Instructible Information Agents for Web Mining

Mathias Bauer, Dietmar Dengler & Gabriele Paul

DFKI

Stuhlsatzenhausweg 3

66123 Saarbrücken, Germany

{bauer,dengler,gpaul}@dfki.de

ABSTRACT

Information agents are intended to assist their users in locating *relevant* information in vast collections of documents like the WWW. In many cases, e.g., when trying to *integrate* pieces of information from previously unrelated sources, it is not sufficient to merely identify documents containing relevant data. Instead, information agents have to identify the interesting *portions* of these documents and make them available for further use. This paper deals with the problem of training an information agent to identify and extract interesting pieces of information from online documents.

Keywords

programming by demonstration, information agents, wrapper induction

INTRODUCTION

Information agents play an important role in the process of identifying and locating relevant documents in vast, at best semi-structured document collections like the WWW. Their tasks range from directing user queries to a number of online search engines to the filtering of search results or information streams according to the user's interest represented in a *user profile* (e.g. [9]). When trying to *integrate* pieces of information from previously unrelated sources, it is usually not sufficient to merely identify and retrieve the documents containing the desired piece of information. Instead, a *wrapper* is required that makes certain well-defined categories of information contained in a Web site available to the user or other reasoning services/information agents. Examples include online hotel servers from which a travel agent could receive information regarding name and address of a hotel, the prices of various room categories and so on [3,4].

To overcome the tedious task of hand-coding information extraction wrappers and updating them whenever the corresponding Web sites have been redesigned, a number of approaches to the automatic acquisition of wrappers based on inductive machine learning techniques were presented [5,8]. Here the user is required to *annotate* the documents under consideration, thus providing a set of examples to be used by a machine learning algorithm. This approach, however, either puts a significant workload on the user (who has to provide *several* examples of pieces of information she or he is interested in), or it produces very "unstable" wrappers in that even slight modifications of the corresponding document make the wrapper fail to identify the right portions. This is the case if the instances presented to the system do not constitute a *representative* set of examples of what a document provided by some information source might look like (or even both).

To overcome this deficiency we present an approach based on the paradigm of *programming by demonstration* (PBD) in which the system's capacity to parse and structure even incorrect HTML documents is complemented by the user's ability to (optically) identify salient features of a document that stand in a semantically meaningful relation to the relevant piece of information. Examples include headings announcing the contents of the following table or graphics used to direct the reader's attention to a particular point. Hope is that such relations will survive at least some of the potential modifications of such documents—like the inclusion of a commercial banner—and thus provide *robust* navigation aids to be used to address a particular document part and render additional examples or retraining unnecessary.

[2] already introduced the basic notions and procedures on how to train an information agent. The approach described there left the user all the freedom to explore the space of possible wrapper types for a particular piece of information to be extracted from a Web site. While this seemed to be an attractive feature of the system at first glance, it turned out to put a very high workload on the user as she had to make all the decisions on how to procede by herself (e.g. "Have I already provided a sufficient number of hints to produce a good wrapper?", "What hint might be most useful in the current situation?").

As a consequence, we now introduce a measure to quantify the *expected utility* of the various actions feasible at each stage of the training process that enables the system to suggest future steps in an unobtrusive way, thus carefully guiding the user through the training dialog.

