A Workflow-based Approach for Creating Complex Web Wrappers

Paula Montoto, Alberto Pan, Juan Raposo, José Losada,
Fernando Bellas, Javier López

Department of Information and Communication Technologies.
University of A Coruña, Spain
{pmontoto,apan,jrs,jlosada,fbellas,jmato}@udc.es

Abstract. In order to let software programs access and use the information and services provided by web sources, wrapper programs must be built to provide a "machine-readable" view over them. Although research literature on web wrappers is vast, the problem of how to specify the internal logic of complex wrappers in a graphical and simple way remains mainly ignored. In this paper, we propose a new language for addressing this task. Our approach leverages on the existing work on intelligent web data extraction and automatic web navigation as building blocks, and uses a workflow-based approach to specify the wrapper control logic. The features included in the language have been decided from the results of a study of a wide range of real web automation applications from different business areas. In this paper, we also present the most salient results of the study.

Keywords: web wrappers, data mining, web automation, web information systems

1 Introduction

Most of today's Web sources are designed to be easily used by humans, but they do not offer suitable interfaces to allow software programs to interact with them. During the last years, a growing interest has arisen in automating the interactions with web sites. Most previous research works in this field have focused on the concept of wrapper. A wrapper abstracts the complexities involved in automating a certain task on a web source, providing a programmatic interface to external applications. As related work, see [3],[4],[5],[9],[10],[11],[12],[13],[15],[16]. [8] provides a survey.

Most previous works on web wrappers assume a particular underlying model we will call 'query wrapper' model. Query wrappers consider a web source as a special kind of database where queries can be posed using a form, obtaining a result set composed of structured data records. The query wrapper model assumes a pre-defined list of execution steps: first, a navigation sequence is used to automatically fill in some query form. Secondly, data extraction techniques are used to obtain the list of results from the response HTML pages. In addition, query wrappers may also support paginated result listings and accessing 'record detail' pages to extract further information.

While query wrappers are useful, they do not fit some important web automation

applications. For instance, many tasks involve taking decisions in function of the retrieved data to continue the navigation in a way or another.

The objective of this paper is to propose a graphical language for creating wrappers able to automate any web automation process on a given website. It is important to note that, in our view, a wrapper automates the interaction with a single website and for a single purpose. For tasks involving the combination and/or orchestration of several web sources, our approach consists in enabling the wrappers to participate as basic components in usual data and process integration architectures such as data mediators [14] or Business Process Management systems. Other key objective for our language is being simple: wrappers should be created graphically and the language should not include features that introduce unnecessary complexity. Programming-skills should not be necessary in the majority of cases.

As a source of inspiration for our model, we have studied BPM orchestration patterns [1] and technologies (such as BPMN [7] and WS-BPEL [6]). As in our case, BPM technologies are also concerned with graphically specifying complex logic.

To base our proposal on firm roots, we have studied a wide range of real web automation tasks, which are being used by corporations from different business areas, trying to capture all the requirements needed in real applications. Some of the main results of the study are also reported in the paper. The language has been implemented in a fully functional prototype.

The paper is structured as follows. Section 2 reports some of the most important results and conclusions of the motivating study. Section 3 describes the proposed language. Section 4 discusses related work.

2 Motivation

To guide the development of the language, we have studied a wide range of real web automation applications. We have studied 391 wrappers belonging to 24 applications. These applications are a sub-set of those developed by a European enterprise specialized in web automation applications during the last three years. We have chosen applications in different business areas to increase the generality of our approach: B2B web automation (i.e. automating repetitive operations with other organizations through a web interface), batch data extraction, internet metasearch applications, web account integration, and technology and business watch (i.e. monitoring web information relevant for business and/or research purposes, such as competitor prices).

