Communication between Trainer and Agent in Programming by Demonstration

Mathias Bauer, Dietmar Dengler, Gabriele Paul
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
Firstname.Lastname@dfki.de

Abstract

One of the most natural ways to acquire procedural knowledge is to watch somebody performing a number of actions and try to abstract this observed action sequence in order to make it applicable to a wider range of situations. This abstraction process can be significantly facilitated if the performing agent gives additional hints e.g. on the reasons for choosing a particular action or dealing with some particular object. Doing so requires both partners—the observer and the "trainer"—to share some common knowledge about the domain and the task at hand as well as a common language to exchange ideas about the current situation. This paper deals with the communication problem in cases where the observer is a software agent that is to be taught some procedure in a programming-by-demonstration context.

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. (Pazzani, Muramatsu, & Billsus 1996)). 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 (Bauer & Dengler 1999b; Knoblock *et al.* 1998).

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 (e.g. (Kushmerick, Weld, & Doorenbos 1997)). 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.

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.

(Bauer & Dengler 1999a) 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 proceed 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.

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

before the communication problem is analyzed and future directions are discussed.

Task Description

The InfoBean concept as introduced in (Bauer & Dengler 1999a) 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 interconnected 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.

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 (Bauer & Dengler 1999a) 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. 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 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, which searches on the "Ultimate Band List" server 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 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.

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.[1] To this end the user simply marks the interesting part of the document using the mouse. Apart from identifying the information to be extracted by the wrapper, the user can 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. 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 construction of the wrapper itself, can be achieved either in an automatic mode 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 the user is free to choose. 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.

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.

[1] HyQL is an SQL-like Web-query language that allows both navigation over the Web and within one single document. Being a very expressive language, HyQL allows document portions to be described at a rather high, abstract level which facilitates the task of producing robust wrappers. For an introduction to HyQL the reader is referred to `www.dfki.de/~dengler/hyqltutor.html`.

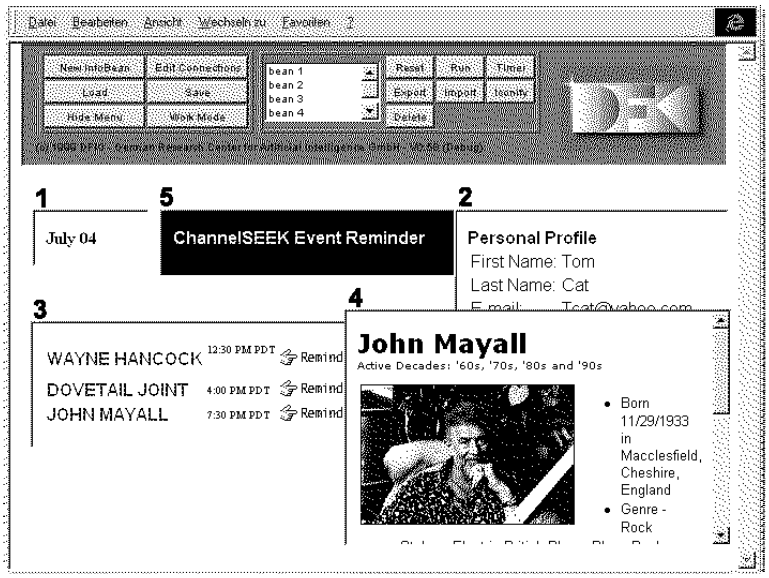

Figure 1: A sample InfoBox.

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.

General Wrapper Construction

Given a selection, a set of feasible wrapper classes is determined according to the hierarchy. For each hypothetical wrapper the corresponding assessment is computed taking into account

- the intrinsic assessments of the various HyQL templates to be used,
- heuristic assessments of the way these components will be combined,[2]
- the acceptance by the user.

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 an assessments taking into account 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 suggestions e.g. regarding a better landmark and/or context to improve the assessment of the favored wrapper, further specifications of the selection, or the termination of the dialog. 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.

To summarize, the training dialog starts with the user marking the desired portion of the document. Then the system suggests a number of actions that are expected to increase the estimated wrapper quality. The user accepts one of these suggestions or decides to do something completely different and so forth until a seemingly good wrapper was produced.

The Communication Problem

Effective communication between user (trainer) and agent (student) is hard to achieve. This is particularly true in scenarios like the one sketched above where an agent is to be trained to interact with an already existing application that was not designed to be "programmed" this way.

One of the most fundamental obstacles is the lack of

[2]This serves the purpose to prefer simple wrappers over more complicated ones, as the former are expected to be more robust. For example, the search for "the first bold-face word in the first table" is considered to be less fragile than the search for "the 245th bold-face word in the document", even if both produce the same outcome for the current document. See (Bauer, Dengler, & Paul 2000) for details.