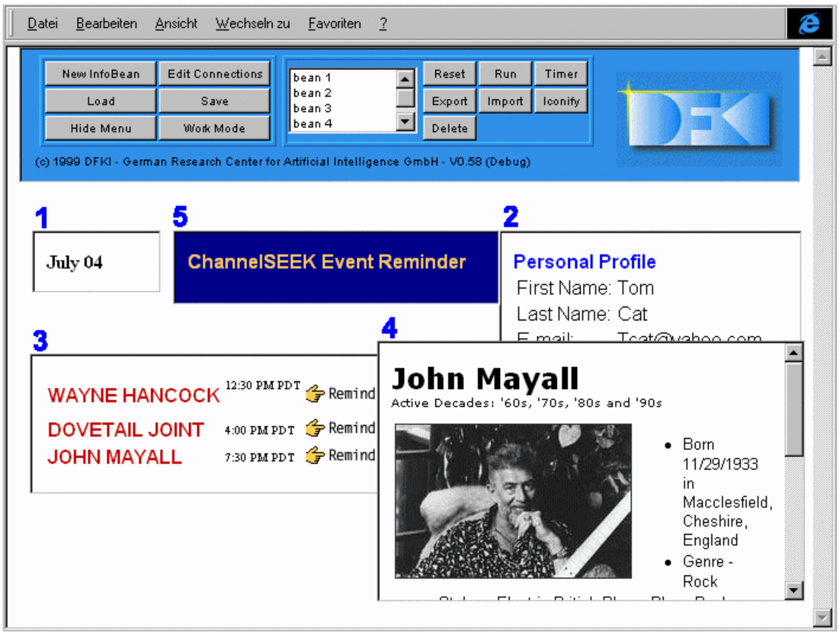

Figure 1: The "House of Blues" InfoBox.

The rest of this paper is organized as follows. The next section will describe one of the two scenarios in which the procedure sketched above will be used. Then the mechanism guiding the dialog between user and system is introduced at some length before a discussion of related and future work conclude this paper.

TASK DESCRIPTION

The InfoBean concept as introduced in [2] is an approach to the system-supported configuration of individualized information services. The common interface of the individual services is provided using a standard Web browser displaying a specific HTML page. Additionally, there is an InfoBean management service on client side providing Save/Load functionality w.r.t. InfoBean configurations as well as a proxy service in order to connect to the script evaluation engine. On server side this engine is implemented as a multi-threaded script interpreter on top of a script database.

Roughly speaking, an InfoBean represents a primitive information service. An InfoBox consisting of various inter-connected InfoBeans then is a complex service *integrating* information from previously unrelated sources where the flow of information is determined by the links drawn between its primitive constituents.

To be a little more precise, the term *InfoBean* describes a configurable component that either encapsulates an existing information service or specifies a process of information gathering and providing. Such a component communicates with its environment through channels that serve the purpose to pass information to or obtain data from other components.

The interplay of connected components thus defines an *information integration* network. The input channels of an InfoBean can be connected with output channels from various other InfoBeans that provide input data in an appropriate format.[1]

Figure 1 shows a screenshot of the browser interface where the user has configured an InfoBox to satisfy her interest in cybercast music events (see [2] to read more about the configuration of an InfoBox).

The InfoBox consists of five InfoBeans marked in the figure by numbers 1 to 5, respectively. Running InfoBean 1 provides the current date in its display area as well as an output channel (the channels are visualized in Figure 2). In the same way, InfoBean 2 provides some personal information of the user. InfoBean 3 is the essential part of the music event service. Using the date delivered by InfoBean 1 it searches on the "House of Blues Online" (HOB Online) server[2] for broadcast online events w.r.t. the given date. As output it produces a table with columns "artist", "time of event", and "reminder service input data". An "artist" value can fill the input channel of InfoBean 4,

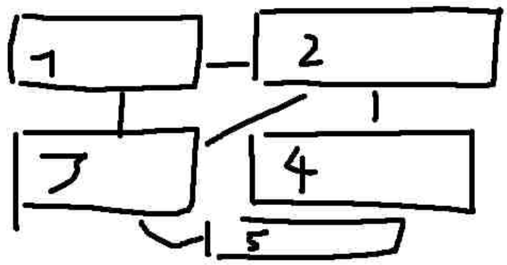

Figure 2: IO channels of the "House of Blues" InfoBox.

which searches on the "Ultimate Band List" server[3] to find an appropriate entry for the given artist and then selects only that biography field which the user has decided is of most interest for her. Finally, InfoBean 5 realizes an automatic reminder service on a chosen event (by clicking on a value in the third column of InfoBean 3) using information from the user's profile. It automatically fills the appropriate form at "HOB Online" with the effect of an email reminder to be sent to the user one hour before the broadcast event starts. The wrappers realizing the InfoBeans 3 to 5 have been built by the user in a cooperative dialog with the system supported by the

---

1 From a query planning point of view (see [1]), an InfoBean network can be used to visualize the query plan's execution.