Previously to the study, we analyzed existing workflow technologies for BPM (such as [1],[6],[7]) to identify a features which could apply to web automation applications. Then we analyzed the wrappers in the study to check if they required or could benefit from them. The features we found useful are: conditional bifurcations, error management, parallelism, asynchronous events and sub-processes. We also analyzed if the wrappers conformed to the query wrapper model. Now, we report and discuss some of the main results of the study:

1. A first conclusion is that only the 57% of the wrappers conform to the *query wrapper* model. The percentage varies in function of the application type: 100% of the wrappers in metasearch applications conform to that model, while in B2B applica-

tions the percentage only reaches 53%. The study allows concluding that the query wrapper model is too simple for many real web automation applications.

2. Regarding *conditional bifurcations*, 54% of the wrappers that do not conform to the query wrapper model require them. Therefore, the language should support them.
3. Regarding *error management*, most of the wrappers considered require or could benefit from: 1) On the apparition of an error in the process, indicating which action to perform: either ignore it or halt the process and return the error to the invoking application, 2) Executing retries (e.g. when executing web navigation sequences). In addition, 37% of the wrappers in the B2B application area require or could benefit from user-defined, application-specific exceptions to return to the calling application. Therefore, the language includes support for these features.
4. Regarding *parallelism*, we observed it is very useful in two cases: 1) When a wrapper needs to process a list of records extracted from a web page, and the processing of each record involves executing one or more navigation sequences (e.g. accessing a detail page). Since navigation sequences can be relatively slow, processing the records in parallel can greatly improve performance, and 2) We have also observed that some applications execute the same query wrapper multiple times using different query parameters and then merge the obtained results. Therefore, it is useful to include in the language specific support to allow specifying the parallel execution of multiple queries on the same web form, thus alleviating the invoking application of this task. 84% of the studied wrappers could benefit from either one or both of these kinds of parallelism. On the other hand, no wrapper required other types of parallelism. Recall that, in our model, wrappers abstract the interactions with a single source for a given task. More room for parallelism would undoubtedly arise if we considered web automation tasks involving the combination and/or orchestration of several sources. Nevertheless, we follow the common approach in integration architectures of separating access and coordination layers. To coordinate and/or integrate several sources (which may or may not be web sources), our approach consists in enabling the wrappers to participate as components in usual data and process integration architectures such as data mediators [14] or Business Process Management systems. Therefore, we conclude the language should not include more general support for parallelism because it would considerably increase the complexity and its benefits would be unclear.
5. Regarding *asynchronous operations*, some web pages may dynamically change its content without requiring a full reload of the page (e.g. sources which use Javascript / AJAX technology to update its content). In these situations, it would be useful if the wrapper could be asynchronously notified of changes in the content of the areas of interest in the page. 6 wrappers in the study have to deal with these sources but, since AJAX sources are gaining prominence, we expect this feature to increase its importance. The wrappers we examined are not notified asynchronously. Instead, they access the target regions at specified intervals (i.e. “polling”). The reason is that most automatic navigation systems use browsers as basic components for navigating and hosting HTML pages, and with current browser APIs it is difficult to identify content-change events at the desired granularity. Polling can be easily supported by including a *wait* activity in the language.

6. Regarding *subprocesses*, we have detected that the most complex wrappers studied could be greatly simplified by using sub-processes. We have also observed there are several *structural patterns* that occur in a great number of wrappers (e.g. most wrappers have to deal with different types of results pagination, many wrappers need to poll a list of results for changes, etc.). These patterns appear many times, sometimes with slight variations. We conclude a desirable feature for the language is to allow creating reusable components to support them.

3 Language Description

In this section, we describe the proposed language. Section 3.1 describes the overall structure of a workflow in our approach. The pre-defined activities that can be used in the workflows are described in section 3.2. Section 3.3 describes how users can create reusable components. Section 3.4 presents an example.

3.1 Workflow Model

The data instances handled in the process flow (we will call them values) belong to a structured type [2]. A structured type can be atomic, a record type or a list type. The language specifies support for the atomic types commonly found in programming languages (*string*, *int*, *long*, *double*, *float*, *date*, *boolean*, *binary*, *url*, *money*) and for a specific type called *page*, which encapsulates the information needed to allow the web automation system to access a page: that is, an URL and the required cookies .

