Journal of Universal Computer Science, vol. 14, no. 11 (2008), 1838-1856
submitted: 30/9/07, accepted: 25/1/08, appeared: 1/6/08 © J.UCS

A Workflow Language for Web Automation

Paula Montoto, Alberto Pan, Juan Raposo, José Losada
Fernando Bellas, and Víctor Carneiro
(University of A Coruña, Coruña, Spain
{pmontoto,apan,jrs,jlosada,fbellas,viccar}@udc.es)

Abstract: Most today's web sources do not provide suitable interfaces for software programs to interact with them. Many researchers have proposed highly effective techniques to address this problem. Nevertheless, ad-hoc solutions are still frequent in real-world web automation applications. Arguably, one of the reasons for this situation is that most proposals have focused on query wrappers, which transform a web source into a special kind of database in which some queries can be executed using a query form and return resultsets that are composed of structured data records. Although the query wrapper model is often useful, it is not appropriate for applications that make decisions according to the data retrieved or processes that use forms that can be modelled as insert/update/delete operations. This article proposes a new language for defining web automation processes that is based on a wide range of real-world web automation tasks that are being used by corporations from different business areas.

Key Words: web wrappers, data mining, web automation, web information systems
Category: D.1.7, H.2.5, H.2.8, H.3.3

1 Introduction

Most today's web sources were designed to be easily used by humans, but they do not provide suitable interfaces so that software programs can interact with them, which can be considered a hindrance for the Web to reach its full potential. Recently, a growing interest has arisen in automating the interactions with a web site by using so-called web automation applications. Most previous research proposals focus on wrappers, which abstract the complexities involved in automating a task on a web source and provide a programmatic interface. A wrapper must address several difficult tasks, the most important being: executing automated navigation sequences through web sites and obtaining structured data records from the resulting HTML pages.

Most current proposals focus on the second task only, cf. [Doorenbos et al., 1997] [Kushmerick, 2000] [Baumgartner et al., 2001] [Knoblock et al., 2000] [Arasu and Garcia-Molina, 2003] [Zhai and Liu, 2006] [Zhai and Liu, 2007] [Álvarez et al., 2008] ([Chang et al., 2006] provides a survey). The first task has been paid much less attention, although it has been addressed in a number of proposals, e.g., [Anupam et al., 2000] or [Pan et al., 2002]. These approaches use different techniques, but a common feature is that they allow to create wrappers quickly without requiring any programming skills since they rely on a variety of graphical tools and intelligent learning techniques. They further assume a particular underlying model to which we refer to as the "query wrapper model". Query wrappers transform a web source into a

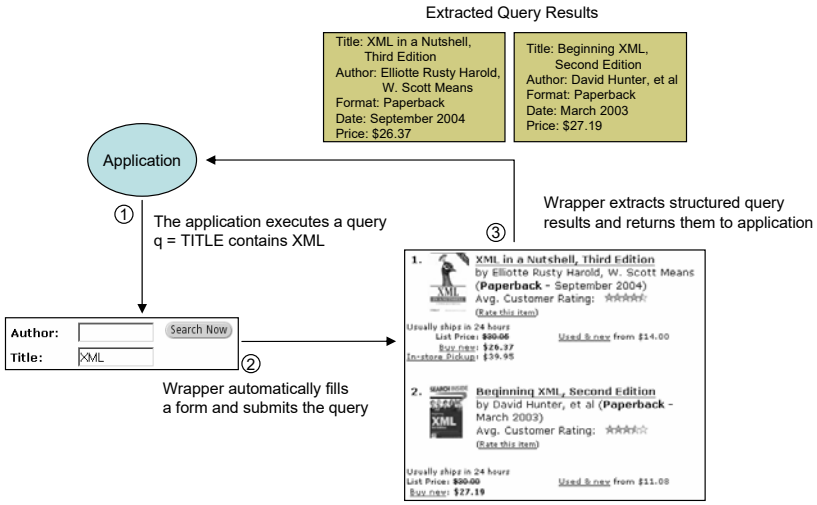

Figure 1: Query Wrapper.

special kind of database in which queries can be executed using a form and produce a resultset that is composed of structured data records. The query wrapper model typically assumes a pre-defined list of execution steps: first, a navigation sequence is used to fill in a query form automatically; then, intelligent data extraction techniques are used to gather the results from the target HTML pages. Figure 1 sketches the execution flow of a wrapper able to query an Internet bookshop. Query wrappers may also allow for paginated result listings in which detail pages need to be fetched to extract further information about each record. Web automation applications are a reality today, and they are being used in business areas such as competitive intelligence, comparative shopping, and B2B integration. Nevertheless, in spite of the many published research proposals, ad-hoc solutions are still frequent in real-world applications. One of the main reasons is that, despite the query wrapper model being useful, it does not fit some important web automation applications. For instance, many tasks involve making decisions according the data retrieved so that navigation can continue; other tasks use web forms that can be easily modelled as insert/update/delete operations; an example of web automation task that does not fit the query model is extracting book data from an Internet bookshop and, according to the price and availability of each book, deciding to push either the "buy new" button, the "buy used" button, or none of them.