2 www.hob.com

3 www.ubl.com

technique of “programming by demonstration”. Later on in this paper, we will demonstrate in detail a sample PBD-session aiming at building the wrapper for InfoBean 3.

HyQL (cf. [2]) is the language that is used to implement the procedures for all information processing tasks in our environment: selection and filtering of the contents to be displayed by an InfoBean and extraction of information to be passed to a specific channel. It is the language for operationalizing *wrappers* and thus will also be the target language for the training process.

We will give a brief characterization of HyQL and provide an example in order to demonstrate in which way it supports a clearly structured information gathering process at an appropriate abstraction level best suited for a cooperative synthesis of wrappers.

HyQL is an SQL–like WWW query language that supports flexible selection of document parts as well as navigation through the WWW by following hyperlinks. Its expressiveness allows document portions to be addressed in a variety of ways. The documents themselves are represented as parse tree structures in a canonical form, which means that some obvious faults in documents are repaired, optional start or end-tags are added, and additional annotations like “word” or “number” can be integrated. The basic operation model of HyQL is that all selection and filtering operations work on sets of trees which either represent whole documents or document parts.

Let's consider a sample script which is the part of the wrapper for InfoBean 3 with the subtask to “*find the resource for the construction of the online event table*”. The HyQL script consists of a sequence of five queries and reads as follows:

```
{ let content
  from document D in http://www.hob.com/live/ }
{ let info CONTEXT :=
      root,descendant(1,table)following(1,table)
  from document D
  where root,descendant(3,tr)(4,td)
        applicable to CONTEXT }
{ let info ROWS :=
            root,descendant(all,tr)
  from CONTEXT
  where root,descendant(2,td) applies to ROWS
        matches "*July*04*" }
{ let info URLS :=
        buildstring
          "http://www.hob.com/$pre.html"
          root,descendant(1,td)(1,a)href
  from ROWS }
{ select content
  from all document D1 in URLS }
```

The following table shows the conceptual structure behind the script above. The first query takes the URL of the "HOB online" home page as an external resource into account and the evaluation of its fetch-operation results in binding the variable `D` to the respective *document* object. Then, the second query takes this *document* object as its resource. It applies a filter in the context of the first `table` found in this document, which results in the selection of the next table in the documents object model which has at least a third row (`tr`) with four columns (`td`). This is a sufficient condition on the current “HOB online” home page to find the table with the event schedule. The resulting set containing this table is bound to the object variable `CONTEXT`.[4] The third query filters all the rows from this table whose respective second column matches with the given external resource "`date`" (Applied on the actual page this results in two rows, i.e. two `tr` objects). The first column element of each row contains a hyperlink from which an URL can be build in order to access a document where the relevant information provided by InfoBean 3 can be selected.

External source	*Abstract operation*	*Result objects / resources*	*Variable*
url	Fetch	{●} :document	D
	Filter	{●} :table	CONTEXT
date	Filter	{●, ●} :tr	ROWS
url pattern	Select/ Transform	{●, ●} :string	URLS
	Fetch	{●, ●} :document	D1

