ABSTRACT
In this paper, we describe the Semantic Contents Acquisition Framework (SCAF), an approach to retrieving content from the web and other kinds of information or data sources. It is based on extending standard XSLT with elements for the access to and storage of data. The extensions, embedded into a management and storage infrastructure, provide an entry point to general and personalized harvesting of dedicated sources. In addition to the general presentation of the extensions we have implemented, we present results of experiments performed on automatic retrieval of typed data from semi-structured web sources.

Categories and Subject Descriptors
H.2.5 [Database Management]: Heterogeneous Databases;
H.3.7 [Information Storage and Retrieval]: Digital Libraries

General Terms
Algorithms, Experimentation, Performance, Reliability

Keywords
XSLT, web content extraction, semi-structured data, hidden web

1. INTRODUCTION
With the growth of the World Wide Web in the last years, and the vastly increasing amount of accessible data, the need of content extraction from the web and other information sources has risen enormously. Since the available data usually is semi-structured, or even completely unstructured, and distributed over various sources which again provide their own formats, data often has to be assembled and transformed into the required structures first, before it can be used for further processing. This growing demand for contents extraction from different sources and their consolidation was recognized and led to the development of numerous mechanisms and frameworks for supporting and automating the extraction process.

We claim that the integration of standard data transformation techniques and a sophisticated data validation and management infrastructure provides a highly flexible and robust mechanism for data acquisition.

The remainder of this paper is organized as follows: We begin in section 2 by describing our approach which consists of the infrastructure components and the extensions to the transformation mechanism. In section 3 we describe the experiments we performed with extracting structured information from a set of standard web sites and discuss their results. Section 4 covers similar solutions which can serve to compare the capabilities of our system. Section 5 finally covers our ideas of advanced preprocessing and validation mechanisms to further improve the usability and performance of our system.

2. APPROACH
The “Extensible Stylesheet Language” (XSL) [6] is a family of recommendations for defining XML document transformation and presentation. Its most important part XSLT (XSL Transformations) is responsible for template and flow definitions, while the “XML Path Navigation Language” (XPath) is an expression language used by XSLT to access or refer to parts of an XML document.

Although, the specifications have defined a comprehensive set of functions already, the specific task of transforming content into semantic representations imposes the need for additional functionality. Fortunately, the specifications already define a mechanism for extending the functionality and elements of XSLT, which we exploited for our system.

Different XSLT processors provide slightly different ways to integrate the extensions, however, which requires us to carefully select the XSLT processor. Experiments performed demonstrated
that both Apache Xalan [7] and Saxon [5] fulfill the requirements. Xerces-J, Xalan’s underlying DOM Implementation, showed problems handling the larger transformations in some scenarios, though, which led us to using the Saxon 6.5.3 XSLT processor.

The following sections provide an overview of the general ideas and extensions we developed for solving the aforementioned problems.

2.1 Infrastructure

In order to organize the content extraction process a set of data types and services was created. Among them are services for the storage, validation, and consolidation of contents as well as for the configuration of the XSLT-templates and supervision of the extraction runs. Figure 1 shows the general transformation process which has been implemented.

![Figure 1. The SCA-Framework Overview](image)

A comprehensive use of plug-in mechanisms allows the extension of almost any component of the system for specific purposes which we may have not considered yet. Two concepts – namely access controlled parameter lists and requests – are described in the next two sections.

2.1.1 Parameter Lists

Parameters are used to extend the native variables of XSLT thus allowing to define the “context” or environment in which a transformation is running and use the same transformation with different set-ups. In addition to setting parameters on the stylesheet, all extensions defined here are capable of handling parameters, as well. Parameters are grouped in sets which are further aggregated to parameter lists which can be processed iteratively by a managing instance. If desired the access to parameter lists can be controlled by the management instances. This ensures that sensitive information such as credentials or personal data can only be used by authorized instances within the infrastructure.

