Determining User Interface Semantics Using Communicating Agents

L. Ton and L.J.M. Rothkrantz

Knowledge Based Systems Group
Faculty of Information Technology and Systems, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
L.J.M.Rothkrantz@cs.tudelft.nl

Abstract. The Internet offers remote access to many information systems to users independent of time and location. This paper describes an agent based approach to deal with issues that rise from the differences between user interfaces to functionally similar information systems. The approach yields agents that interact with user interfaces to extract the model of the underlying information system. The model is represented by labeling input fields of the interface with their meaning. The agents are modeled after human dialogue behavior and human computer interaction. The interaction agents are capable of independently and robustly querying the information system without explicit instructions. Two types of agents are implemented and tested on European sites of rail travel planners. The agent design and test results will be reported.

1 Introduction

Information systems are often available independent of where a user is and what time it is. Remote access to such information services, independent of time and location, is often facilitated by information providers through offering their service via the Internet. Well known examples of such services are rail travel planners. Almost all European rail companies have made their schedules available on this medium.

The web pages that give access to the rail planners all offer the same functionality, but all interfaces differ somewhat. There is no “de facto” standard for the look and feel of this kind of interface. These differences make offering information services in an alternative way through e.g. wireless devices or through a speech interface more difficult than necessary. Also creating a system where multiple similar services are integrated is hindered.

This paper is about bridging that gap by using agent technology to assist with creating a unified interface to similar systems. The basic idea is that agents start an interaction with WWW-sites. This dialogue is aimed to result in knowledge about the underlying structure. Our agent is modeled after human behavior (see [1,2]).

2 User Interfaces

Many information systems can be accessed in different ways. Some can be assessed as web pages on the Internet using a web browser, others can be accessed using a telephone, a

V. Matoušek et al. (Eds.): TSD 2001, LNAI 2166, pp. 349–356, 2001.

usable voice interface might have been defined in voiceML. Recently the wireless markup language (WML) has become popular. Information systems can thus have different multiple interfaces that each use different means of communication. To make it possible to automate access to similar systems without having to use different specifications and procedures for each different interface, we need a way to describe what information an interface requires to perform its task. Such a user interface description language will be described in the section 6. With formal methods a system can be described using a precise and semantically sound language. Geert de Haan in [3] describes a model to design user interfaces based on the knowledge a competent user likely has. The model is called ETAG (Extended Task-Action Grammar).

Describing the human computer interaction enables testing an implementation against its specification. Also functional requirements can be checked before implementation. Another reason for describing HCI is that it enables (formal) verification with e.g. temporal logic.

Great effort has been put in the HCI field of research to formalize the definition of user interfaces. This has enabled requirement testing and eased building correct interfaces. Also valuable work has been done on unifying interface description languages. An example is UIML (User Interface Markup Language).

3 Rail Planners

There are many services on the Internet that offer rail planners. In this section we will analyze these planners in order to make a classification based on the similarities and differences. An Internet rail planner is a web interface to a program with which train schedules can be searched. On top of this minimal service, many rail planners allow searching for the best connection between two locations or retrieving pricing information. A rail planner interface exists of several entry forms and result pages. All services use forms to request travel data such as departure station and travel time from a user. Most services have one of those pages. We'll call these pages start pages. If a user forgets to specify a station, enters an invalid date or otherwise offers invalid or incomplete data, most planners present a page that shows what was wrong. Such pages usually contain a form with which the errors can be corrected. These pages are called intermediate pages. If all data was entered correctly, one or more schedule pages are shown. These pages hold the result of the query and are therefore called result pages.

3.1 Visual Characteristics

A rail travel planner helps a user with creating a travel schema given a time and date, a departure station and an arrival station. Figure 1 shows one of the forms of the Dutch rail travel planner. The visual characteristics are all about how forms look, the order of fields and which fields are used for data entry. We can consider these input fields as the expressions for the interaction dialogue. We've found the following characteristics: Station Selection can be done by pull down list and by free text box. Time can be entered in a text field for hour (24) and minute each; in one text field, hour and minute separated by a colon (:) or in two pull down menus for hours (24) and minutes (per 15 minutes).

