Visual Programming of Web Data Aggregation Applications

Robert Baumgartner
Institut für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
baumgart@dbai.tuwien.ac.at

Georg Gottlob
Institut für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
gottlob@dbai.tuwien.ac.at

Marcus Herzog
Institut für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
herzog@dbai.tuwien.ac.at

Abstract

Most of the information needs today can be satisfied by searching and browsing the Web. However, repetitive tasks such as monitoring information on Web sites should be done automatically on behalf of the user. Likewise, important events such as new information available on a monitored Web site should be actively reported to the user.

These requirements can be met by building Web aggregation applications that distill required data from Web pages. Building such applications from scratch is a cumbersome and error prone task. Application programmers need to program functionalities such as Web navigation and page retrieval, content extraction, and result aggregation.

In this paper we will present a fully visual development environment for Web data aggregation applications, the Lixto Transformation Server (TS). We will also demonstrate how this environment can be used to build services for personalized information channels. Users can subscribe such services to receive information on various types of client devices.

1 Introduction

Today the Web is the largest distributed data vault ever to be built by human mankind. The information that can be extracted from this huge data space is of enormous variety and of high potential benefit for all kinds of interests and information needs. Despite these promises extracting information can become a nightmare due to the unstructured and heterogeneous nature of the data formats employed. The majority of all information sources on the Web utilize HTML to transport the information from the information provider to the information consumer. Although this is a convenient method for human consumers it is hard to process by non-human entities. This is especially true for tasks where different information sources have to be integrated to fulfill the specific information needs.

In spite of the growing number of native XML data sources it is our assumption that the Web will stay a heterogeneous environment. The demand for personalized information across the boundaries of a single information provider will remain an issue. The focus of the Lixto TS system is on converting Web-based data sources into personalized information channels. While search engines [7] focus on locating potential information sources the TS system's main purpose is to process preselected information sources. The system extracts and integrates the information from various sites, transforms the information according to the user's needs, and delivers it to the user exploiting the delivery channel of choice (e.g., by e-mail, HTML page, or mobile phone). The whole process is configured by the user by means of an interactive Web-based configuration user interface. After successful configuration, the system monitors the Web on behalf of the user and conveys the extracted information to the client device.

In the remainder of the paper we will first give an overview of the methodology used in processing Web data. We will continue with a demonstration of the applicability of the system by presenting an example application. We conclude by discussing some related work and giving an outlook to future research directions.

2 Methodology

Automatic processing of Web-based information is a complex task. In contrast to conventional data sources data on the Web is at the best semi-structured, potentially unbound, and highly volatile. Techniques developed in the areas of, e.g., database and information retrieval can only cover a limited set of information processing needs that are intrinsic to this area [1].

To structure the process of building information aggregation applications based on Web data we have broken down the overall task into the following sub-tasks: acquiring the required data from the source locations, extracting the relevant data from various document formats, integrating, transforming, and delivering the data to client applications. A more detailed analysis of the above described processing stages can be found in [10].

2.1 Visual Programming and Data Flow

Given the task decomposition as described above we will present a visual programming language to program applications using these functional building blocks. In our approach processing stages are mapped onto processing nodes. Nodes are represented as icons which symbolize the function that the

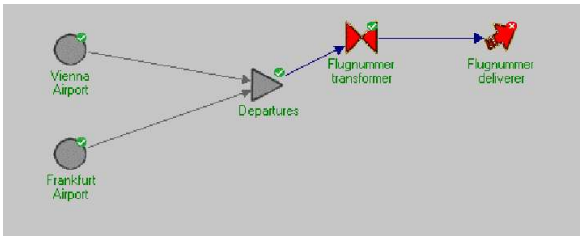

Figure 1: Visual representation of an infopipe example

specific node performs. Currently we support four different types of processing components: Source component, Integrator component, Transformer component, and Deliverer component. The data flow between these components is represented in the visual language by directed edges of a network graph. The information processing network can be visually defined by adding and connecting nodes as appropriate. We call such a processing network an *infopipe*. An example infopipe is depicted in Figure 1.