2.1.2 Requests

Requests provide an extensible mechanism for loading additional data or resources from a given location. For each kind of source which can be addressed a specific type of request is used to define the settings which are needed to successfully access the source. Dedicated request handlers for each type can use these settings to execute the request. The result of a successful execution is a MIME-typed input stream which can be used by other extensions or for typed parsing. Currently supported types of requests are HTTP, FTP, OAI-PMH and file system but the list can be easily extended because of incorporation of a plug-in mechanism.

For HTTP both the GET and the POST operation with an arbitrary set of parameters is supported, thus facilitating the simulation of filling and submitting forms, logging in or querying web-databases. Such requests combined with the parameter lists give access to the typically hidden web and allow personalized information retrieval. For FTP both passive and active mode are implemented together with user/password basic authentication. For OAI-PMH all types of request for the protocol version 2.0 are supported. Finally, the file request allows access to a local file system.

Other parts of the infrastructure which deal with access control, management of the artifacts (parameter lists, configurations, projects/users, subscriptions and defined result types) have been defined, but will not be described here because they don’t fit into the focus of this paper.

2.2 Extension Elements

The following sections describe the XSLT-extensions which have been defined and implemented to facilitate the content extraction, validation and storage from different sources.

2.2.1 Open

The open extension is roughly equivalent to the document() function [6] of XSLT in that it loads data from another location into the current context of the transformation. The major difference is the ability to not only load a specific URI but to execute an embedded request with the mechanism described in section 2.1.2. As such it is capable of loading a wider range of contents than the original XSLT function. For instance it is possible to retrieve documents from FTP servers which require authentication, to simulate logins to web pages for access to personal pages, retrieving detail pages for a main page and much more. Because the execution of a request returns a MIME-typed input stream, the extension element uses MIME-type specific parsers to convert the delivered content into XML for the transformation. An additional feature is its capability to define a match to select only a subset of the loaded document for further processing.

Therefore, this extension is the basic component which opens the opportunity to create a template-based spider or crawling mechanism.

![Figure 2. The Open Extension](image)
The about extension is named after the entities which have registered in the corresponding hooks. Again the manager can issue notifications to programs and listen—resulting data is stored.

merging can be performed without corrupting the knowledge, the known facts and possibly merges the information. If the known content or meta-data. If so it tries to consolidate the data. Afterwards the manager checks whether the data refers to management component it is first validated against a data type definition. Currently they include the date, current-uri and hash extension.

The date-extension allows users to parse a given XML-node into a date and format the result for a given locale. Additionally, the user specifies the original pattern and a locale, to facilitate a wide range of situations where dates appear in different forms.

The current-uri-extension returns the currently loaded URI. This is important in cases where the URI changes, either because the Open extension was used or some requested URI led to a redirect.

The hash-extension allows creating a hash value over the result of a transformation. This hash value can be used as an identifier as well a fingerprint of a specific source or target fragment. It therefore supports identification, duplicate detection and validation of retrieved contents.

3. EXPERIMENTS AND EVALUATION

In order to estimate the reliability and performance of the implemented mechanisms we performed a number of experiments. The goal was to extract meta-data about objects from public web sites and compare the automatically extracted data with manually aggregated data. The main criteria were whether we received all available data for our requests and whether each data-item was fully and correctly extracted.

We chose CiteSeer, a search engine for scientific publications and references among them, and Flickr, a community portal for uploading, sharing and commenting on pictures. Both CiteSeer and Flickr provide too many items for manual aggregation so that we decided to focus on a reasonable set of data.

3.1 CiteSeer

For the evaluation of the CiteSeer-results we made use of the CiteSeer OAI collection, a collection compliant with the Open Archives Initiative Protocol for Metadata Harvesting. On CiteSeer OAI-data is available by invoking an OAI-request on http://cs1.ist.psu.edu/cgi-bin/oai.cgi'. We requested 1000 records which had been added to the repository after August 01, 2004 and used these records as a reference to validate our extraction results. Since the output of the OAI-request is available in the Dublin Core Standard format we created test wrappers conforming to the standard and extracted the information found on the page of each publication into this representation. Altogether we gathered and transformed 1000 items. Having the reference data and the extracted data in the same format we could easily compare each item. Table 1 shows the results of the comparison.