Fig. 1. Dutch rail travel planner

Dates can be entered by separate fields for year, month and day, and one field for these three values separated by a blank space or a dot (.) or a dash. In a left to right, top to bottom sense, many of the interfaces ask for departure station first, then for arrival station. Afterwards the date is asked for and finally time can be entered. All planners except the Portuguese planner, offer a toggle with which can be selected whether the specified time is a departure or an arrival time. The Portuguese planner does not allow for a time to be specified at all. In almost all cases, date and time are pre-filled with the current date and time. This might give the user a hint as to where what needs to be filled out. All planners make use of labels: un-editable text fields or images that indicate what needs to be filled in a nearby input field. Image labels can contain icons (little images depicting a certain meaning) and text. Recognizing such images is well beyond the scope of this paper. Text labels contain valuable indicative information, but are language dependent and irregularly placed and will not be used. Some planners offer special aides i.e. a “today” toggle so that no date needs to be entered. The Dutch planner even offers a “tomorrow” toggle.

3.2 Technical Characteristics

In the first web browsers, HTML forms were rather regular and well defined. But in order to meet user demands many additions to forms have been introduced. Because of this mechanically interpreting forms has become a challenge. Different sorts of scripting extensions were added, the use of multiple submit buttons or alternatives to those buttons (e.g. images) was enabled and sessions can be used now to easily divide forms over multiple pages. We have cornered a number of these additions to help us find out which techniques are needed to deal with HTML forms. Interacting with an information system is sometimes done more efficiently if the interaction is split up in several stages. In HTML different stages can be realized using multiple forms. None of the travel planners we checked use sessioning. After a user has filled out a form, he can submit it by pressing the submit button. Many interfaces now offer help with filling out forms. Access to this help

is sometimes offered through additional submit buttons. These buttons do not actually "submit" the form, but are used to navigate. These navigating buttons are difficult to distinguish from the real submit button. They only differ in label and name which are generally language dependent characteristics.

4 Interface Knowledge

When human users interact with a user interface they find out where to enter a date, how to enter a time etc. quite easily, even if they don't master the used language. We assume that this is because human users can recognize the layout of a page based on experience with other interfaces, and have common sense and knowledge of the information they need to enter. Users that possess the required skills and knowledge can learn how to deal with alternative interfaces to rail planners. Experience with other interfaces may have taught a user that form fields of a certain size usually indicate that the requested data has a length that suits it. One of our goals is to extract the meaning of the parts that an interface is made of, such as free entry fields, check boxes and pull down menus.

5 SMART

Our goal was to find the meaning of the HTML fields on an interface form. We can assume that this meaning cannot always be extracted from the form, e.g. in the case that labels and other explanatory material are given in an unknown language. To find out the meaning independent of the layout, coloring and labels of a form, we chose to interact with the interface. Such interaction is a process guided by rules. These rules allow us to assume certain meaning in a yet unknown interface. We can then test our assumptions and change them if necessary until we find one that is correct. SMART consists of an HTML Framework, a module for result evaluation, an agent and an interface between these three parts (see Figure 2). In section 8 we will talk about agents. Agents contain the logic to take decisions about how to deal with an HTML interface best.