The actual data flow within the infopipes is realized by handing over XML documents. Each stage within the infopipe accepts XML documents as input, performs its specific task, and produces an XML document as result. This result is fed to the successor components which in turn will perform the next information processing stages. Components which are not on the boundaries of the network are only activated by their neighboring components. Boundary components (i.e., source and sink components) have the ability to activate themselves according to a user specified strategy and trigger the information processing on behalf of the user.

Once activated the information flow within the infopipe is request driven. In our architecture the activation is spreading from the sinks towards the sources. The processing takes place through backward chaining to all immediate predecessor components which in turn activate their predecessor components. A caching strategy ensures to process only those nodes where input ports have changed during consecutive calls.

2.2 Service Abstraction

An infopipe service is a fully configured Web aggregation application. Although configuring such a service is easier and less time consuming than programming an equivalent application from scratch it might be too complicated for a lay user to perform such a task. Therefore we introduced another abstraction layer: we differentiate between the role of a *service administrator* and a *service user*.

By means of this mechanism service administrators can publish pre-configured information services. These services can be subscribed by other users. To allow these users to personalize the infopipes, service administrators can declare components as so-called *user components* (see Figure 2). In the user component the values of certain parameters can be overwritten by the user. In addittion, it is possible to add additional components to such a subscribed service. Thus, experienced users are able to personalize the service by extending the structure of the information processing network.

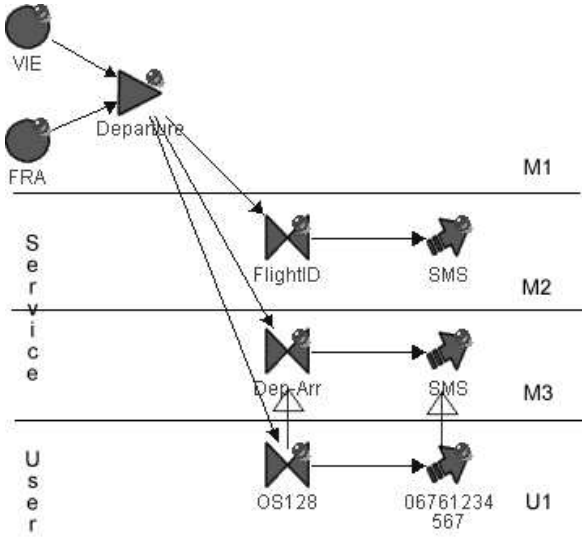

Figure 2: The Service Abstraction

Service users are restricted to a limited configuration interface for pre-configured service components, the so-called user view. In contrast, administrators of a service see the full-fledged administrator view.

End consumers have access to the directory of services on the TS and personalize subscribed services by submitting values for parameters. In the following sample application, the Transformer component contains such a user-definable parameter. In the end consumer view the user can specify which flight number value or destination/departure airport value is interesting for him. Additionally, the end consumer can specify how often the source web pages are queried, and to which cell-phone number or email-address the data is delivered.

3 Flight Notification Scenario

In the following we will demonstrate the capabilities of the TS system by going through a demonstration scenario. The application domain is flight notification, where users can subscribe to specific flights and receive notifications about the current state of the flight. Consider the following situation: A business traveler needs to attend a meeting in Frankfurt. From her travel agent she receives a ticket issued for the Austrian Airlines morning flight OS121 to Frankfurt and coming back to Vienna in the evening on flight number OS128. Because the meeting is very important she wants to be informed if the flight schedule changes due to some unexpected event. By subscribing to an appropriate information pipe that has been set up by a service provider the user can easily register for that service. The system provides personalization interfaces for the user either through Web access (e.g., filling in some form) as well as through wireless devices utilizing either WML pages or a SMS (short message service) interface. While the Web-based configuration interface is convenient for office use, the configuration interfaces on wireless devices are essential for out of office use, i.e. if the user wants to update the configuration while she is already on the busi-