The date-extension allows users to parse a given XML-node into a date and format the result for a given locale. Additionally, the user specifies the original pattern and a locale, to facilitate a wide range of situations where dates appear in different forms.

The current-uri-extension returns the currently loaded URI. This is important in cases where the URI changes, either because the Open extension was used or some requested URI led to a redirect.

The hash-extension allows creating a hash value over the result of a transformation. This hash value can be used as an identifier as well a fingerprint of a specific source or target fragment. It therefore supports identification, duplicate detection and validation of retrieved contents.

3. EXPERIMENTS AND EVALUATION

In order to estimate the reliability and performance of the implemented mechanisms we performed a number of experiments. The goal was to extract meta-data about objects from public web sites and compare the automatically extracted data with manually aggregated data. The main criteria were whether we received all available data for our requests and whether each data-item was fully and correctly extracted.

We chose CiteSeer, a search engine for scientific publications and references among them, and Flickr, a community portal for uploading, sharing and commenting on pictures. Both CiteSeer and Flickr provide too many items for manual aggregation so that we decided to focus on a reasonable set of data.

3.1 CiteSeer

For the evaluation of the CiteSeer-results we made use of the CiteSeer OAI collection, a collection compliant with the Open Archives Initiative Protocol for Metadata Harvesting. On CiteSeer OAI-data is available by invoking an OAI-request on http://cs1.ist.psu.edu/cgi-bin/oai.cgi'. We requested 1000 records which had been added to the repository after August 01, 2004 and used these records as a reference to validate our extraction results. Since the output of the OAI-request is available in the Dublin Core Standard format we created test wrappers conforming to the standard and extracted the information found on the page of each publication into this representation. Altogether we gathered and transformed 1000 items. Having the reference data and the extracted data in the same format we could easily compare each item. Table 1 shows the results of the comparison.

The date-extension allows users to parse a given XML-node into a date and format the result for a given locale. Additionally, the user specifies the original pattern and a locale, to facilitate a wide range of situations where dates appear in different forms.

The current-uri-extension returns the currently loaded URI. This is important in cases where the URI changes, either because the Open extension was used or some requested URI led to a redirect.

The hash-extension allows creating a hash value over the result of a transformation. This hash value can be used as an identifier as well a fingerprint of a specific source or target fragment. It therefore supports identification, duplicate detection and validation of retrieved contents.

3. EXPERIMENTS AND EVALUATION

In order to estimate the reliability and performance of the implemented mechanisms we performed a number of experiments. The goal was to extract meta-data about objects from public web sites and compare the automatically extracted data with manually aggregated data. The main criteria were whether we received all available data for our requests and whether each data-item was fully and correctly extracted.

We chose CiteSeer, a search engine for scientific publications and references among them, and Flickr, a community portal for uploading, sharing and commenting on pictures. Both CiteSeer and Flickr provide too many items for manual aggregation so that we decided to focus on a reasonable set of data.

3.1 CiteSeer

For the evaluation of the CiteSeer-results we made use of the CiteSeer OAI collection, a collection compliant with the Open Archives Initiative Protocol for Metadata Harvesting. On CiteSeer OAI-data is available by invoking an OAI-request on http://cs1.ist.psu.edu/cgi-bin/oai.cgi'. We requested 1000 records which had been added to the repository after August 01, 2004 and used these records as a reference to validate our extraction results. Since the output of the OAI-request is available in the Dublin Core Standard format we created test wrappers conforming to the standard and extracted the information found on the page of each publication into this representation. Altogether we gathered and transformed 1000 items. Having the reference data and the extracted data in the same format we could easily compare each item. Table 1 shows the results of the comparison.

Figure 3. The Content Extension

In order to save arbitrary content for later reference or processing such as analysis and indexing the content extension was developed. It takes the result of the contained transformation and treats it as “native” content. As such is it capable of either performing a regular XSLT transformation or executing a single contained request to load data from another location.