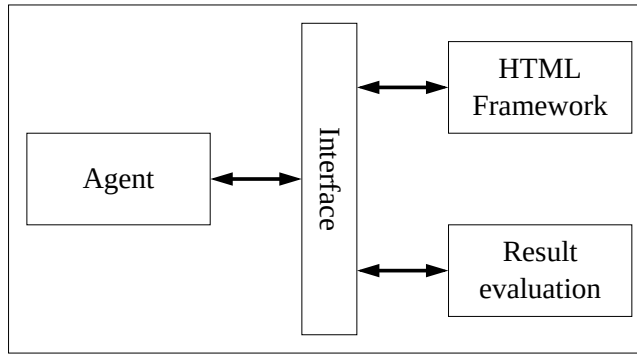

Fig. 2. Overview of the four parts in SMART

6 Interface Structures

To help extracting the meaning of the parts we will analyze the knowledge that is needed in this section. In order to use a web interface to a rail travel planner a user needs knowledge about traveling by train and skills for using web interfaces. In order to enable the use of this knowledge, a representation must be made to specify it and enable it. This section introduces such a representation for start pages. Start pages are described using Structures. Structures can express either exactly which input items a user interface has or they can define which combination of Structures describe this. The former type of Structure is called Primitive, the latter is called Combined.

With Primitive Structures we can specify which individual values have fields in a start page. Such a field could be a field for departure time, if the form has one field to enter time. It could also be a field for hour if the form has a separate field to enter a value for hour.

Combined Structures can be used to specify which field combinations exist on a form. An example combination is “time”. To fill out the time in a form a combination of two HTML fields for hour and minute are required. Such combinations can be specified as Combined Structures using the input and choice fields. In a Combined Structure it can be stated how many and which (sub-) Structures are expected. It can also be stated which alternatives are available. The HTML Framework can use this Combined Structure to search for the Primitive Structures for *year* and *yearshort*. The Primitive Structure for year was introduced in the previous paragraph as an input field with default value “2000”. The *yearshort* Structure could be similar but have “00” for a default value. Combined Structures can be also used to define combined values. A combined value is a value that is built up from more than one value. A form that requires such a combined value has more than one input field for the value. Time in many forms is a combined value - there can be fields for both minute and hour or for one for minute:hour and a selection for am/pm.

6.1 Configurations

A Structure defines which input is expected, while a Configuration shows how this input is expected. A Configuration is a list of Primitive Structures that each represent data that is to be entered somewhere in one field on an HTML form. Configurations are deduced from Structures by selecting and combining the options of the Structure. Creating a Configuration from a Combined Structure can be seen as a non deterministic rewriting system that rewrites the Combined Structure to a string of Primitive Structure using rules.

6.2 Assumptions

After a Configuration is selected, we have explicitly stated how many fields we are expecting, what type of fields these are and what their default values are. The next step to take is to find a suitable HTML field on a form for each of the Configuration parts. A mapping of these parts onto the fields of the HTML form is called an Assumption.

6.3 Leftovers

An Assumption need not to cover all HTML elements in a form. The elements that are left over after filling out a form using an Assumption are called Leftovers. Some interfaces may require that all elements are filled out however. Since the interface we are expecting should be matching our starting structure we can assume that Leftovers are irrelevant for the outcome so we use default values for Leftovers.

Summarizing, the steps to find out what fields on an HTML start page mean are the following: select a Start page, select a Start Structure, select a Configuration, select an Assumption, select a Leftover, submit the Start page and test the result.

7 HTML Framework

The HTML Framework was built in order to send an HTML form, extract forms from an HTML page, read structures, rewrite structures to configurations, combine a Configuration and a Form to an Assumption, combine an Assumption and a form to Leftovers, fill out a form based on a Leftover and Assumption. Our parser should be capable of parsing HTML Forms. The Java 2 v1.3 SDK offers a call back based HTML 3.2 Parser that was reused by rewriting the call back methods. The result evaluation module assu-mes that results are presented in tables. It also assumes that checks for existence of a structure representation and relative location of such representations are sufficient. Detailed information about the implementation is beyond the scope of this paper and will be published elsewhere.

8 Rail Planner Agents

In this section interaction with an HTML interface will be used as a means of determining which HTML fields represent what. Agents are used because they have advantage that they can do the interaction automatically. Two agents are designed.