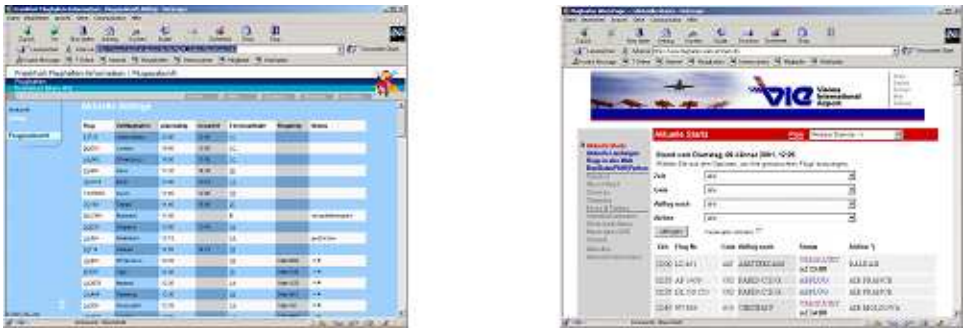

Figure 3: Frankfurt (left) and Vienna (right) Airport sites

ness trip.

3.1 Configuration of Source Components

We will start our demonstration tour by setting up an infopipe that integrates data from two airport information sources: Vienna airport and Frankfurt/Main airport. These two airports provide current flight information on their Web sites. The infopipe administrator will generate a new information pipe in his workspace. He will then generate two new Source components that will monitor the two airport sites. The configuration dialog of the Source component starts with providing a starting URL that will be the starting point of the navigation process. Typically this URL is the homepage of the corresponding service that we will monitor. In the case of the Frankfurt airport this will be http://www.frankfurt-airport.de/ (see Figure 3). To configure this Source component the infopipe administrator just follows the usual navigation path that leads to the desired page that contains the needed information. During navigation he can utilize any kind of links such as hyperlinks, images, image maps, or form submit buttons. The system will observe the navigation path and store the request sequence along with parameter information for later replay.

Using the Lixto Visual Wrapper application interface, an extraction program can be defined by visually marking the interesting parts of the page that are relevant for the application domain. Defining an extraction program exploits the structure of the HTML page. More information on the building process for extraction programs in the Lixto methodology can be taken from [5] and [6].

During navigation, the infopipe administrator can at any time associate extractor programs with the current page. For that purpose the system inserts a small command interface into the body of each page or frame. Through this interface the administrator can either associate a content extraction component or a link extraction component to this Source node. During run-time these extraction components will extract the needed information from the HTML source pages and generate so-called XML companion pages. Once an extraction program is associated with a Source component, that Source component is considered to be valid (i.e., there will be an XML output that can be retrieved from that Source component). Figure 3 shows the original HTML pages of both the Frankfurt and Vienna airport Web site. For the extracted XML pages that reflect the current status of these HTML pages see Figure 4 for the Frankfurt airport and Figure 5 for the Vienna airport example.

```
<document>
<FRA>
   <Flug>
      <Flugnummer>LH 3870</Flugnummer>
      <Zielort>Malta</Zielort>
      <Planzeit>09:35</Planzeit>
      <Terminal>1A</Terminal>
      <Flugsteig>Gate B27</Flugsteig>
      <Status>boarding</Status>
   </Flug>
   <Flug>
      ....
```

Figure 4: XML fragment Frankfurt Airport

```
<document>
<VIE>
   <flight>
      <time>16:50</time>
      <status>ABFLUG</status>
      <startingTime>17:22</startingTime>
      <destination>BRUSSELS</destination>
      <airline>SABENA</airline>
      <flightnumber>SN2906</flightnumber>
   </flight>
   <flight>
      ....
```