The loaded content is handed to a manager instance which decides how to validate and store the content and assigns it an ID. By using the corresponding hooks interested projects or programs can receive notifications about new, changed or deleted contents.

As a result, the extension can be used to easily implement mirroring functionalities.

Figure 4. The About Extension

The about extension is named after the rdf:about attribute which is used to express knowledge about something and can be seen as an extension of the xsl:element instruction. It computes the result of the contained transformation and treats it as meta-data about known or unknown data. The format of the meta-data can be anything just as the typical result of a XSLT transformation, such as analysis and indexing the content extension was developed. It takes the result of the contained transformation and treats it as “native” content. As such is it capable of either performing a regular XSLT transformation or executing a single contained request to load data from another location.

The loaded content is handed to a manager instance which decides how to validate and store the content and assigns it an ID. By using the corresponding hooks interested projects or programs can receive notifications about new, changed or deleted contents.

As a result, the extension can be used to easily implement mirroring functionalities.

2.2.2 Content

<x!-- Category: instruction -->
<dai:content
    final? = { "yes" | "no" }
    id? = { string-expression }
    <dai:about
    use-attribute-sets? = { qnames }
    id? = { string-expression }
    name = qname
    namespace? = { uri-reference }
    <dai:open
</dai:open>

Figure 3. The Content Extension

In order to save arbitrary content for later reference or processing such as analysis and indexing the content extension was developed. It takes the result of the contained transformation and treats it as “native” content. As such is it capable of either performing a regular XSLT transformation or executing a single contained request to load data from another location.

The loaded content is handed to a manager instance which decides how to validate and store the content and assigns it an ID. By using the corresponding hooks interested projects or programs can receive notifications about new, changed or deleted contents.

As a result, the extension can be used to easily implement mirroring functionalities.

2.2.3 About

<x!-- Category: instruction -->
<dai:about
    use-attribute-sets? = { qnames }
    id? = { string-expression }
    name = qname
    namespace? = { uri-reference }
    <dai:open
</dai:about>

Figure 4. The About Extension

The about extension is named after the rdf:about attribute which is used to express knowledge about something and can be seen as an extension of the xsl:element instruction. It computes the result of the contained transformation and treats it as meta-data about known or unknown data. The format of the meta-data can be anything just as the typical result of a XSLT transformation, but XML-Schema based XML or RDF are preferred because the implemented management components are able to process this data in special ways.

When the result of the transformation is handed over to the management component it is first validated against a data type definition. Afterwards the manager checks whether the data refers to known content or meta-data. If so it tries to consolidate the data with the known facts and possibly merges the information. If the merging can be performed without corrupting the knowledge, the resulting data is stored.

Again the manager can issue notifications to programs and listeners which have registered in the corresponding hooks.

2.2.4 Others (Date, Current-uri, Hash)

Other extension elements have been introduced to ease the process of Stylesheet definition. Currently they include the date, current-uri and hash extension.

1 http://citeseer.ist.psu.edu/
2 http://flickr.com/
3 http://citeseer.ist.psu.edu/oai.html
4 http://dublincore.org/
Table 1. Results of the CiteSeer extraction

<table>
<thead>
<tr>
<th>Evaluated Aspect</th>
<th>Result</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested Records</td>
<td>1000</td>
<td>100.0</td>
</tr>
<tr>
<td>Received Records</td>
<td>1000</td>
<td>100.0</td>
</tr>
<tr>
<td>Correct Records</td>
<td>955</td>
<td>95.5</td>
</tr>
<tr>
<td>Incorrect Records</td>
<td>45</td>
<td>4.5</td>
</tr>
</tbody>
</table>

It shows that we were able to retrieve all documents from the OAI-request reference set with the automatic extraction. The main difference between both data sets was the value of the included date. We were unable to clearly figure out why most dates in the OAI derived data set were different from the information on the web site. However, the samples we took in order to verify the problem suggest that the dates from the automatic extraction are identical with the dates on the web page, and often with dates of the publications themselves. The main task left for the manual comparison was different differently annotated data (e.g. the authors had different notations in the OAI-data and on the web page).