Now, we present the part of the wrapper which is used to select the name of the “artist” from the respective document. This name is used in order to fill the first column of the result provided by InfoBean 3 (see Figure 3, where a browser’s view of the relevant document is shown). In order to robustly identify the desired information item the wrapper characterizes the text to be selected using some salient features like style and color. This results first in the search for some "red text of size 3". Additionally, the script integrates into the search that the selected text is contained in a HTML "b" object and more importantly that it is in close relation to a specific graphical element in the respective document: the tabular’s data cell before the text’s cell contains an image (that with oval shape in Figure 3). Now, this image can be considered as a relevant hint in order to localize the selection. The two qualification conditions in the script below consider the constraints just mentioned.

```
{ let info X :=
     root,descendant(1,font,
                    {size=“3“ color=“red“})
  from document D1
  where root,child(1,b) applicable to X
        and
        here,ancestor(1,td)previous(1,td)
             descendant(1,img)
        applicable to X
        }
{ select text root,child(1,b) from X }
```

---

[4] The keyword **`let`** instead of **`select`** effects that the query result is not provided as a query output, but only available for internal processing during the script evaluation.

THE PBD DIALOG

The aim of a training session is to construct from one sample selection a HyQL wrapper that is as *robust* as possible, i.e. the wrapper should tolerate minor changes of the HTML page. To this end the user simply marks the construction of the wrapper itself —can be achieved either with a *passive* command-driven system through pull-down menus or with an *active* system that makes suggestions for the best actions to be taken next in a dialog window.

In the latter mode the system maintains in each step a list of ranked suggestions for possible next actions among which

Figure 3: Initial phase of the training dialog.

interesting part of the document using the mouse. Apart from identifying the information to be extracted by the wrapper, the user can therefore also give additional hints by pointing at relevant features of the selection, such as a specific style or format, or by identifying the selection as a concept defined in a given ontology[5]. Furthermore, the definition of a *context*, i.e. an area surrounding the part to be extracted, might be useful, if the context itself can be easily characterized and localized within the document (e.g. a table containing the highlighted portion of the document) as it reduces the search space for the localization of the selection. A *landmark* can also be used as a navigation aid or as an additional characterization of the selection (e.g. a graphics immediately preceding the highlighted text).

Depending on the user's choice a unique characterization of the relevant information—and thus, the effective

[5] This can be particularly useful in cases where certain syntactic restrictions exist, e.g. for concepts like zip code or time. In these cases the wrapper can be enhanced by an appropriate "type checking" mechanism that ascertains that the document portion to be extracted actually forms a proper instance of this concept (e.g. a zip code).

the user is free to choose. Online help for the short instructions is available through mouse-click. Thus, given a selection a *wrapper assessment* is computed taking into account different contexts or landmarks which the system considers useful (in the above sense of leading to a robust identification of the target concept). A *valuation function* then helps to determine what suggestion is best by computing a value representing the expected utility of the suggestion within the training dialog. This process is described below in more detail. Figure 3 depicts the PBD window for a session in which the wrapper for the selection of an artist in the context of the "House of Blues" InfoBean is constructed.

We assume that the user has marked the first occurrence of an artist in the given document and only describe at the end of this section how the constructed wrapper might be generalized for finding multiple occurrences. In the dialog window the system suggests as best choice the further characterization of the selection by additional hints.[6] This

[6] Note that the "S" button indicates that the system can make suggestions regarding the corresponding task,

facilitates the localization of the selected information in the document and makes it less dependent on structural information. But the user is also free to concentrate on a landmark or context first. At each point a dialog is used to fill knowledge gaps of the system and to give the user the control over the system by allowing her to override system suggestions.

Input Parameters

Apart from the HTML document, which is preprocessed into an HTML parse tree, the PBD system works with a *wrapper hierarchy*. This library consists of a number of wrapper classes where each class has associated some HyQL *template*, i.e., building blocks to form fully functional information extraction scripts, together with a description of the parameters yet to be instantiated. In addition, each wrapper class is given a numerical *intrinsic valuation* which is intended to represent the estimated robustness of this type of wrapper. For example a wrapper that simply counts all tags (HTML formatting instructions) of the document preceding the selection is more likely to fail than a more sophisticated wrapper that relates the selection to a surrounding table. Thus, the latter has a higher intrinsic valuation. A simplified part of the wrapper hierarchy is given in Figure 4.