In this article, we propose a graphical language for creating wrappers. Note that our wrappers automate the interaction with a single web site for a single purpose. For tasks that require to combine and/or orchestrate several web sources, our approach enables the wrappers to participate as basic components in usual data and process integration archi-

	B2B Web Automation	Batch Data Extraction	Meta-search	Technology and Business Watch	Account Aggregation	Total
Applications Number	7	6	2	7	2	24
Wrappers Number	45	45	38	145	118	391
Query Wrappers	53%	80%	100%	13%	95%	57%
Bifurcations in non-query wrappers	92%	67%	-	26%	0%	54%
Wrappers with User-Defined Error Management	37%	0%	0%	0%	0%	4%
User-Defined Error Management in Non-Query Wrappers	68%	0%	-	0%	0%	11%
Wrappers with Asynchronous Operations	13%	0%	0%	0%	0%	2%
Asynchronous Operations in Non-Query Wrappers	24%	0%	-	0%	0%	4%

Table 1: Results of the experimental study.

tectures such as data mediators [Wiederhold, 1992] or Business Process Management systems. Another key goal for our language is to be simple to use: wrappers should be created graphically, and the language should not include features that are not useful in practice but introduce unnecessary complexity. Programming-skills should not be necessary, at least in the vast majority of cases. As a source of inspiration, we have studied orchestration technologies such as BPMN [OMG, 2003] or WS-BPEL [Oasis, 2003], and patterns [Aalst et al., 2003], which are also concerned with specifying complex execution logic in a simple, graphical way. To provide our proposal with firm roots, we have studied a wide range of real-world web automation tasks, which are being used by corporations from different business areas.

The rest of the article is structured as follows. Section 2 reports on our motivation; Section 3 describes our proposal, which is a graphical design language inspired by classic workflow systems, but adapted to the particular needs of web automation; Section 4 describes an example to illustrate our language; Section 5 describes related work in this area, and Section 6 concludes the article.

2 Motivation

To guide the development of our language, we have studied a total of 391 wrappers used in 24 real-world web automation applications that were developed during the last three years by a European company. We have chosen applications in quite different business areas to increase the generality of our conclusions: B2B web automation, i.e., automating repetitive operations with other organisations through a web interface, large volume batch data extraction, Internet meta-search applications, technology and business watch,

i.e., monitoring web information that is relevant for business and/or research purposes, such as competitors prices or new patents, and web account aggregation.

We organised the study into two stages: first, we studied existing workflow technologies for BPM, e.g., [Aalst et al., 2003] [OMG, 2003] [Oasis, 2003], to identify a set of features relevant to web automation applications, and we also analysed if the wrappers fitted the query wrapper model; we then searched for common structural patterns and analysed if our language should allow for defining and reusing them.

2.1 Workflow Requirements

The features that we found useful are: conditional bifurcations, error management, parallelism, asynchronous events and subprocesses; we also analysed if the wrappers fitted the query wrapper model. Below, we report on our conclusions, cf. Table 1:

1. Only 57% of the wrappers fit the query wrapper model. The percentage varies with the application area: 100% of the wrappers in meta-search applications fit this model, whereas the percentage is 53% in B2B applications. Our conclusion is that this model is too simple for many real-world web automation applications.

2. Roughly 54% of the wrappers that do not fit the query wrapper model require bifurcations. Therefore, our proposal should support them.

3. Most of the wrappers require or could benefit from the following error management policies: i) indicating which action to perform when an error happens, e.g., either ignore it or halt the process and return the error to the caller; ii) executing retries, e.g., when executing web navigation sequences. In addition, 37% of the wrappers in the B2B application area required or could benefit from user-defined, application-specific exceptions. Therefore, our proposal should support these features.

4. We observed that parallelism is very useful in two cases: i) 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. fetching a details page; since navigation sequences can be slow, processing the records in parallel can significantly improve performance; ii) we have also realised that some applications execute the same query multiple times using different query parameters and then merge the results; thus it seems useful for our language to support specifying the parallel execution of multiple queries on the same web form. 84% of the wrappers could benefit from either one or both of these kinds of parallelism, and no wrapper required other types. 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, our approach enables the wrappers to participate as components in usual data and process integration architectures such as data mediators [Wiederhold, 1992] or BPM systems. Therefore, we conclude that the language should not include more general support for parallelism because it would increase the complexity and its benefits would be unclear.

5. Regarding asynchronous operations, some web pages may change using AJAX without reloading them. In such cases, it would be useful if the wrapper could be notified of changes to a region asynchronously. Six of the wrappers we have studied have to deal with these sources, and they resort to polling selected target regions at regular intervals. The reason is that most automatic navigation systems use browsers for navigating and hosting HTML pages, and it is difficult to identify content-change events with current browser APIs. Since AJAX sources are gaining importance, our proposal allows for polling.

6. Regarding subprocesses, we have found only six wrappers that actually use some kind of subprocess. Furthermore, we have detected that the implementation and maintenance of the most complex wrappers could be simplified by using subprocesses. Therefore, we conclude that this feature is desirable for our proposal.

2.2 Structural Patterns

We have identified several structural patterns that occur in many wrappers with slight variations. Building such wrappers might be eased by defining source-level reusable templates to implement them, so that only the behaviour that is specific to a wrapper must be configured. Since these patterns occur frequently and some of them are relatively complex, supporting this feature might greatly simplify the design of complex wrappers by non-programmers. In this section, we just briefly describe some common structural patterns found in our study, cf. Section 3.4 for further details.