3.2 Flickr

The experiments with Flickr consisted of extracting the “day’s 500 most interesting pictures” and comparing the content (description, title, author, tags, dates) and references between items (comments from other users) from the collection. The main interest was to uniquely identify users by their name in order to construct reference graphs between users, both as authors and commenters, and pictures. Even though the manual verification seemed to locate many differences, the main reason for the difference lies in the volatility of the list which may change between the automatic and manual access because of user interaction. Even though 500 documents were found in both lists, only 473 of the automatically extracted pictures could be verified to be among the manually access 500. A second manual run on the same day’s list found 484 matches and the order of the pictures could be verified as different from the first manual run.

Listing 1. Example Flickr Image Descriptor

```xml
<?xml version="1.0" encoding="UTF-8"?>
<image>
  <title>Horse Jump Markers</title>
  <thumb>xxxx</thumb>
  <author>user_88397783</author>
  <description>Found these lined up like soldiers in a horse arena I shot last week.</description>
  <source>xxxx</source>
  <upload-date>13.11.2007 00:00:00</upload-date>
  <shot-date>22.10.2007 00:00:00</shot-date>
  <tags>
    <tag>horse</tag>
    <tag>jumping</tag>
    <tag>wood</tag>
    <tag>colors</tag>
  </tags>
  <comments>
    <comment>
      <by>user_1lc21c13</by>
      Nice shot. From the thumbnail, it looked like a bunch of colored pencils lined up.
    </comment>
    <comment>
      <by>user_fb11c0c8</by>
      Super shot, super colors !
    </comment>
  </comments>
</image>
```

3 Without date comparison

3.3 Conclusion

In our opinion, the results of the experiments prove the stability and reliability of the extraction process. The extraction process was able to perform different look-ups and loaded additional content from related web pages. The results were successfully merged into a complete piece of information and in most cases the automatically collected data was identical with the reference set. The main problems we encountered – only during the validation – were caused by inconsistent or volatile data.

4. RELATED WORK

This chapter introduces three solutions for web content extraction and evaluates them according to the used technologies for data transformation, the output the content extraction produces and the support for wrapper creation and execution. Services for processing the obtained content are examined as well.

4.1 Lixto

The Lixto Suite [1][2] is a commercial system for web data extraction developed by the Lixto Software GmbH. It consists of the Lixto Visual Developer and the Lixto Transformation Server. Together the software components allow navigation and extraction of data from various sources and in a second step the aggregation and delivering of the gained information to other applications and devices.

The main task of the Lixto Visual Developer is to support the editor (the person who is interested in the extracted content) during the creation of so called wrappers that are intended to access semi-structured HTML data and convert it to XML. The tool helps the user defining extraction rules by interactively suggesting relevant parts of a web page to the user. It recognizes patterns from the users example input and constructs semi-automatically rules, which together finally form the entire wrapper. The wrapper then can be applied to a single page or to all sub-pages.

The underlying programming language Lixto uses is Elog. This logic-based data extraction language is derived from datalog and was developed by Lixto in cooperation with the Technical Uni-
versity of Vienna. It works on the tree representation of a HTML document and lets the user specify paths with conditions (and filters) similar to XPath to select parts of the document to extract.

The Lixto Transformation Server contains the generated wrappers and is responsible for its scheduling and execution and for the aggregation of extracted content. It allows transforming extracted XML data from different wrapper programs to a domain-specific format. For this step Lixto uses the standardized XSLT language. Another feature of the Transformation Server is delivering the extracted data to other systems through several channels.

4.2 Piggy Bank

Piggy Bank [4] is a Firefox web browser extension, which enables the users’ browser to extract data from webpages, to aggregate extracted data and store it for future usage. The tool was developed within the scope of the SIMILE project led by the MIT Libraries and MIT CSAIL (MIT Computer Science and Artificial Intelligence Laboratory) and supported by several contributors.