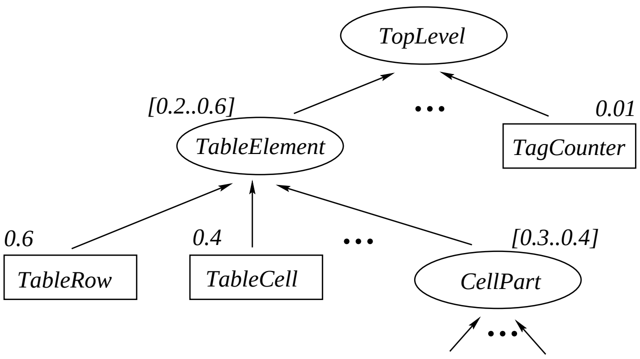

Figure 4: Part of the wrapper hierarchy.

The hierarchy also contains *abstract* wrappers such as *TableElement* whose valuation is an interval representing the valuations of all successive nodes. The applicability of each wrapper is determined by associated necessary conditions, i.e., for *TableElement* the selection has to be part of a table environment. This allows to prune complete parts of the hierarchy if the corresponding top-level conditions are violated. Choosing the hierarchy as a parameter allows for simple adjustments of the hierarchy by system-defined predicates and for a learning algorithm to improve intrinsic valuations. A function describing the user's skill (in terms of probabilities of correct answers to certain types of questions) is given as another input parameter to compute the utility of a suggestion. The system provides default values modeling naive or expert users. In addition, each wrapper can already be associated with a concept name, as shown in the example with *artist*, *time* and *reminder*, that can be used by the invoking system for storage or as reference to an ontology (see the section about the TrIAs scenario below).

General Wrapper Construction

Given a selection, a set of feasible wrapper classes is determined according to the hierarchy. This may lead to abstract wrappers and interval utilities. For each hypothetical wrapper the corresponding assessment is computed as outlined below in more detail. As the assessment takes into account the cost for localizing the selection, it also depends on the existence of a "good" context and/or landmark. Candidates for a context are, e.g., HTML tags surrounding the selection, default context is the complete document. A good context should in addition be "easy" to localize.

The system also looks for landmarks such as images, bold-face headings, text ending with a colon, special separators, table headings etc. Thus, each wrapper is assigned a set of assessments corresponding to the respective combination of context and landmark, where candidates not exceeding a certain localization threshold are discarded. Choosing a context corresponds to a recursive call of the system with the context given as selection. To avoid querying the user about all possible contexts and to provide a recursion end the context is characterized automatically. The user is also not asked for additional hints. This follows the supposition that multiple layers of selection, context and landmark do not add to the overall robustness, as possibly too much structural information of the document is used such that its modification is likely to affect this combination.

Given the assessments the system tries to determine the utility of possible queries or suggestions. These consist of suggestions for a better landmark and/or context to improve the assessment of the favored wrapper, queries to fill knowledge gaps to refine an abstract wrapper, further specifications of the selection, or disambiguating queries to decide between competing wrappers. At any time the user is allowed to take the initiative and give some hints to further specify the selection or point to relevant features. Such a hint may serve either as a landmark, i.e., it is incorporated into the wrapper to specify the path to the selection in the document or it may serve to further characterize the selection.

Example

Continuing the example from above we assume that the user wants to give a hint and chooses point 1 (characterize selection). She is offered to choose between style, color, a concept from the ontology, an image etc. The user then characterizes the selection as a "*red text of size 3*". Possible wrappers from the hierarchy include *TableCell*, *SpecialTag*, and *TagCounter*, which is the default wrapper as it is always possible to merely count all tags. Localizing the selection with the *SpecialTag* wrapper yields a HyQL script

---

whereas the "U" button allows the user to take over control.

that searches for the bold-faced text in the scope of the first occurrence of an HTML f*ont* formatting instruction with size 3 and red color (see the script given above). This should be reflected by a high assessment.