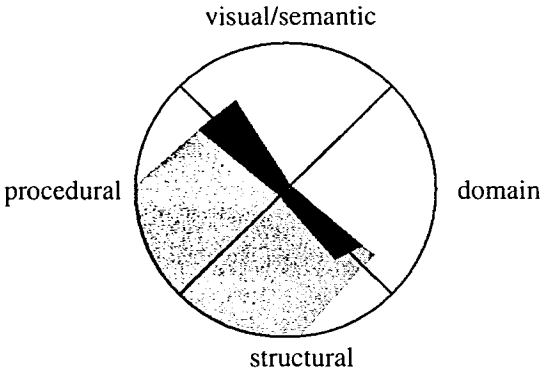

Figure 2: Knowledge shared by user and agent.

shared knowledge or insight into the various aspects of the training task at hand. As depicted in Figure 2, at least four different types of information play a key role during a training dialog.

- *Structural* knowledge refers to the internal properties of the programming domain (in this case, the HTML structure of a document to be processed).
- *Procedural* knowledge refers to the understanding of (at least) the basic concepts of the target programming language (in this case, HyQL) and the way the programs to be developed are intended to work (here, the principle functioning of wrappers).
- The *visual/semantic* category comprises the optical perception of the application system to be dealt with, the representation of domain objects and its interpretation allowing e.g. semantic relationships to be derived. Examples include the rendering of an HTML document in a browser and the identification of some particularly formatted portion of text as a heading describing the subsequent paragraph.
- Finally, *domain* knowledge includes the basic understanding of concepts from the application domain (e.g. about hotels, room categories etc.) and the wording typically used to describe them.

These various categories are not mutually independent. Rather, a lack of structural knowledge prevents a sufficiently deep insight into the procedural aspects of the task at hand; little or no domain knowledge exacerbates the interpretation of visual encoding of information beyond purely syntactical aspects etc. Unfortunately, only a relatively small part of this information is shared by user and agent and can thus be used for communication purposes.

As is depicted in Figure 2 the learning agent, for example, may possess deep insight into the (programming) domain *structure*. In the given example, this refers to the system's ability to evaluate and exploit the underlying document structure induced by the HTML code.

While the typical user can be expected to have some basic understanding of HTML at best, this overlap usually proved to be insufficient to explain the system behavior—its suggestions regarding next actions or wrapper components—referring to such structural document properties.[3]

The user's most prominent capability is in interpreting and understanding the *visual appearance* of a document in her Web browser. Doing so, she can easily identify "important", never-changing aspects of a document and *semantic* relationships among objects (e.g. a graphics and its title) by exploiting her background domain knowledge. The system, on the other hand, has to rely on a number of more or less feasible heuristics (such as "the first line of a table column typically contains a header describing its contents") in order to at least make a guess of which objects are related and might thus make a good navigation aid.

The coverage of *procedural* aspects of the training task—i.e. details of the target language and the general functioning of wrappers—largely depends on the user's experience in such training activities on the one hand and on the system's learning *bias* on the other hand.

Even without taking into account misconceptions on either side,[4] the above discussion indicates that an effective communication providing perfect mutual understanding of both partners is almost impossible.

What are the consequences for the training dialog in the InfoBeans scenario? Obviously there exist two almost disjoint, complementary *competence areas*. The learning agent suggests the next actions to be taken or wrapper components (e.g. a landmark) to be used mainly based on its understanding of the underlying HTML structure. The user—in her role as a trainer—evaluates these suggestions based on her domain knowledge and the visual impression of the document's rendering in the browser. Depending on the user's estimated expertise, either the agent autonomously decides on how to continue the training dialog—that is, the user is only presented the seemingly best action at each point in time—or the user is free to accept or ignore the agent's suggestions, taking over control by herself.

Adding image processing capabilities to the learning agent, as is done in (Amant *et al.* 2000) enables it to (at least partially) understand the structure and functionality hidden in the user interface. In terms of the above categories this means that the agent's coverage of the "visual" information portion is significantly extended. Equipped with this enhanced insight into the visual appearance of an application, the agent can try to immediately interpret and generalize the user's interaction with the various interface elements, thus enabling it to program applications without an API.

Applications specifically *designed* to be programmed by demonstration try to bridge the gap between user and system by providing a visual access to both the programming domain structure and the procedural aspects of the program to be developed. A typical example of this class is *Stagecast*

[3] The fact that the same visual rendering can be achieved using a number of different HTML encodings—just think of the many creative ways to use tables—aggravates this problem as it is almost impossible to correctly guess the actually used HTML code by just looking at its rendering.