– Simple_Pagination: This pattern refers to paginated result listings in which the navigation sequence required to fetch the next chunk of results is always the same, e.g., clicking the “Next” link.

– Multiple_Sequence_Pagination: This pattern is used to process a paginated result listing in which the navigation sequence required to fetch the next chunk differs from page to page, e.g., clicking on the “11–20” link to navigate to the second chunk, on the “21–30” link to navigate to the third one, and so on. There are several variations of this pattern on which we do not report for the sake of briefness.

– Detail: This pattern refers to executing a navigation sequence according to the data records extracted, e.g., navigating to the details page. There are several variations of this pattern, too.

– Filter_And_Transform: This pattern receives a record and filters or transforms it according to a pre-defined condition and transformation.

Regarding asynchronous sources, most automatic navigation systems rely on standard browsers, which makes it difficult to identify content-change events at the desired granularity; thus polling at given intervals is the most common solution. We define the following related structural patterns:

– Polling: Executes a task at a specified time interval until a condition is met.

– Monitor_List: This allows to monitor changes to a list of records, e.g., the results of a query or the articles in the home page of an Internet news service. This pattern allows to define an action to perform when a new record is found, modified or removed. Combined with the Polling pattern, it allows simulating asynchronous operation since each type of change in the monitored list triggers an action.

3 Description of Our Language

In this section, we describe our language and motivate the main design choices using the evidence gathered from the previous study.

3.1 Encapsulate Data Extraction and Web Navigation Tasks

Many research proposals have reported on a variety of techniques to automate web navigation and data extraction, namely:

– In proposals such as [Anupam et al., 2000] or [Pan et al., 2002], web navigation sequences are recorded by a plug-in that monitors the navigation actions performed by a user. The sequence thus captured is then transformed into a script that can be repeated by a web navigation component.

– As for data extraction, there are proposals in which the user needs to provide some examples of the data to extract, cf. [Kushmerick, 2000] [Baumgartner et al., 2001] [Pan et al., 2002] [Knoblock et al., 2000] [Zhai and Liu, 2007], but others can analyse the pages without any user intervention, cf. [Arasu and Garcia-Molina, 2003] [Crescenzi and Mecca, 2004] [Zhai and Liu, 2006] [Álvarez et al., 2008].

These techniques allow the most complex tasks in web automation to be performed in a fast and easy way even by users with little or no programming skills. To preserve these advantages, our proposal uses a high-level approach that encapsulates automatic web navigation and data extraction functionalities into built-in activities to which we refer to as SEQUENCE and EXTRACTOR, respectively. An instance of the SEQUENCE activity executes a navigation sequence configured by the user, and returns

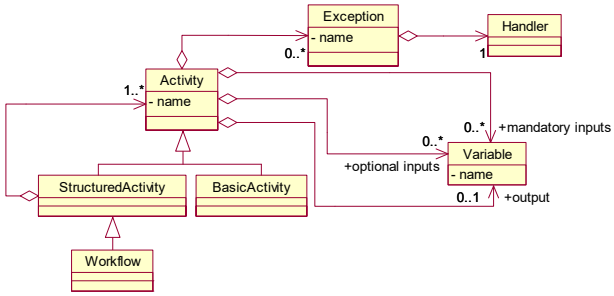

Figure 2: Basic language structure.

the furthest web page reached. An instance of the EXTRACTOR activity gets a page as input and outputs the list of structured data records found in that page. Our implementation uses an extension of the techniques proposed in [Pan et al., 2002] to implement the SEQUENCE activity, and wrapper induction techniques to implement the EXTRACTOR activity, but other methods might be used, as well.

3.2 Data Model

We refer to the data instances handled in a process flow as values, and we assume they have a structured type [Abiteboul et al., 1995], which is defined as follows:

- Our language supports the usual atomic data types found in common programming languages, e.g., string, int, long, double, float, date, boolean, binary, and url. There is another type called page that encapsulates the information required to fetch a web page, i.e., its URL, the cookies, and other session information.
- If $t_1, \ldots, t_n$ are types, we then can define a new type $T\langle t_1, t_2, \ldots, t_n\rangle$ called record; $t_1, t_2, \ldots, t_n$ are the fields or attributes of T. An instance of T is a tuple of the form $\langle v_1, v_2, \ldots, v_n\rangle$, where each v_i is an instance of t_i. We call such instances record values. Optionally, a record type may define a key, which is a set of fields that identify the real-world entity that record represents uniquely.
- If t is a type, we then can define a new type $T[t]$ called list. An instance of $T[t]$ is a sequence of elements $[r_1, r_2, \ldots, r_m]$, each of which is of type t. We call such elements list values.

3.3 Workflow Model

Figures 2–5 show a partial meta-model for our language. A workflow gets a set of variables as input and returns a single variable as output, cf. Figure 2. The value of

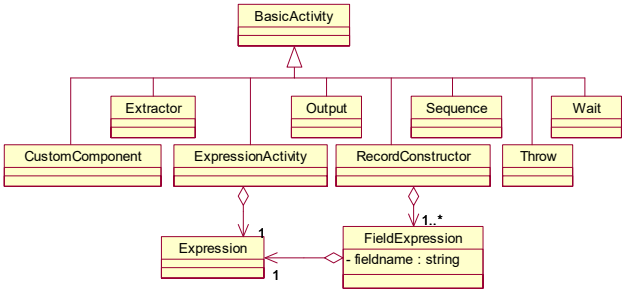

Figure 3: Basic activities.