Figure 1 shows an UML diagram describing the basic structure of the language. A workflow receives a set of variables as input parameters and returns a single variable as output. The value of a variable is an instance of a valid data type. The input parameters can be mandatory or optional. A workflow is composed of a set of ordered *activities*. Activities can be either basic or structured. Structured activities include those used for loops and bifurcations and enclose one ordered sequence of activities (bifurcations enclose one sequence for each execution path). A workflow can be seen simply as a sub-class of structured activity. Basic activities perform the actions in the workflow. Although not shown in the diagram due to space constraints, certain activities require some of the variables they use to be of a certain data type.

To handle error management, the language leverages on the concept of *exceptions*, that can be either pre-defined or user-defined. There exist pre-defined exceptions to represent generic runtime errors, typical errors produced while executing a navigation sequence (http error, timeout, connection error) and while extracting data records (e.g. a record does not match the expected type). User-defined exceptions are generated using the *THROW* activity. Each exception has assigned one in a set of pre-defined handlers. Handlers exist for throwing and ignoring the exception. It is also supported to configure retries before the error is handled.

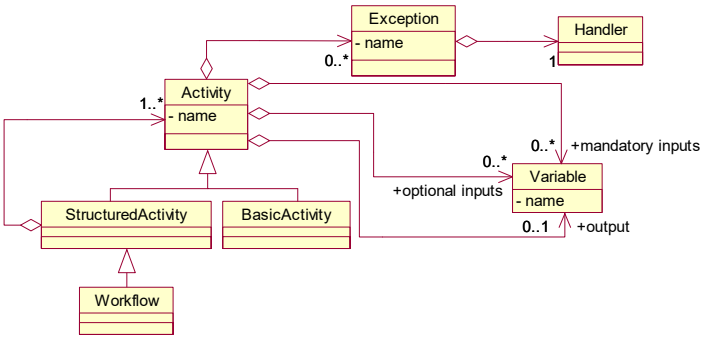

Fig.1. Basic structure of the language

3.2 Pre-defined Activities

This section describes the activities that can be used to create the workflows. We begin describing the basic activities and then describe the structured activities:

- *SEQUENCE*: An instance of the *SEQUENCE* activity executes a navigation sequence and returns a *page* value representing the final web page reached. Optionally, it can receive the following input parameters: 1) One *page* value. If provided, the page is loaded in the automatic navigation component before executing the sequence. This is useful if the configured sequence needs to start from a given page. 2) One or more values of either atomic or record type. These are needed because navigation sequences are often expressed in function of variables. For instance, a sequence automating a query on an Internet bookshop can receive as input the title and author to search. Our system uses an extension of the techniques proposed in [12] to implement the *SEQUENCE* activity, but any other method could be used.
- *EXTRACTOR*: An instance of the *EXTRACTOR* activity receives a *page* value and outputs a list value containing a list of records in the page. Our implementation uses wrapper induction techniques to generate extraction programs.
- *I/O Activities*: Set of activities for reading/writing data from/to files and databases.
- *WAIT*. It causes the workflow to wait the specified number of milliseconds.
- *THROW*: This activity throws user-defined exceptions.
- *EXPRESSION*: It receives zero or more values of any data type as input and outputs a single value. The output value is computed using an expression that can use constants, functions and the input values. The implementation of this activity supports arithmetic operations, text processing and regular expressions, date manipulation, textual similarity functions and functions to manage list values.
- *RECORD_CONSTRUCTOR*: This is the basic activity for transforming and combining data records. It receives zero or more values of any data type as input and outputs a record. The workflow creator defines the fields that form the output record. For each field, she/he needs to provide an expression to compute its value, expressed in function of the input values. The expressions used should support the same operations supported by the *EXPRESSION* activity.

- *CREATE_LIST/ADD_RECORD_TO_LIST*: The *CREATE_LIST* activity creates an empty list value. The *ADD_RECORD_TO_LIST* activity receives as inputs a list value and a record value and outputs a list value containing the result of adding the input record to the input list.
- *OUTPUT*: This activity produces the workflow output. Although a workflow usually returns a list of records, the output activity allows returning each data record to the invoking application as soon as it is available. Since web navigations can be slow, the lapse between obtaining the first output record and the last can be big.
- *Custom Activities:* It is useful to allow developers to create new activities by using a standard programming-language. For instance, this allows invoking external applications. In our implementation, custom activities are created using Javascript.