For the web-content extraction Piggy Bank utilizes client-side wrappers, in this context known as screenscrapers. After installing and activating a screenscraper in the Piggy Bank, the screenscrapers gathers relevant information fragments from a webpage and transforms the data into the semantic web format RDF. The execution of a screenscraper for a particular webpage is triggered by the user by clicking the Piggy Bank symbol in the browser while visiting the page.

The creation of screenscrapers is the task of an editor by writing JavaScript code to retrieve the desired parts of a webpage. To support screenscraper editors another Firefox extension is available. The Solvent open-source tool allows defining XPath expressions in a visual way. With the interaction of the user the tool generates XPath expressions that afterwards are translated to JavaScript. In addition Solvent offers the publication of screenscrapers to share them with other users.

Besides the explicit extraction of data by invoking JavaScript screenscrapers on Document Object Models (DOM), Piggy Bank also evaluates available XSL Transformations and retrieves attached RDF-data if a website publisher already provides its contents in this semantic representation.

After the web-content extraction the Piggy Bank user has the possibility to organize the collected data, to save and to browse it or to use several predefined services to view it. For storage the user can select between a local repository and a remote repository where the extracted information can be shared within a registered group.

4.3 GRDDL

Unlike the two solutions introduced in the sections 4.1 and 4.2 above GRDDL [3] is not a web-content extraction system, where it is the responsibility of the user to develop wrappers to extract data from external sources but is a mechanism concerning the content publishers. GRDDL stands for ‘Gleaning Resource Descriptions from Dialects of Languages’ and is currently a Proposed Recommendation of the World Wide Web Consortium (W3C).

GRDDL defines a markup that allows the content provider declaring its documents containing RDF-conforming data and attaching transformation algorithms to its documents. These transformations are either directly linked to a document or associated to a document by its namespace. Although GRDDL transformations are typically XSL Transformations potentially each programming language can be used for writing GRDDL transformations.

Since GRDDL only makes available a markup there are no tools to create any transformation algorithms provided by GRDDL. It is in the responsibility of the author to select adequate tools for the generation of such wrappers.

The extraction of RDF-data must be triggered by the user or by GRDDL-aware agents that also are responsible for the processing of the retrieved contents.

4.4 Summary

After introducing three different approaches in the preceding sections this section confronts these approaches and shows how they relate to the strategy of the SCA Framework.

One difference between all solutions is the kind of sources that are supported for data extraction and the output the wrappers deliver. While Piggy Bank and GRDDL restrict the user, Lixto and the SCA framework are more flexible approaches that allow the extraction of data from various kinds of sources. In addition their wrappers are able to transform harvested data to any desired format. Concerning the used transformation language the SCA framework uses the standardized XSLT language and exploits all its possibilities. Furthermore the SCA framework extends XSLT to provide even more functionality and permits in this way the whole transformation process described by only one wrapper. By following links and executing different types of requests it also supports the extraction of whole websites instead of fetching only single pages. The technologies for data-accessing and -harvesting and the wrapper output formats are summed up in Table 2.

<table>
<thead>
<tr>
<th>Framework/Mechanism</th>
<th>Supported Access</th>
<th>Technology</th>
<th>Wrapper-Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAF</td>
<td>Websites, File system, DB, Services</td>
<td>XSLT with extensions</td>
<td>Any</td>
</tr>
<tr>
<td>Lixto</td>
<td>Websites, DB, Services</td>
<td>Elog, XSLT</td>
<td>XML, Any</td>
</tr>
<tr>
<td>Piggy Bank</td>
<td>Webpages</td>
<td>JavaScript, XSLT, RDF/XML, RSS</td>
<td>RDF</td>
</tr>
<tr>
<td>GRDDL</td>
<td>XML</td>
<td>Typically XSLT</td>
<td>RDF</td>
</tr>
</tbody>
</table>

Concerning the wrapper creation Table 3 gives an overview. It shows which of the examined approaches provides tool support, where the created wrappers are stored and who is responsible for the wrapper creation. Due to the fact that GRDDL provides just a mechanism to assign RDF-containing documents all responsibilities are of the content provider without any tool support. So only the Lixto, Piggy Bank and SCAF solutions offer tools for wrapper creation and merely Piggy Bank and SCAF allow the sharing of wrappers with the aid of remote repositories.
Table 3. Wrapper Creation