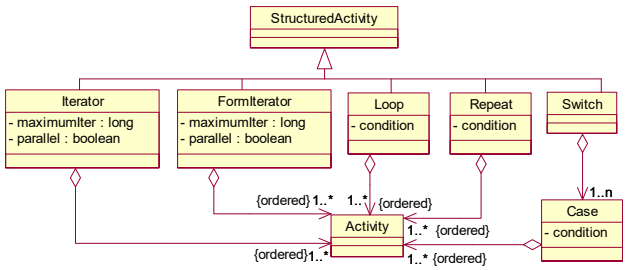

Figure 4: Structured activities.

a variable is an instance of a 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, namely: basic activities perform the actions in a workflow, and are described later, cf. Figure 3; structured activities include looping and bifurcations, each of which can enclose one or more sequences of activities, cf. Figure 4. Note that a workflow can be seen as a subclass of Structured Activity. Although not shown in the diagram, some activities may impose constraints on the variables they use, e.g., an EXTRACTOR requires an input of type page and returns a list value.

To handle errors, the language leverages the standard concept of exception, which can be either pre-defined or user-defined, cf. Figure 5. Pre-defined exceptions represent generic, typical runtime errors, e.g., HTTP errors, or timeouts, or errors found while extracting data records, e.g., invalid record type. User-defined exceptions can be generated using the THROW activity. Each exception has a handler, that can either ignore, throw the exception, or retry the faulty action several times.

Graphically, activities are represented as squared boxes. Typically, each activity has one input port and one exit port, except bifurcations, which can have several exit ports.

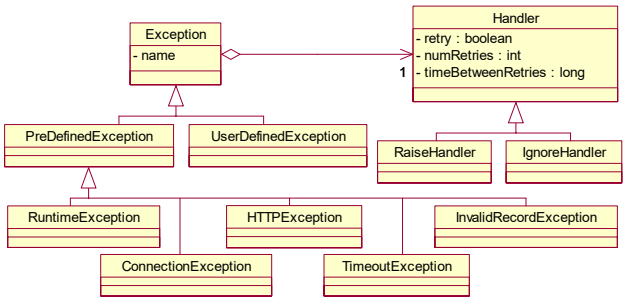

Figure 5: Exceptions.

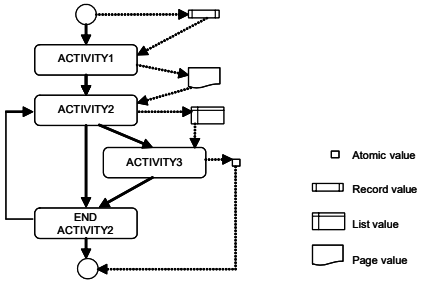

Figure 6: Sample workflow.

The activities used to represent loops are represented using two boxes that delimit the beginning and the end of the loop. The activities in a workflow are executed sequentially in the order established by their interconnections.

In our implementation, workflows are created by adding activities to a workspace and interconnecting them. Each activity has a wizard to configure it. For instance, the SEQUENCE activity is configured with the navigation sequence to execute, which may be customised according to the inputs. For instance, Figure 6 shows a sample workflow in which the arrows represent executions flows and the dotted lines represent data flows.

Below, we briefly describe each activity:

– SEQUENCE: These activities execute navigation sequences and return page values. Optionally, they can have the following input parameters: i) one page value that is loaded before executing the sequence; ii) one or more atomic or record values that allow parameterising parts of the sequence, e.g., the data used to fill in a form.

– EXTRACTOR: These activities get page values and output list values that account

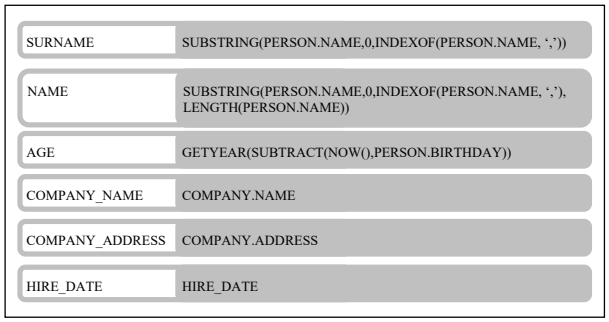

Figure 7: Example of a RECORD_CONSTRUCTOR.

for the list of records in those pages.

– SWITCH: These activities implement conditional bifurcations. They get zero or more values as input, and each outgoing arrow represents an execution path, each of which has a Boolean condition that usually relies on the input parameters.

– EXPRESSION_ACTIVITY: These activities get zero or more values as input and output a single value that is computed using a custom expression. Our implementation supports common arithmetic operations, text processing and regular expressions, date manipulation and textual similarity functions, list or record manipulation.