The computation of assessment and suggestions are repeated until one non-abstract wrapper is accepted by the user. With the collected data and the information from the hierarchy that contains general wrapper templates a HyQL script is constructed.

Wrapper Assessment and Query Utility

The assessment of a wrapper depends on

- the estimated robustness of the wrapper, i.e., the intrinsic valuation in the hierarchy,
- the robustness of the localization of the current selection in dependence of context and landmark,
- the acceptance by the user given by a function *Acc*.

Acc reflects the idea that a user-defined feature should be given higher priority than one suggested by the system that is merely accepted by the user. The assessment of a wrapper *W* with respect to a selection *S*, a context *C*, a document *D*, and a landmark *L* is given by

$$as(W,S,C,L) := \frac{\frac{I(S)}{L_{C,L}(S)} + \frac{I(C)}{L_{D,L}(C)} * Acc(C) + \frac{I(L)}{L_{C,D}(L)} * Acc(L)}{\# addends}$$

where $I(T)$ is the intrinsic valuation of T and $L_{X,Y}(T)$ is the localization of T with respect to concept X or Y depending on which is closer in the underlying document. The localization is defined as the number of necessary tag searches in the document. Context or landmark do not have to be defined. A hint is reflected in the assessment in two different ways. It either simply reduces the number of tag searches, as for the color and size hint above or it adds tag searches by relating other concepts to the selection such as an image. In the latter case the selection part of the assessment is given by

$$\frac{I(S)}{L_{C,L}(S)} + \frac{c}{L_S(H)} * Acc(H),$$

with a constant value c for hints and the localization $L_S(H)$ of the hint with respect to the selection. This reflects that a hint given by the user should add to the overall assessment but only relative to the "overhead" it produces.

Example (continued)

The assessment for the *SpecialTag* wrapper with an assumed intrinsic valuation of 0.5 is 0.5 as $L(S)$ = 1 and no context or landmark are given. Adding as a context, e.g., the surrounding table leads to

$$as(SpecialTag, S, table, \{\}) = \frac{0.5 + \frac{0.6}{5}}{2} = 0.31$$

with the intrinsic valuation 0.6 of the table characterization and a localization cost of 5 for searching the 4th table in the body of the document. Considering *TableCell* the assessment results in

$$as(TableCell, S, \{\}, \{\}) = \frac{0.4}{8} = 0.05$$

for taking the 2nd cell in the 2nd row of the 4th table in the body of the document. Note that the table environment is not visible to the user.

Given these assessments the possible next actions are ranked depending on

- the expected use of an action for refining abstract wrappers or getting hints in dependence on the user's skill,
- the assessment of the wrapper, and
- the difficulty of the query.

Possible suggestions include the identification of a landmark or context, knowledge queries allowing abstract wrappers to be refined, further characterization of the selection, or the request to accept the best wrapper according to the current system belief. The valuation of a suggestion q is defined by

$$v(q) = (as(NewW) - as(cW)) * P(+) * diff(q),$$

where *NewW* and *cW* specify the parameters of the best wrapper that can be reached by the suggestion q and the currently favored non-abstract wrapper, respectively. Note that with *TagCounter* at least one non-abstract wrapper can be computed. $P(+)$ should express the chance that the user gives helpful answers and is at the moment a function depending on the user's skill. $diff(q)$ describes the difficulty of the query.

Example (continued)

The valuation for suggesting a context for SpecialTag is negative, i.e., $(0.31-0.5)*P(+)*diff(sugContext)$, if *0.31* is the best assignment for any considered context. This means that the system does not suggest a context, but the user still has the opportunity to do so.