The structured activities included in the language are:

- *SWITCH*: This activity implements conditional bifurcations in the workflow. It receives as inputs zero or more values of any type. Each output arrow from the activity represents a possible execution path. Each path has an associated Boolean condition (expressed in function of the input values) which triggers its activation.
- *LOOP/REPEAT*: These activities allow creating conditional loops. They specify an exit condition typically expressed in function of the input values.
- *ITERATOR*: It allows specifying a non-conditional loop by iterating on a list of re cords. It receives a list value as input and, in each iteration, outputs a record contained in the list. It allows configuring parallel execution of its iterations. According to the results of the experimental study, this is very useful, for instance, to access in parallel detail pages of a list of extracted data records.
- *FORM_ITERATOR*: According to the results of the experimental study, web automation tasks frequently need to execute several queries on the same web form using different combinations of query parameters. While this can be done with the basic activities, it is very useful to have a specific activity for this purpose.

3.3 User-defined Reusable Components

The proposed model allows users to create reusable components of two kinds: *binary-reuse* components and *source-reuse* components. *Binary-reuse* components allow exporting an existing workflow as an activity (we call such activities "Workflow Activities") that can then be used to create new workflows. This way, sub-processes implementing functionality common to several wrappers can be easily reused.

Source-reuse components allow defining reusable templates to represent frequent *structural patterns*. Templates are reused at the source level because the implementation of structural patterns in each wrapper may suffer slight variations that prevent reuse at the binary level. The remaining of this section describes the use of templates.

At workflow creation time, workflow creators can drag and drop templates to the workspace and compose several templates to easily create wrappers that need to implement common structural patterns. A template is created in a similar way as a workflow, with the following differences:

- The template creator does not need to provide configuration information for every activity in the template. The configuration of these activities will be "filled in" when the template is used to create a workflow.

- As well as workflows, templates return an output value and can require mandatory and optional parameters. Nevertheless, when instantiating a template to create a workflow, the workflow creator may add as many new input parameters as wished. This is allowed because those additional parameters may be needed as inputs for the activities of the template left without configuration at template-creation time.
- Templates can include special activities called "*Interface Activities*". Interface activities specify a list of input parameters and one output result, but they do not specify any particular implementation. When the workflow creator uses the template to create a new workflow, she/he will specify an implementation for the Interface Activity. This implementation can be any activity or complete workflow having entries and outputs conforming to the ones defined by the interface activity. The workflow creator can also implement an Interface Activity by using another template. As well as with templates, at workflow creation time, the workflow creator may add as many new input parameters as needed to the Interface Activity.

Now, we introduce some example templates. Figure 2 shows a template called *Simple_Pagination* used to process the common kind of paginated result intervals, where the next interval is accessed by clicking on a 'Next' link or button (**NOTE**: in all workflow figures, the arrows connecting the activities represent the execution flow and the dotted lines represent the data flow: that is, how values are produced and consumed by the activities. Also notice the legend on the lower right corner of Figure 2, indicating how the values of the different data types are represented in the figures). The template receives as input a *page* value and returns a list of records. The activities the workflow creator needs to configure appear in grey in the figure. The template iterates through the result listing pages until there are no more intervals left (this is detected by a *SWITCH* activity). The *SEQUENCE* activity called *Go_to_Next_Interval* navigates to the next result interval. The *EXTRACTOR* activity obtains the list of data records in each page. The workflow creator also needs to provide an implementation for the *Process_Record* Interface Activity, which is in charge of processing each record. Now, we consider three example implementations of *Process_Record*:

- The first one is directly using the *OUTPUT* activity. This can be used when the workflow only needs to return the extracted records to the invoking application.
- The second example implementation is built using a template called *Filter_and_Transform* (see Figure 3) that: 1) Filters the extracted records according to a condition specified in the configuration of the *SWITCH* activity, and 2) Transforms the records that passed the filter according to the expressions specified in a *RECORD_CONSTRUCTOR* activity. The template also uses the *Process_Record* Interface Activity to allow further processing of the records.
- The third example implementation is a template called *Detail* (see Figure 3). This template allows accessing a 'detail' page in order to complete the data extracted for each item. It receives as input the record extracted from the result listing page. The template starts navigating to the detail page of the item (*SEQUENCE* activity). Then, it extracts the detail information (*EXTRACTOR* activity) and combines it with the input record to form a single record (*RECORD_CONSTRUCTOR* activity). The template uses a *Process_Record* Interface Activity to process the record containing the complete item. Therefore, the possible implementations for the interface activity include the three discussed options. For instance, if it is needed to access several levels of detail pages, the *Detail* template can be used recursively.

3.4 Example

This section illustrates some of the main features of the language through a wrapper automating the interaction with a web portal providing information about the incidences reported by the clients of an Internet Service Provider. When an incidence requires an intervention in the user's home, the ISP subcontracts an enterprise partner to perform it. The example wrapper is used by an enterprise partner of the ISP to automate the retrieval of the incidences data that a given worker could attend.

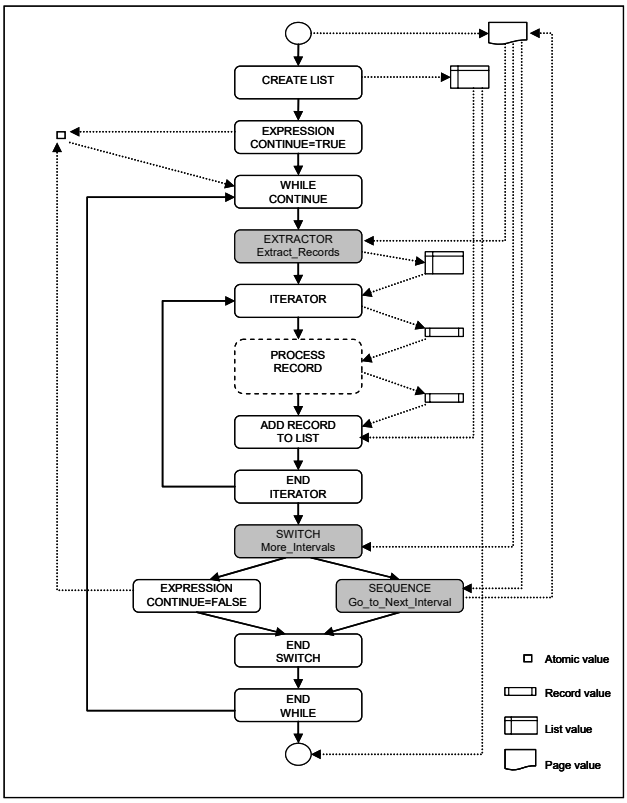

Fig. 2. *Simple_Pagination* template

The wrapper has the following inputs: the login/password to access the portal, the zipcode indicating the current location of the worker of the enterprise partner, the maximum distance the worker could travel to solve an incidence, and the type of incidences the worker can solve. The wrapper should perform the next steps:

1. Authenticate in the ISP web portal using the login/password pair.
2. Fill in a search form to obtain all the active incidences located near the input zipcode. The incidence listing is paginated using a 'Next' link.
3. Extract all the incidences data. The data shown in the result listing includes the incidence type. If the incidence is of the type the worker can attend, then it is needed to access a detail page to obtain additional information, such as the distance with

respect to the input zipcode. The returned incidence data must include a derived field indicating the deadline for attending the incidence; it is computed from two extracted items: the date when the incidence was open and the maximum number of days agreed between client and ISP.

4. Return the incidences of the input type having distance less than the input maximum distance. In addition, the wrapper should be able of dealing with one error condition: the incidences search form can return a message error when the input zipcode is outside the area assigned to the partner.