– RECORD_CONSTRUCTOR: These are the basic activities for creating, transforming and combining data records. They get zero or more values as input and output custom records. For each field, the user needs to provide an expression to compute its value. For instance, Figure 7 depicts how a record constructor activity can be configured to implement the following requirements: it gets three values as input, namely: i) a register called PERSON that has fields NAME and BIRTHDAY, a register value named COMPANY with fields NAME and ADDRESS, and a date value called HIRE_DATE that refers to the date when the person was hired by the company. The workflow combines these data into a single record value. Furthermore, it needs to divide both the name and surname into two fields and compute the age of each person. The user must then create fields in the output record, each of which is defined using an expression: the expressions to compute most fields are trivial, but computing the SURNAME, NAME and AGE fields requires to use constants, functions and the input values.

– LOOP/REPEAT: These activities allow to create conditional loops. They receive one or more values as inputs and have an exit condition.

- ITERATOR: These activities allow to iterate on a list of records. They get a list value as input and output a record in each iteration. Iterations can be executed in parallel to improve the response time, and the user can control how many parallel threads are executed to avoid excessive load.
- OUTPUT: These activities produce the output of a workflow, i.e., to return the records of the result list one by one as they become available.
- CREATE_LIST/ADD_RECORD_TO_LIST: A CREATE_LIST activity creates an empty list value; an ADD_RECORD_TO_LIST activity gets a list value and a record value as input and outputs a list value in which the record has been inserted at a user-defined position.
- WAIT: These activities cause a workflow to wait for a number of milliseconds, which is useful for polling, for instance.
- THROW: These activities are used to throw user-defined exceptions.
- FORM_ITERATOR: These activities allow to execute several queries using different combinations of query parameters. They are configured with the navigation sequence required to fill in and execute a given form, and can get one or more values that can be used to generate the combinations of query parameters to fill the form in. The values of the form fields like drop-down lists, radio or check buttons are inspected by a wizard so that the user can easily specify appropriate combinations. The iterations can be configured to run in parallel. For instance, consider a workflow that gets a list of cities and a list of professions and searches for every combination on an on-line job database. Assume that the query form has three fields: city, category and salary_range; the former two are text fields, whereas the latter is a drop-down list with three options. Thus, if the input list of cities has two values and the input list of categories has three values, the workflow needs to execute 18 queries to gather the desired information.
- Input/Output Activities: These activities allow to read/write data from/to files and databases.
- Custom Activities: It is useful to allow developers to create new activities by using a standard programming language (we use Javascript). For instance, these activities are useful to invoke external applications to perform custom tasks.

3.4 User-defined Reusable Components

Our proposal allows to create both binary- and source-level reusable components. The former allow to export an existing workflow as an activity that can then be used to create new workflows (we call such activities Workflow Activities). Thus, subprocesses that implement a piece of functionality that is common to several wrappers can be reused by

exporting them as Workflow Activities. Source components allow to define templates to represent frequently used structural patterns. Templates are reused at the source level because the implementation of structural patterns in each wrapper may require slight variations that prevent reuse at the binary level.

When a workflow is created, users can drag and drop templates to the workspace and compose them to create new wrappers. A template is created as if it were a workflow, but there are some differences, namely:

- The user does not need to configure all of the activities in the template, since this will be accomplished when the template is instantiated in a workflow.
- Templates return a single output variable and can require mandatory and optional input parameters. Nevertheless, when a template is instantiated, users may add as many new input parameters as necessary. Such parameters may be necessary as input for the activities that are not configured when the template is created.
- Templates can include special activities called Interface Activities that are analogous to methods in a Java interface since they specify a list of input parameters and one output result, but they do not specify any implementation. When the user uses the template to create a new workflow, he or she has to specify an implementation for each interface activity; the implementation may range from a simple activity to a complex workflow or event another instantiated template.

Now, we introduce some useful templates. Figure 8 shows a template called Simple_Pagination that is intended to process a common kind of paginated result pages, cf. Section 2.2. It gets a page value as input and returns a list of records. The activities the user needs to configure are depicted in gray. The template iterates through the result pages until there are no further chunks left, which is detected by the SWITCH activity. The SEQUENCE activity called Go_to_Next_Chunk navigates to the next chunk. The EXTRACTOR activity called Extract_Records gathers the list of data records in each result page. The user also needs to provide an implementation for the interface activity called Process_Record, which is responsible for processing each record. Below, we report on three sample implementations:

- The simplest one is to use an OUTPUT activity, which can be used as long as the workflow just needs to return the records extracted.
- Another implementation might build on the Filter_and_Transform template in Figure 9. It first filters the records by using the SWITCH activity; then, it transforms the records that pass the filter using a RECORD_CONSTRUCTOR activity. By default, the SWITCH activity in the template specifies a condition that evaluates to true, and the RECORD_CONSTRUCTOR activity simply outputs the record it gets as input. The template also uses the Process_Record interface activity to allow further processing of the records.

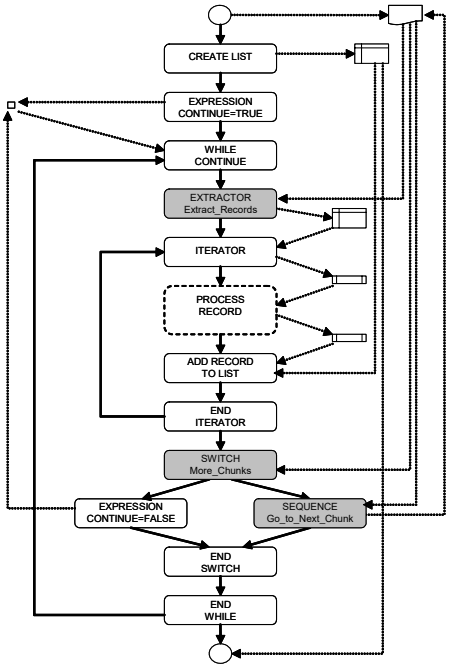

Figure 8: Template Simple_Pagination.