Assume that the user wants to specify the image directly left from the selected text portion as a landmark. The system has to decide whether to use the landmark as a script part or as an additional hint that further characterizes the selection. The resulting assessments are

$as(SpecialTag,S,\{\},img)$ =

$$\frac{\frac{0.5}{1} + \frac{0.4}{9} * 1.5}{2} = 0.28$$

where the landmark valuation is *0.6* and the landmark is the first cell of the 2nd row of the 4th table in the body of the document, and $as(SpecialTag,S,\{\},\{\})$ =

$$\frac{0.5}{1} + \frac{0.2}{3} * 1.5 = 0.59$$

with a hint constant of *0.2*. If the user then finishes the dialog, the script given in the section describing the HyQL language is generated.

The constructed wrapper can be generalized to select multiple occurrences of the selected text with the special HyQL command *select sequence.* Next occurrences are then searched within the specified context. Thus, when generalizing from single to multiple occurrences the user should pay attention to selecting a context which contains all desired iterations, e.g., the default context.

THE TrIAs SCENARIO

A second application scenario for the generation of information extraction wrappers using PBD can be found in the context of the TrIAs system ("Trainable Information Assistants", see [2]). Here the user interacts with a Web-based application like a travel planning agent that is intended to create a schedule for a trip satisfying all of the constraints entered by the user. Whenever this agent has to make use of dynamic information to be found somewhere on the Web, it formulates an information request and hands it over to the *Information Broker* that has access to a huge database with annotated Web sites describing what kind of information can be found at a particular site and how to extract it. In cases of wrapper failures, the original information request and the HTML document currently under consideration are handed over to *the Information Extraction Trainer* that in turn initiates a PBD training dialog with the user in order to create a new wrapper and satisfy the information request. The basis of communication between all TrIAs components and the user is a formal *ontology* providing the set of formal concepts to be used for message exchange (in the example depicted in Figure 2 the concepts are artist, time, and reminder).

A PBD session in the TrIAs context should also take into account the annoyance of the user resulting from too many queries. An additional input is therefore an annoyance function *annoy* modeling the readiness of the user to work on the wrapper construction. The query valuation *v* then becomes

$$v'(q) := v(q) - annoy(\#q, user)$$

where *#q* is the number of queries or suggestions posed up to now. Thus, if the annoyance is high the system decides to prefer the best wrapper at hand instead of suggesting further improvements with context or landmark.

RELATED WORK

The work presented in this paper is in the line of [7] where the user tries to characterize pieces of text using abstract patterns and the learning system mainly keeps track of the user's actions in order to prune inconsistent alternatives from the search space and suggest possible characterizations of unknown text items.

[6] characterizes programming by demonstration as a particular instance of inductive machine learning. Here the prerequisite is for the user to provide a *number* of examples to be generalized by the system using a particular learning bias. That work focuses on the development of general, domain-independent generalization methods that allow PBD to be applied to virtually any application without relying on the existence of highly specialized heuristics to guide the learning process. In contrast to this, our approach tries to minimize the number of required user interactions. This is particularly useful in situations similar to the above mentioned TrIAs scenario where the system has to rely on the user's readiness to help the system to overcome its current problems. Nevertheless, future work will include the integration of inductive machine learning algorithms in order to deal more efficiently with situations in which the user is actually ready to provide more than one example.

[12] focuses on the learner's perspective in a PBD setting. The "felicity conditions" formulated there describe the desired behavior of the trainer in order to optimally structure the lessons to be taught and thus facilitate the learning task as much as possible. Again, scenarios like TrIAs require the system to also take into account the trainer's "felicity" during the training dialog. As a consequence the system has to find a balance between gathering enough information to produce a most robust wrapper and annoying the user by involving her/him in a lengthy training session.

In this aspect our approach is similar to the query generator described in [10]. There the system tries to identify the user's current plan by asking a minimal number of questions to disambiguate the set of actions taken by the user w.r.t. the underlying intentions. When determining what question to ask next the expected utility (in terms of information gain) takes into account the so-called "nuisance factor" representing the annoyance caused by a question that is obviously unrelated to the user's actual plan.