Figure 5: XML fragment Vienna Airport

3.2 Configuration of Integrator Component

Once the information sources are configured, an Integrator component can be added to the infopipe that integrates the output of the airport sources. If we look closely at the output that is generated by the Source component we will see that both the structure (i.e., the schema) as well as the content of the XML output is obviously related. Still the naming of elements in the two schemata differs, e.g. the elements `<flightnumber>` and `<Flugnummer>` both refer to the

Figure 6: The Integrator dialog

```
<document>
  <flight departure="VIE">
   <time>16:50</time>
   <status>boarding</status>
   <startingTime>17:22</startingTime>
   <destination>BRUSSELS</destination>
   <airline>SABENA</airline>
   <flightnumber>SN2906</flightnumber>
   <gate />
  </flight>
    .....
  <flight departure="FRA">
   <time>09:35</time>
   <status>boarding</status>
   <startingTime />
   <destination>Malta</destination>
    <airline />
   <flightnumber>LH4202</flightnumber>
   <gate>B 27</gate>
  </flight>
      ....
```

Figure 7: XML fragment output Integrator

flightnumber of that flight. Likewise the content of some elements might refer to the same concept, yet utilizing different terms to express that concept, e.g. *boarding* and *ABFLUG* in the `<status>` elements. These are examples for mappings that can easily be inferred by humans but have to be stated explicitly for automatic information processing.

The Integrator allows for mapping element names, element attributes and content of elements between input and output XML structures by means of a visual dialog (see Figure 6). When the Integrator dialog is called, the Integrator component requests input documents from the components that are connected to this Integrator component on the input ports. In this example scenario these components are the two Source components that we have already configured in the previous step. When the Integrator will be configured for the first time, a dialog asks for the definition of the appropriate output structure. The output structure can be either defined from scratch or one of the input structures can be used as template. The input structures are determined by analyzing the XML documents that are handed over on the input ports. The configuration dialog already utilizes live data that are gathered from the Web sources.

The mapping dialog allows for defining mappings between elements and attributes of the input document and the output document. The mapping process will be performed by an XSLT stylesheet that is generated according to the mapping rules. In the configuration dialog the structure of the input document is visually represented by a nested table while the structural elements of the output document are shown in XPath notation. By selecting the appropriate element or attribute in the selection list, a mapping between input and output is specified. In an advanced settings dialog, default values for elements or attributes can be specified. To normalize the content of XML elements regular expressions can be defined that are executed after the mapping process.

The output of the Integrator component is an XML document that contains data in a normalized view. An example of the integrated XML structure containing two example data instances from Vienna and Frankfurt sources is given in Figure 7. Note that the `<Status>` element just contains normalized values. The content of the `<Flightnumber>` element has been normalized to omit whitespaces in flightnumbers. This is important for the selection process that takes place in the following component.

3.3 Configuration of Transformer Component

In this example we use two Transformer components to configure two queries that allow the subscribers of this infopipe to select the desired flight information. The first Transformer will select flights according to the value of the flightnumber, whereas the second will select flight information according to departure and destination values. The infopipe administrator expresses these queries by defining which elements of the output structure will be bound to variables. For these variables, a comparator function can be specified, along with default values and the specification whether a value for this variable can be given by the subscriber. If user input for this variable is allowed, corresponding input masks will be generated by the system. Again, the result of the configuration is an XSLT stylesheet that is automatically generated.

3.4 Configuration of Deliverer Component

Because the flight notification service is predominately interesting for travelers, information delivery will be targeted for mobile devices. The most popular service for this kind of messages is SMS, which allows to transmit short text-based messages to GSM mobile phones. The purpose of the Deliverer component configuration dialog is to allow the service administrator to specify the formatting of the XML result according to the chosen delivery platform. In case of SMS, a composer editor can be used to build messages out of data elements and fixed text constants. The administrator can select

Figure 8: InfoPipe personalization

appropriate elements or attributes from the XML document schema to include it in the output message.