– Our last implementation builds on the Detail template in Figure 9, which is useful when the web automation task needs to fetch detail pages to complete the data extracted for each item. The template starts by navigating to the detail page of each record; then, it extracts the detail information and combines it with the input record to form a single record. This is accomplished by a RECORD_CONSTRUCTOR activity. Although it is depicted in gray, it does not need to be configured if the default behaviour is appropriate. This template uses the Process_Record interface activity again. If it is necessary to fetch several levels of detail pages, the Detail template can be used recursively.

We have also defined two templates called Polling and Monitor_List to deal with sources that change asynchronously, cf. Section 2.2. We, however, do not report on them due to space limitations.

4 An Example

This section illustrates some of the key features of our language by means of an example that was inspired by a real-world B2B web automation application. The sample wrapper

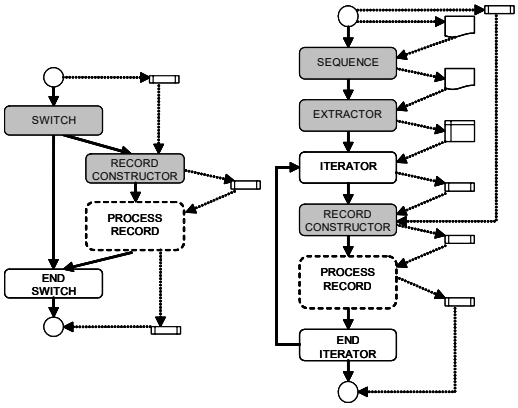

Figure 9: Templates Filter_and_Transform and Detail.

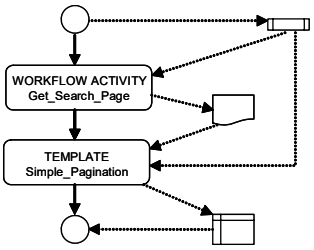

Figure 10: High-level activities.

implements a process for the fictitious company AcmeInstall, which is a local company that has established a partnership with an Internet Service Provider, or ISP for short. When an ISP client in the local area of AcmeInstall reports a problem that requires a technician to work at his or her place, the ISP subcontracts AcmeInstall. The ISP reports new problems to AcmeInstall by means of a web portal.

The wrapper we need to design must log on to the ISP portal and gather the problems with which a worker can deal according to where he or she is located and the type of problems he or she can solve. The wrapper has the following inputs: the login and password required to log on to the ISP portal, the zip code that indicates where each worker is located, the maximum distance between the worker and the location where the problem is to be solved, and the type of problems the worker is able to solve. To simplify the process, we assume that each worker can only solve one type of problem. To meet these requirements, the wrapper must perform the following actions:

1. Log on to the ISP web portal using the input login/password pair.

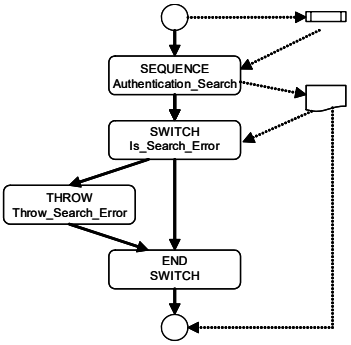

Figure 11: Workflow activity Get_Search_Page.

2. Fill in a search form to gather all of the active problems that are located near the input postal code. The problem listing is paginated, and the chunks are reached by means of a typical "Next" link. The wrapper must deal here with eventual errors if the input zip code is not within the geographical area assigned to AcmeInstall.

3. Extract the data about the active problems, which includes their types. If a problem is of the appropriate type, it is then necessary to click on its "More info" link to gather additional information, e.g., the distance to the input zip code. The data returned must include a derived field to indicate what the deadline is, which must be computed from the date when each problem is reported and the maximum delay agreed between the corresponding client and the ISP.

4. Return all of the problems of the appropriate type that are within the maximum input distance.

The process flow of this wrapper is defined as the execution of two high-level subprocesses, cf. Figure 10: a workflow activity called Get_Search_Page, which is in charge of logging on to the ISP portal, executing searches, and handling errors; then, the flow executes an instantiation of the Simple_Pagination template, which is in charge of processing the search results. The Get_Search_Page subprocess gets a ZIPCODE, a LOGIN and a PASSWORD as input. It first performs the authentication process and the search using a SEQUENCE activity called Authentication_Search, cf. Figure 11. This activity fetches the page that contains the authentication form, fills in the LOGIN and PASSWORD fields and submits the form. Then, it executes the search by fetching the query form, filling in the ZIPCODE and submitting the form. Then, a SWITCH activity called Is_Search_Error is used to check if the page contains "You have entered an incorrect zip code". If the message is found, the Throw_Search_Error activity returns exception IncorrectZipcode to the caller and the process finishes.