[4] Some of the heuristics used by the system might be perfectly wrong, just as a user's guess of the document structure derived from its visual appearance.

(Smith, Cypher, & Tesler 2000), a system intended to enable its user to program a kind of video games. The world inhabited by various kinds of creatures has the structure of a (visually perceivable) grid, and program steps consist of simple state transition rules represented by the respective states before and after rule application.

Even here, however, not all interaction can be solely grounded on visual perception. Instead, the user must be willing to delve into some of the more advanced features of the system in order to describe what is called "hidden states" in (McDaniel 2000). This notion refers to additional aspects of the world that cannot be inferred from just one example, but have to be explicitly stated by the user or inferred by the system. In the Stagecast example, such additional information includes abstractions like "*any* kind of object should be in this place in order to make the rule applicable" or "variables" that represent the internal state of the characters.

This perspective is somewhat similar to the one taken in the Gamut project (McDaniel 2000) where the user is considered to be actually programming. That is, the user is in charge of conveying all required information to the system which in turn has to provide appropriate communication channels to facilitate this task for the user.

While this may be acceptable in applications like InfoBeans where the user is actively trying to establish a certain information service, this definitely differs from scenarios like TrIAs (Bauer & Dengler 1999b)[5] where the user initially did not intend to *program* a system, but to *use* it. As a consequence she cannot be expected to be infinitely patient and willing to invest her time and effort to teach the system something she expected it to already know. In other words, it's not (only) about making *learning* as easy as possible for the agent (compare (VanLehn 1987)), but also to facilitate the *teaching* process for the user. Thus the system's contribution must clearly exceed the simple check for structural properties of a document. In TrIAs this additional accomplishment is achieved by the library of numerically assessed wrapper components and the strong heuristics enabling the system to hypothesize semantic relationships among (graphical) objects.

Future Directions

From our perspective the key to improved user-agent communication lies in the extension of that knowledge that is shared between both partners (compare Figure 2). As it probably would not be acceptable to teach all potential users the details of the underlying programming languages and domain object structures, the only way to go is to extend the agent's capabilities such as to provide some common understanding—and thus, a common language—of what is going on during the training sessions.

At least the Internet Explorer provides means to localize (in a geometric sense) HTML structures within the rendering of a document. Using these values, it is not that difficult to implement a form of spatial reasoning such that agent and user can talk in terms like "the image to the left of the heading" etc.

[5] In this scenario the user helps a web-based application like a travel agent overcome difficulties in dealing with new or modified information sources.

Acknowledgment

We would like to thank Robert St. Amant whose "visual generalization" approach provided a lot of inspiration for our own future work.

References

Amant, R. S.; Lieberman, H.; Potter, R.; and Zettlemoyer, L. 2000. Visual Generalization in Programming by Example. In Lieberman (2000). to appear.

Bauer, M., and Dengler, D. 1999a. InfoBeans – Configuration of Personalized Information Services. In Maybury, M., ed., *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI '99)*, 153–156.

Bauer, M., and Dengler, D. 1999b. TrIAs: Trainable Information Assistants for Cooperative Problem Solving. In Etzioni, O., and Müller, J., eds., *Proceedings of the 1999 International Conference on Autonomous Agents (Agents'99)*, 260–267.

Bauer, M.; Dengler, D.; and Paul, G. 2000. Programming by Demonstration for Information Agents. In Lieberman (2000). to appear.

Knoblock, C.; Minton, S.; Ambite, J.; Ashish, N.; Modi, P.; Muslea, I.; Philpot, A.; and Tejada, S. 1998. Modeling Web Sources for Information Integration. In *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence.*

Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 729–735. Nagoya, Japan: Morgan Kaufmann Publishers.

Lieberman, H., ed. 2000. *Your Wish is My Command: Giving Users the Power to Instruct their Software* . Morgan Kaufmann Publishers.

McDaniel, R. 2000. Demonstrating the Hidden Features That Make an Application Work. In Lieberman (2000). to appear.

Pazzani, M.; Muramatsu, J.; and Billsus, D. 1996. Syskill & Webert: Identifying Interesting Web Sites. In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence*, 54–61.

Smith, D.; Cypher, A.; and Tesler, L. 2000. Novice Programming Comes of Age. In Lieberman (2000). to appear.

VanLehn, K. 1987. Learning one Subprocedure per Lesson. *Artificial Intelligence* 31:1–40.