3.5 Use of Service

Once the configuration process is finished by the administrator, this infopipe is added to the directory of available services. Users can now subscribe to this service. The subscription can take place either by SMS, email, WML or HTML interface. Either way, a new infopipe is derived and added to the userspace of the subscriber. The mechanism is the same for all interaction modalities, only the interaction interface differs. For the HTML interface, the user simply clicks on the desired information pipe. In case of the SMS interface, each information pipe is bound to a specific service number (i.e., a telephone number). In this example, the user can actually choose between two information pipes, one allowing to query the flight information by flightnumber and the other by querying the flight information by submitting destination and departure airports.

The user has the ability to personalize an infopipe by filling in parameters that are defined in the user components. In this example, only two of the components are user components: the Transformer and the Deliverer. The Transformer either excepts a flightnumber or a destination/departure airport, according to the chosen service. The Deliverer user component allows the user to schedule the delivery of the service (either event driven, i.e. on the availability of data, or at a certain time and date). She can also choose whether data should only be delivered if there is a content change between subsequent messages, or if all messages should be delivered.

Figure 8 shows the personalization dialog for the flight information notification service. In case of the SMS dialog, the user can issue a simple command string such as `ADD LH4608` which would be interpreted as adding a new infopipe and setting the user parameter flightnumber to LH4608.

4 Related Work

A lot of work has been done in the field of data extraction from the Web. Projects such as W4F [15] and XWrap [11] provide visual tools for supervised interactive wrapper generation. However, the Lixto TS goes beyond the scope of mere Web data extraction tools. In fact data extraction is merely one — although a very important — step in the processing model for which we employ the Lixto extraction engine. Lixto Visual Wrapper allows for very expressive extraction programs [8], that can be generated by means of a visual builder interface. Therefore it integrates very nicely with the overall visual programming approach.

Related work has been carried out in the field of automating the retrieval of Web pages and Web data. While crawler technology is directed towards fetching and possibly indexing as much pages as possible, it first concentrated on “easy”-to-reach pages, i.e. by simply following hyperlinks. Only more recently the focus has been laid on pages that are hidden behind forms or other means of interactive communication with the user that prevent simple crawlers form accessing these pages. The HiWE crawler [14] for instance is specifically targeted at interacting with form-based search interfaces. This capability is of great use when information has to be extracted from the so-called deep Web [4]. In the context of our system this is very likely when input to infopipes stem from private sources, e.g. bank accounts or shopping baskets.

Automatic Web navigation has been the focus of the WebVCR project [2]. As the name implies, users are able to record and play back a sequence of browsing steps applying this tool. The intention of this tool has been to provide shortcuts to hard-to-reach pages, hence the name *smart bookmark* for the result of the recording procedure. WebVCR has been implemented as a client-based tool, using a Java Applet and JavaScript to track the user interaction. In the WebViews architecture [9] smart bookmarks have been utilized as building block for customized views of Web content and services.

Besides WebViews and our own approach, also the ANDES framework [13] deals with the issue of merging crawler technology with XML-based data extraction technology. The ANDES methodology defines five steps of extracting structured data from Web sites: navigation, extraction, structure synthesis, data mapping, and data merging. The extraction and transformation operations are achieved by utilizing XSLT. The `HTML` output of the crawler has to be convert to `XHTML` before XSLT can be applied. XSLT files are applied according to the URL of the original HTML document. The order of processing is defined by a fixed pipeline according to the steps of the methodology. However it remains unclear how the user of the ANDES framework can express the intended flow of information. Furthermore, a lot of domain-specific knowledge has to be encapsulated in the XSLT code.