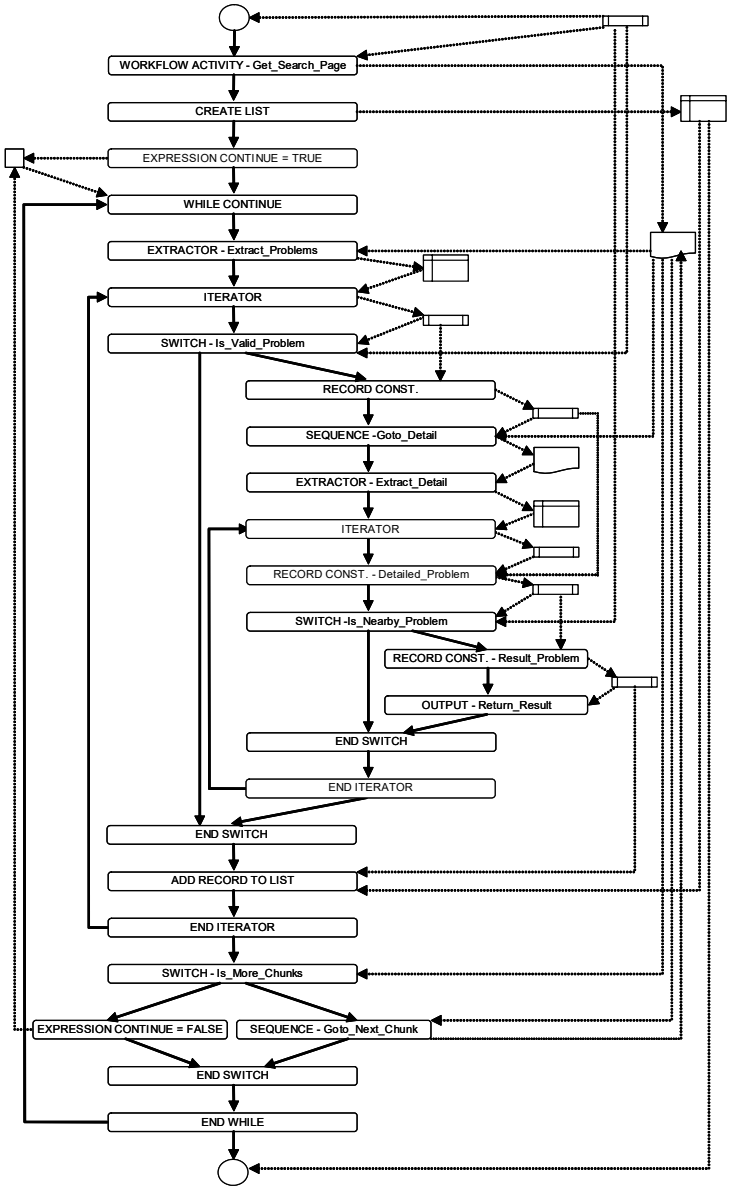

Figure 12: Complete wrapper using template Simple_Pagination.

Now, we describe how to instantiate the Simple_Pagination template to extract and process problem records. The input page for the activity is the page output by the Get_Search_Page subprocess. It returns a list of records of type problem. The process

is as follows, cf. Figure 12:

1. To implement pagination, we use the Simple_Pagination template, which is instantiated by configuring the Go_to_Next_Chunk activity so that it clicks on a link labelled "Next" to fetch further chunks of records, and the Extract_Records activity with the appropriate extraction rules.

2. The Process_Record interface activity of Simple_Pagination is implemented by means of an instance of the Filter_and_Transform template to filter out the problems according to their type. Note that it is not necessary to configure the RECORD_CONSTRUCTOR activity since its default settings are appropriate.

3. The Process_Record interface activity in the Filter_and_Transform template used in the previous step is implemented by means of the Detail template; we just need to provide the sequence for navigating to the detail page of every record and the extraction rules needed to gather the detail data. The configuration of the RECORD_CONSTRUCTOR activity must also be extended to add the additional derived field DEADLINE_DATE.

4. The Process_Record interface activity in the Detail template used in the previous step is implemented by means of the Filter_and_Transform template to filter the problems and verify that they are within the maximum input distance. The Process_Record interface activity in the Filter_and_Transform template is implemented by means of a simple OUTPUT activity.

5 Related Work

Web wrapper generation has been an active research field for years. Most previous proposals have focused on web data extraction and automatic web navigation problems, but the former is the one that has been paid more attention, cf. [Doorenbos et al., 1997] [Kushmerick, 2000] [Baumgartner et al., 2001] [Knoblock et al., 2000] [Arasu and Garcia-Molina, 2003] [Zhai and Liu, 2006] [Zhai and Liu, 2007] [Álvarez et al., 2008] ([Chang et al., 2006] provides a survey). Techniques for automating the generation of web navigation sequences were proposed in [Anupam et al., 2000] or [Pan et al., 2002]. None allows for specifying the logic of a complete wrapper, since they just provide the foundations for EXTRACTOR and SEQUENCE activities.

The proposals that address the problem of building complete wrappers can be divided into two categories:

– Proposals that provide specific-purpose languages and require programming skills, cf. [Hammer et al., 1997] [Kistlera and Marais, 1998] or [Luque et al., 2002]. Our proposal has several advantages with respect to them: i) it allows to specify the logic of a wrapper in a graphical manner, which makes them easier to create and maintain

since the user is not required to have any programming skills; ii) it leverages current methods to navigate or extract information and encapsulates them, whereas the previous approaches rely on the user to accomplish these tasks.