The brute force agent will check all possible ways to fill out an HTML form based on a Structure. The steps this agent takes are the following. First a start page is selected. Then all configurations are selected one at a time. For each configuration all Leftovers are selected, one at a time. Finally all Configurations/Assumptions/Leftover combinations are selected and the result is tested until a stop condition is reached. For a real life information system such as a rail planner this agent is not very efficient. But if there is a solution and the tests are defined well, the solution will be found.

The push agent starts checking all possibilities, but tosses away Assumptions that are unlikely to be correct and sorts Configuration so that those that match best are tested. In the initialization phase, the HTML Framework is set up, the Structures being initialized, a Start page is selected and some variables are set. The Structure that describes the expected interface is chosen and its Configurations are calculated by the Framework. A flow diagram of the push agent is displayed in Figure 3.

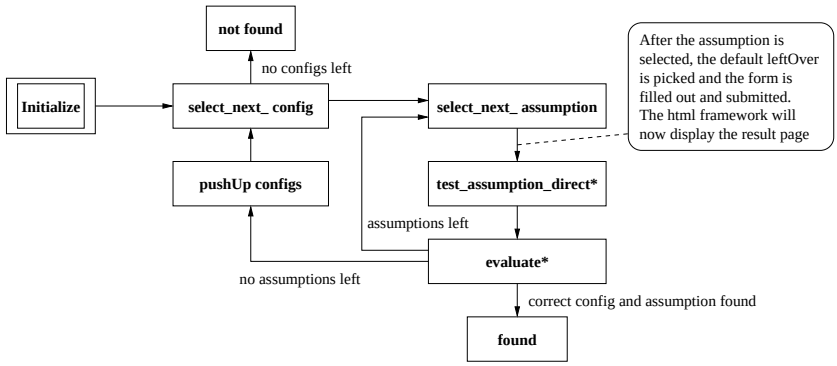

Fig. 3. Flow diagram of the push agent

9 Test

A testing application has been defined that request a date and time. The application is programmed as a Java Server Page. A JSP is a combination of HTML code and a Java program. The date is requested as two numbers, the time is also requested in two separate fields. From the analysis of the test run it proves that after several startups messages, the agents indicates it is searching for date_time. Afterwards it chooses the first Configuration that the HTML Framework offers for this Structure. It then calculates all possible Assumptions for the starting page. Each Assumption consists of field-value pairs. Then the Leftovers are constructed by the Framework. One of the found Leftover consists of the fields year, month and minute. They are filled with an empty value. After less then 20 rounds of testing Assumptions and pushing down unlikely candidates, the tested Assumption meets all the test. The agent concludes that time and date were found and the stop condition is reached. We tested our system on the Internet based rail travel planners from the Netherlands, Germany, Czech Republic, Italy, Portugal, Denmark and France. We note that our test, of date and time is language independent. It proves that our agent found the requested information without explicit knowledge of the language of the 7 countries and structure of the Internet pages.

10 Summary

In this paper a system was described that realized this by extracting interface semantics through determining the meaning of input fields of interfaces. In order to make the extraction of semantics feasible we have restricted our research to Internet based rail travel planners. A corpus of seven rail planners was created. We modeled human computer interaction based on the analysis of these planners. A notation that can be used to describe start pages based on their functionality was defined. A framework to use the start page description language to fill out a start page form and to retrieve the results was created. A testing module for testing result pages was implemented. Two example agents for extracting user interface semantics were developed. These agents are programmed as

expert systems that have rules to take action based on intermediate and result pages. A testing environment was created.

References

1. Hoppenbrouwers, J., et al.: NL structures and conceptual modeling: Grammalizing for KISS, Special Issue of the Journal of Data and knowledge Engineering (1997)
2. Nwana, H.: Software agents: An overview, The Knowledge engineering Review (1996)
3. de Haan, G.: Extended task-action grammar, PhD dissertation, Eindhoven Technical University, ipo, 2000.