The way the system tries to conduct the training dialog is somewhat similar to the approach taken in COLLAGEN [11]. Based on an explicit task model the interface agent can suggest both the next action to be taken and who should be responsible for its execution. In this kind of mixed-initiative dialog the user can override the agent's suggestions at any time, thus keeping control over the whole system. The integration of a plan-recognition component making use of a task model library could be an interesting complement to the utility-based approach to conducting the dialog taken here in that it would facilitate the application to other domains by exchanging the underlying task model.

CONCLUSION AND FUTURE WORK

This paper presented an approach to the acquisition of procedural descriptions of information categories in terms of the Web query language HyQL. The *wrappers* taught by a human trainer can be used to locate and extract certain pieces of information from online documents. They form an important part of the action repertoire of information agents trying to find information that is potentially relevant to a user (according to her/his interest profile) or required by another system integrating data from previously unrelated information sources.

The expressive target language HyQL allows the formulation of very abstract (and thus, very robust) information extraction wrappers such that at least certain types of modifications do not affect their effectiveness, thus reducing the number of training sessions required. The utility-based dialog management tries to minimize the user's workload by suggesting seemingly "good" ways to advance the training dialog and taking into account her/his assumed degree of annoyance. All of these are means to make the training process as painless as possible to the user, thereby hopefully increasing her/his readiness to quickly help the system out of a problematic situation.

Besides the points already mentioned in the previous section, future work will include the empirical evaluation with different types of users and the development of a better method to estimate the quality or robustness of a wrapper based on statistical data about wrapper failures and the reasons for them.

ACKNOWLEDGMENTS

This work was supported by the German Ministry of Education, Research, Science, and Technology under grant ITW 9703 as part of the PAN project.

REFERENCES

1. Ambite, J.L. and Knoblock, C.A. Flexible and Scalable Query Planning in Distributed and Heterogeneous Environments, in *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (Pittsburgh, PA, 1998).
2. Bauer, M. and Dengler, D. InfoBeans - Configuration of Personalized Information Services, in *Proceedings of the International Conference on Intelligent User Interfaces* (Redondo Beach, CA, 1999), 153-156.
3. Bauer, M. and Dengler, D. TrIAs: Trainable Information Assistants for Cooperative Problem Solving, in *Proceedings of the 3rd Conference on Autonomous Agents* (Seattle, WA, May 1999), 260–267.
4. Knoblock C., Minton S., Ambite J., Ashish N., Modi P., Muslea I., Philpot A., and Tejada S.: Modeling Web Sources for Information Integration, in *Proceedings of Fifteenth National Conference on Artificial Intelligence* (Madison, WI, 1998).
5. Kushmerick, N., Weld, D., and Doorenbos, R.: Wrapper Induction for Information Extraction, in *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (Nagoya, Japan, 1997), 729-735.
6. Lau, T. and Weld, D.S. Programming by Demonstration: An Inductive Learning Formulation, in *Proceedings of the International Conference on Intelligent User Interfaces* (Redondo Beach, CA, January 1999), 145–152.
7. Liebermann, H. Training Agents to Recognize Text by Example, in *Proceedings of the 3rd Conference on Autonomous Agents* (Seattle, WA, May 1999), 116–122.
8. Muslea I., Minton S., Knoblock C. STALKER: Learning Extraction Rules for Semistructured, Web-based Information Sources, *AAAI-98 Workshop on "AI & INFORMATION INTEGRATION"* (Madison, WI, 1998).
9. Pazzani, M., Muramatsu, J., and Billsus, D. Syskill & Webert: Identifying Interesting Web Sites, in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI96)* (Portland, OR, August 1996), 54–61.
10. Raskutti, B. and Zukerman, I. Acquisition of Information to Determine a User's Plan, in *Proceedings of the 11th European Conference on Artificial Intelligence* (Amsterdam, August 1994), 28–32.
11. Rich, C. and Sidner, C.L. COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*, 8:315–350, 1998.
12. VanLehn, K. Learning one Subprocedure per Lesson. *Artificial Intelligence*, 31:1–40, 1987.