– Other complementary higher-level proposals, cf. [Sahuguet and Azavant, 1999] [Doorenbos et al., 1997] [Baumgartner et al., 2001] or [Pan et al., 2002]. They do not require any programming skills, but they assume the query wrapper model, which we have proved not to be adequate in general, cf. Section 2.

Furthermore, there are a variety of industrial tools in this field. QL2 and NewBie, for instance, fall within the first category, cf. http://www.ql2.com and http://www.newbielabs.com. Another interesting tool is Dapper, which allows to create and share wrappers that fit the query wrapper model, cf. http://www.dapper.com. The Kapow Robomaker tool also uses a workflow approach for web automation, cf. http://www.openkapow.com; our approach, however, has a number of advantages: i) Robomaker does not encapsulate complex data extraction tasks into activities; the extraction of a list of data records requires an activity in the workflow to extract each record field, and optional attributes require bifurcations in the workflow, which usually leads to large workflows even for relatively simple tasks; furthermore, their model does not support example-based wrapper induction techniques; ii) Robomaker does not allow for reusable components; iii) Robomaker does not support user-defined exceptions. Other relevant commercial tools include Fetch and Lixto, cf. http://www.fetch.com and http://www.lixto.com; unfortunately, they are not available for download, and we could not compare them with our proposal.

6 Conclusions

This article describes a new graphical language for designing web automation applications. Our approach models each task as a high-level workflow composed of activities that leverage previous research proposals in automatic web navigation and web data extraction techniques, but allow for complex logic-control features such as branching or parallelism. Our language allows to create reusable components easily. Subprocesses that are common to several wrappers can be easily created and reused through workflow activities. Furthermore, we allow to create templates that implement frequent structural patterns at source-level. Reusable components also help ease the development process, since advanced users can create complex patterns reused by other users. Our language has been designed from the study of real-world web automation tasks, to ensure that it allows for the most important requirements.

References

[Aalst et al., 2003] Aalst, W., Hofstede, A., Kiepuszewski, B., and Barros, A. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison Wesley.

[Álvarez et al., 2008] Álvarez, M., Pan, A., Raposo, J., Bellas, F., and Cacheda, F. (2008). Extracting lists of data records from semi-structured web pages. *Data and Knowledge Engineering*, 64(2):491–509.

[Anupam et al., 2000] Anupam, V., Freire, J., Kumar, B., and Lieuwen, D. F. (2000). Automating web navigation with the WebVCR. *Computer Networks*, 33(1–6):503–517.

[Arasu and Garcia-Molina, 2003] Arasu, A. and Garcia-Molina, H. (2003). Extracting structured data from web pages. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 337–348.

[Baumgartner et al., 2001] Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Declarative information extraction. In *Proceedings of the 6th International Conference on Logic Programming and Non-monotonic Reasoning*, pages 21–41.

[Chang et al., 2006] Chang, C.-H., Kayed, M., Girgis, M., and Shaalan, K. (2006). A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428.

[Crescenzi and Mecca, 2004] Crescenzi, V. and Mecca, G. (2004). Automatic information extraction from large web sites. *Journal of the ACM*, 51(5):731–779.

[Doorenbos et al., 1997] Doorenbos, R., Etzioni, O., and Weld, D. (1997). A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the First International Conference on Autonomous Agents*, pages 39–48.

[Hammer et al., 1997] Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., and Vassalos, V. (1997). Template-based wrappers in the tsimmis system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 532–535.

[Kistlera and Marais, 1998] Kistlera, T. and Marais, H. (1998). WebL: A programming language for the Web. In *Proceedings of the 7th International World Wide Web Conference*, pages 259–270.

[Knoblock et al., 2000] Knoblock, C., Lerman, K., Minton, S., and Muslea, I. (2000). Accurately and reliably extracting data from the web: A Machine Learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41.

[Kushmerick, 2000] Kushmerick, N. (2000). Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68.

[Luque et al., 2002] Luque, V., Sánchez, L., Delgado, C., Breuer, P., and Gonzalo, M. (2002). Standards-based languages for programming web navigation assistants. In *Proceedings of the 5th IEEE International Workshop on Networked Appliances*, pages 70–75.

[Oasis, 2003] Oasis (2003). WS-BPEL: Web Services Business Process Execution Language. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[OMG, 2003] OMG (2003). BPMN: Business Process Modelling Notation. Available at http://www.bpmn.org.

[Pan et al., 2002] Pan, A., Raposo, J., Álvarez, M., Hidalgo, J., and Viña, A. (2002). Semi automatic wrapper-generation for commercial web sources. In *Proceedings of IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, pages 265–283.

[Sahuguet and Azavant, 1999] Sahuguet, A. and Azavant, F. (1999). Building light-weight wrappers for legacy web data sources using W4F. In *Proceedings of the 25th International Conference on Very Large Databases*, pages 738–741.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, 25(3):38–49.

[Zhai and Liu, 2006] Zhai, Y. and Liu, B. (2006). Structured data extraction from the web based on partial tree alignment. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1614–1628.

[Zhai and Liu, 2007] Zhai, Y. and Liu, B. (2007). Extracting web data using instance-based learning. In *Proceedings of the 16th International World Wide Web Conference*, pages 113–132.