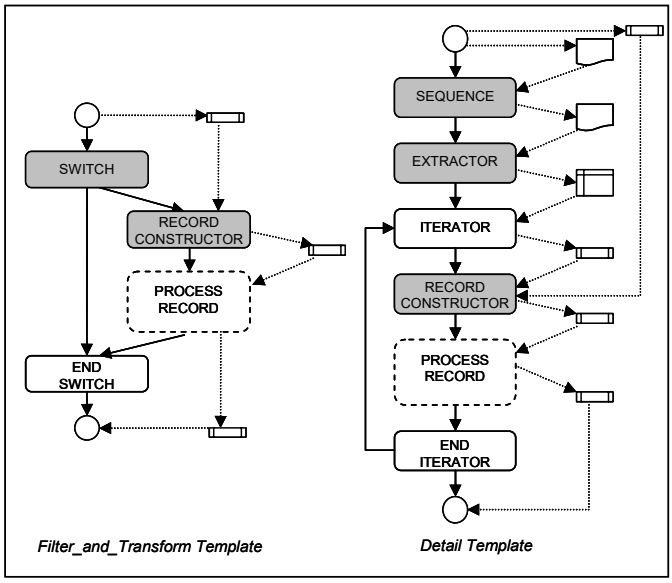

Fig. 3. Filter_and_Transform and Detail templates

The process flow of the wrapper executes two high-level sub-processes: one *workflow activity* (recall section 3.3) called *Get_Search_Page* and an implementation of the *Simple_Pagination* template (see Figure 2). Figure 4 shows the complete wrapper.

The internal workflow of *Get_Search_Page* is not shown in the figure due to space constraints. It first performs the authentication process and the search using a *SEQUENCE* activity. The sequence accesses the page containing the authentication form, fills in the *LOGIN* and *PASSWORD* fields and submits the form. Then, it executes the incidences search by accessing the query form, filling in the *ZIPCODE*. Then, a *SWITCH* activity called is used to check if the page source code contains the text '*Incorrect zipcode'*. If the message is found, the *Throw_Search_Error* activity ends the process returning an exception. Notice that this sub-process could be reused by other wrappers using the same source and search form but processing the results differently.

Now, we describe how to fill in the *Simple_Pagination* template to process the search results (see Figure 4). The steps needed to fill in the template are:

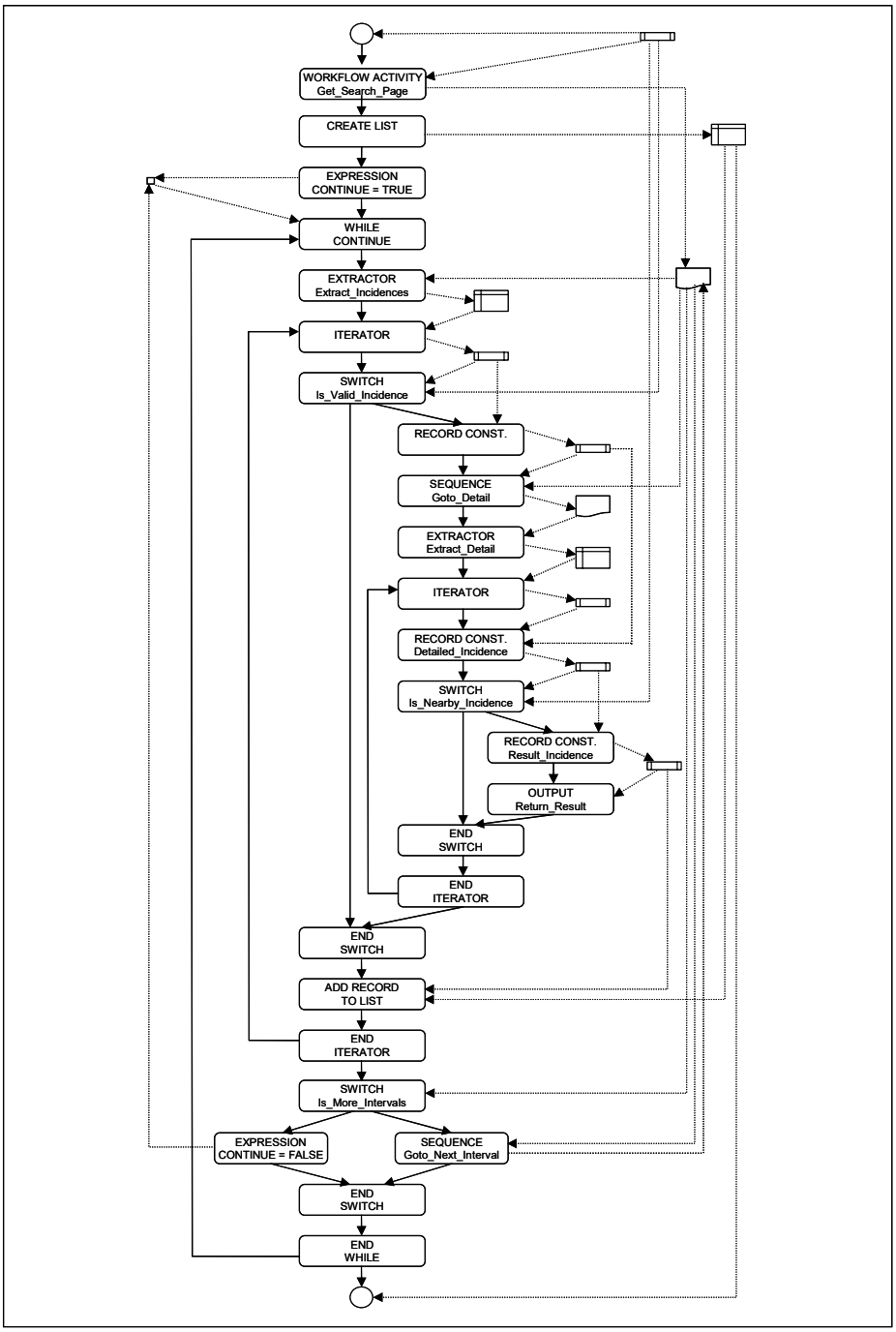

Fig. 4. Full wrapper using the *Simple_Pagination* template

1. We need to "fill in" the template by configuring the *Go_to_Next Interval SEQUENCE* activity with the sequence for navigating to the next result interval (e.g.. clicking on an anchor) and the *Extract_Records EXTRACTOR* activity with the needed extraction rules to obtain the incidences list from the search result.
2. The *Process_Record* Interface Activity of *Simple_Pagination* can be implemented using an instance of the *Filter_and_Transform* template to filter the incidences of the input type using the *SWITCH* activity. It is not needed to configure the *RECORD_CONSTRUCTOR* activity since its default settings are valid.
3. The *Process_Record* Interface Activity from the *Filter_and_Transform* template used in step 2 can be implemented using the *Detail* template (the incidence detail page is accessed only for the incidences of the input type). We need to provide the sequence for navigating to the detail page and the extraction rules to obtain the detail data. We also need to configure the *SWITCH* activity to filter the incidences according to the input *DISTANCE*, and the *RECORD_CONSTRUCTOR* activity to add the additional field *DEADLINE_DATE*.
4. The *Process_Record* Interface Activity from the previous *Detail* template can be implemented using again *Filter_And_Transform* to filter all the incidences of the input type verifying that its distance is less than the input maximum distance. Finally, the *Process_Record* Interface Activity from the *Filter_And_Transform* template is implemented by simply using the *OUTPUT* activity.

4 Related Work

Most previous works on wrapper generation have focused on the building blocks for web automation: web data extraction and automatic web navigation. The web data extraction problem has been addressed for instance in [4],[5],[9],[10],[11],[12],[15], [16]. [8] provides a survey. Techniques for automatic generation of web navigation sequences were proposed in [3] and [12]. These works do not consider the problem of how to specify the logic of the complete wrapper. Nevertheless, they provide the foundations for the *SEQUENCE* and *EXTRACTOR* activities of the proposed model. There are two kinds of works addressing the problem of building complete wrappers:

- Some works such as [10] define specialised languages for programming wrappers. Our proposal has several advantages: 1) it allows graphically specifying the wrapper logic: this way, wrappers are simpler to create and maintain and programming skills are not required; 2) it encapsulates the data extraction and automatic navigation tasks, leveraging on semi-automatic methods.
- Other works such as [5],[9],[11],[13] propose techniques to create complete wrappers without needing programming skills. Nevertheless, these systems implicitly assume the query wrapper model. As it has been previously discussed, this model is not suitable for a substantial number of web automation applications.