Related work in the area of XML-based data integration are Cupid [12] and Momis [3]. One goal of the Momis mediator system is the the task of schema integration. Cupid operates both on XML and relational schemata, and is based on XSTL. Cupid builds on the BizTalk Client tools to visually define a transformation. The BizTalk client tools are restricted to 1:1 and 1:n mappings. Both systems, Cupid and Momis also

feature matchings based on ontologies. The unique feature of the Lixto Suite is that the whole process comprising data extraction, data integration, data delivery and personalisation can be configured in a fully visual way within the framework. There is ongoing work on coupling the Lixto Suite with the MOMIS mediator system.

5 Conclusion

In this paper we have first presented our methodology for building Web aggregation applications through visual programming. We have identified the main building blocks for such applications: source navigation, data extraction, data integration, data transformation, and data delivery. We have also defined a data flow model that integrates well with the visual programming paradigm. Furthermore we have shown how such visual programs can be used to provide users with services that deliver personalized information on multiple platforms.

We have presented the Lixto TS system that implements the model as a rapid application development environment. Using the system service administrators can easily configure data aggregation applications. Service users can subscribe to these applications and interact with the services by means of multi-modal interfaces.

For future work we plan to address the following topics: It turned out that error detection and handling of exceptional states is vital for maintaining configured infopipes. For instance, it is important to detect whether data is missing because a network connection is temporarily unavailable or an extraction program is broken or the configuration of a component has become invalid. Therefore we will annotate the processing stages with meta information that is generated by the system. This information can be utilized to detect abnormal processing states and action can be taken appropriately.

Another issues is the transparent integration in publishing platforms such as Web portals. While the execution of content formating within the Lixto TS framework is feasible for dedicated devices such as mobile phones or PDAs, it is less useful if other formating infrastructures can be re-used. In this case, information should be exchanged as pure XML. However, a meta annotation wil be necessary to allow an easy integration of the content into other frameworks. Likewise, offering the information extraction application as Web service is an interesting option that we are going to pursue over the next time.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] Vinod Anupam, Yuri Breitbart, Juliana Freire, and Bharat Kumar. Automating web navigation with the WebVCR. *Computer Networks and ISDN Systems*, 33(1–6):503–517, 2000.

[3] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[4] Michael K. Bergman. The deep web: Surfacing hidden value. Technical report, BrightPlanet.com LLC, 2000.

[5] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Proc. of VLDB*, 2001.

[6] R. Baumgartner, S. Flesca, and G. Gottlob. Declarative information extraction, web crawling and recursive wrapping with Lixto. In *Proc. of LPNMR*, 2001.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, April 1998.

[8] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web Information Extraction. In *Proc. of PODS – Best paper award*, 2002.

[9] Juliana Freire, Bharat Kumar, and Daniel F. Lieuwen. WebViews: Accessing personalized web content and services. In *Proc. of the Tenth International World Wide Web Conference, WWW10*, pages 576–586, Hong Kong, China, May 2001.

[10] Marcus Herzog and Georg Gottlob. Infopipes: A Flexible Framework for M-Commerce Applications. In F. Casati, D. Georgakopoulos, and M.-C. Shan, editors, *Technologies for E-Services, TES 2001*, pages 175 – 186, Rome, Italy, September 2001. Springer (LNCS 2193). In conjunction with VLDB' 01.

[11] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *International Conference on Data Engineering ICDE*, pages 611–621, 2000.

[12] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *The VLDB Journal*, pages 49–58, 2001.

[13] Jussi Myllymaki. Effective web data extraction with standard XML technologies. In *Proc. of the Tenth International World Wide Web Conference, WWW10*, pages 689–696, Hong Kong, China, May 2001.

[14] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proc. of 27th International Conference on Very Large Data Bases (VLDB)*, pages 129–138, Rome, Italy, September 2001.

[15] Arnaud Sahuguet and Fabien Azavant. Building lightweight wrappers for legacy web data-sources using W4F. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, pages 738–741, Edinburgh, UK, 1999. Morgan Kaufman.