<table>
<thead>
<tr>
<th>Framework/Mechanism</th>
<th>Tools</th>
<th>Location of Wrappers</th>
<th>Wrapper Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAF</td>
<td>Yes</td>
<td>Local + Repository</td>
<td>Editor</td>
</tr>
<tr>
<td>Lixto</td>
<td>Yes</td>
<td>Local</td>
<td>Editor</td>
</tr>
<tr>
<td>Piggy Bank</td>
<td>Yes</td>
<td>Local + Sem. repository</td>
<td>Editor, Content Creator</td>
</tr>
<tr>
<td>GRDDL</td>
<td>No</td>
<td>Comes with content</td>
<td>Content Creator</td>
</tr>
</tbody>
</table>

The web-content extraction can either be triggered on demand or via a scheduling mechanism. Both possibilities are realized within the SCA framework. Since we could not find any information whether the Lixto system allows data extraction on demand we assume that it offers none but the scheduling mechanism that is often described. Both the Piggy Bank and GRDDL require executing the web-content extraction through a user or through a robot, respectively.

After the extraction of relevant information the approaches handle the received content in different ways and offer different services for their usage. Again the Piggy Bank and GRDDL dedicate the content handling to the extracting user or robot, whereby Piggy Bank makes available some predefined services. The Lixto Suite and the SCAF provide similar services for content processing. Both solutions consolidate incoming data and handle the delivering of content. The SCAF allows projects and programs to subscribe for content and notifies the subscribers when new or duplicate content arrives or when content was deleted from the repository. A validation component makes sure, that the content repository always stays in a consistent state. It checks incoming content according to a type definition.

Table 4 lists all available content services and the possibilities of extraction triggering provided by the introduced web-content extraction solutions.

Table 4. Extraction execution and Content Services

<table>
<thead>
<tr>
<th>Framework/Mechanism</th>
<th>Extraction-Trigger</th>
<th>Content Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAF</td>
<td>Schedule / Service</td>
<td>Validation, Consolidation, Storage, Notification</td>
</tr>
<tr>
<td>Lixto</td>
<td>Schedule</td>
<td>Consolidation, Delivering</td>
</tr>
<tr>
<td>Piggy Bank</td>
<td>User</td>
<td>Responsibility of the user (Tagging, Sharing, predefined services)</td>
</tr>
<tr>
<td>GRDDL</td>
<td>Robot / User</td>
<td>Responsibility of the user/robot</td>
</tr>
</tbody>
</table>

5. OUTLOOK

Although the SCAF is a powerful framework for content extraction, further enhancements can be envisioned which will contribute to the systems robustness and ease of use.

The current tool for developing wrappers still requires a lot of knowledge and insight into the transformation process from the editors. Further research on supporting the creation process through a deeper visual and structural analysis could greatly enhance the usability of the tool. Additionally, this analysis can assist the runtime transformation process in providing more robust ways of matching the available content to the desired result element and detecting changes in the structure which can not be automatically compensated. At the moment the main work consists of extending the capabilities of the tool for wrapper creation and of the management component that controls and monitors the extraction process.

Another improvement can be achieved by enhancing the underlying content validation and consolidation process. The currently used set of generic validity tests can be updated to perform validation and reasoning against semantic content type definitions such as RDF/OWL. However, traditional reasoning assumes an “open world” which does not allow us to detect invalid data in order to fix the broken parts. We are currently working on extending the existing reasoning mechanisms to enforce the closed-world- assumption for our own data types, which is essential for the validation, as well as identifying and merging partially identical data, which would allow us to successively update our contents with newly found elements as long as they don’t violate the consistency.

In addition to extending the functionality of the transformation and completing the infrastructure, further experiments with larger data sets and additional web pages will be performed and have to show that the system is able to provide consistent bi-directionally associated data from semi-structured content sources.

6. REFERENCES