In the industrial arena, many web automation tools have appeared. QL2 (*http://www.ql2.com*) and NewBie (*http://www.newbielabs.com*) follow the approach of providing specialised programming languages. Our proposal has the same advantages over these tools already mentioned for the research systems using the same approach. Another interesting tool is Dapper (*http://www.dapper.com*) which allows

creating and sharing wrappers between final users. The wrappers that can be created using Dapper are roughly equivalent to those supported by the query wrapper model. As it has been already discussed, this is not enough for enterprise-class web automation. The Kapow Robomaker tool (http://www.openkapow.com) also uses a workflow approach for web automation. The approach proposed in this paper has a number of advantages with respect to Robomaker: 1) Robomaker does not encapsulate complex data extraction tasks in activities. The extraction of a list of data records requires an activity in the workflow to extract each record field. Optional attributes in the records require bifurcations in the workflow. This leads to large workflows even for relatively simple tasks. In addition, their model does not support using semi-automatic methods for extraction, 2) Robomaker does not support defining reusable components, 3) Robomaker does not support other functionalities such as user-defined exceptions.

References

1. W. Aalst, A. Hofstede, B. Kiepuszewski, A. Barros. Workflow patterns. Distributed and Parallel Databases 14(1): 5-51 (2003).
2. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison Wesley, Reading, Massachussetts, 1995.
3. Vinod Anupam, Juliana Freire, Bharat Kumar, Daniel F. Lieuwen: Automating Web navigation with the WebVCR. Computer Networks 33(1-6): 503-517 (2000)
4. A. Arasu, H. Garcia-Molina, Extracting Structured Data from Web Pages, in: Proceedings of the 2003 ACM SIGMOD International Conference, 2003, pp. 337-348.
5. R. Baumgartner, S. Flesca, G. Gottlob. Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto, in: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNR), 2001.
6. Oasis WS-BPEL. Web Services Business Process Execution Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.
7. BPMN: Business Process Modelling Notation. http://www.bpmn.org.
8. Chia-Hui Chang, Kayed M. Girgis, M.R., Shaalan, K.F. A Survey of Web Information Extraction Systems. IEEE Transactions on Knowledge and Data Engineering 18 (10), pp. 1411-1428. 2006
9. R. Doorenbos, O. Etzioni, D. S. Weld: A Scalable Comparison-Shopping Agent for the World-Wide Web. Agents 1997: 39-48
10. T. Kistlera, H. Marais. WebL: A Programming Language for the Web. En Proceedings of the 7th International World Wide Web Conference, 1998, pp 259-270.
11. C.A. Knoblock, K. Lerman, S. Minton, I. Muslea, Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach, in: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1999.
12. A. Pan, J. Raposo, M. Álvarez, J. Hidalgo and A. Viña. Semi Automatic Wrapper-Generation for Commercial Web Sources. Proceedings of IFIP WG8.1 EISIC. 2002
13. A. Sahuguet, F. Azavant. WysiWyg Web Wrapper Factory (W4F). Proceedings of the 8th International World Wide Web Conference, 1999.
14. G. Wiederhold. Mediators in the architecture of future information systems. Computer, 25(3), March 1992.
15. Y. Zhai, B. Liu, Structured Data Extraction from the Web Based on Partial Tree Alignment. IEEE Trans. Knowl. Data Eng. 18(12): 1614-1628 (2006).
16. Y. Zhai, B. Liu, Extracting Web Data Using Instance-Based Learning. Proceedings of the 16th International World Wide Web Conference. 2007