[Lak96] Lakshmanan L.V.S., Sadri F., Subramanian I.N. (1996). A declarative language for querying and restructuring the WEB. In *Workshop on Research Issues in Data Engineering*, 12-21.

[LiinWWW] LiinWWW: http://liinwww.ira.uka.de/bibliography/index.html

[NCSTRL] NCSTRL, http://ncstrl.ubka.uni-karlsruhe.de:8080/

[Sah99] Sahuguet A., Azavant F., (1999). Building bight-weight wrappers for legacy web data-sources using W4F. *In International Conference on Very Large Databases (VLDB'99)*, September, Edinburgh, Scotland, 738-741.

[Springer] Springer, Forum for Science http://link.springer.de/forum.html

[UBKA] Universitätsbibliothek Karlsruhe http://www.ubka.uni-karlsruhe.de/