Designing Intelligent Agents to Support Universal Accessibility of E-Commerce Services

E. Pontelli and T.C. Son
Knowledge representation, Logic, and Advanced Programming Laboratory
Department of Computer Science
MSC CS, PO Box 30001
New Mexico State University
Las Cruces, NM 88003, USA
{epontell*, tson}@cs.nmsu.edu

Abstract

We propose the design of an intelligent agent to improve accessibility of E-commerce applications and Web sites for visually impaired individuals. An important feature of this design is the application of knowledge representation technology – specifically conceptual structures – to explicitly capture and represent the *navigational semantics* of each document. This allows one to view the process of navigating a complex HTML document – e.g., a document containing tables and complex forms – as solving *goals* with respect to an *action theory*, i.e., as a *planning problem*. This information will be used by agents to provide a level of intelligence in the navigation process, allow users to express high level navigation goals, solve such goals using *planning techniques*. Effectively, the navigation of the complex components of HTML documents will be done by agents, on behalf of the visually impaired user.

Keywords: Universal Accessibility, Agents Design, Planning, Answer Set Programming, Universal Accessibility.

1. Introduction

1.1 E-Commerce and Universal Accessibility

The widespread advent of E-commerce and Web-based business services has lead over the last few years to a deepening of the “digital divide”. Accessibility to e-commerce services and resources in our information based society is considered an entitlement by the majority of the industrialized world. Through a reasonably simple series of keystrokes and mouse clicks we are able to access vast amounts of information and conduct a variety of commercial transactions from distant parts of the world. While the value of all the information and services available is sometimes questionable, it is there and it is *accessible*. Accessible that is, if you are not one of the many disabled members of our society. For many individuals in this group, access to these resources (e.g., to conduct business transactions) is difficult or impossible.

In the United States, landmark legislation has mandated equal access to the information world for all Americans. Through laws such as the 1990 Americans with Disabilities Act (ADA), the 1996 Telecommunications Act and the Section 508 Amendment to the 1993 Rehabilitation Act, the disabled are guaranteed accessibility “where reasonably achievable”. These three words belie the problem. In the domains of hardware or architectural modifications, the problems, for the most part, are understandable and achievable. In the domain of computer software however the situation is somewhat different. The areas of system interfaces and web access highlight this problem.

Our research fits within the overall umbrella of *Universal Accessibility*. The basic premise of universal accessibility of the Internet is the presence of alternatives to make Internet and Web-based services accessible and applicable to general users with different backgrounds, learning styles, abilities, and disabilities in widely varied learning and commercial contexts. The “universal” in universal design does not imply *one* optimal solution for everyone, but rather it underscores the need for customizable content and presentation media. Research in the field of universal accessibility has moved along three orthogonal threads: tools in cognitive sciences, multimedia presentation tools, and network technologies. In order to reach users with diverse backgrounds, interests, abilities and disabilities, digital material should be flexible along different dimensions: providing alternative representation modes, multiple options for expression and control, and different options for engagement.

In this project we are particularly concerned with tackling these three dimensions with respect to universal accessibility for the visually impaired. The group of visually impaired individuals is becoming quite large, since with increased life expectancy, more and more senior citizens are joining this group. Furthermore, visually impaired users are more severely affected by the current state of things. While the World Wide Web has rapidly become an enabling technology for e-commerce and B2B applications, the visual nature of existing Web browsers and electronic business tools, and the reliance on graphical and multidimensional presentation of information (e.g., images, tables) has deepened the divide with visually impaired users. Even the rapid development of XML-based encoding of business knowledge (e.g., in BPML), with its emphasis on the separation between *content* and *presentation*, has not helped – as style sheets and other commonly used presentation media are visual-based and embed knowledge elements (e.g., layouts) absent from the content and necessary for effective access and navigation of the content.

1.2 Related Efforts

Various solutions have been proposed to overcome some of these problems. Legislatures (e.g., the ADA and the Telecommunications Act) have imposed the adoption of design practices to make Web pages universally accessible, and initiatives such as the W3C Web Accessibility Guidelines have tried to put these directions in practice. Similar efforts have emerged also outside of the U.S. – e.g., as demonstrated by the recent working documents by the Australian Human Rights and Equal Opportunity Commission on web-accessibility for e-commerce reference.

In spite of this, a substantial part of the Web is still not designed according to these criteria – even popular Web design tools do not enforce such guidelines. For this reason, in recent years a large number of proposals have emerged, aimed at providing accessibility at the level of the final user – i.e., providing the final user with tools to improve accessibility of existing documents and sites. This solution is orthogonal to the application of design guidelines.

The initial proposals based on the use of standard screen readers (e.g., Jaws [24]) in association with traditional Web browsers had limited success – information on the Web is highly context sensitive and highly structured, and only ad-hoc screen readers are capable of taking advantage of this context-sensitivity and adapt their behavior to the structure of the document. Even fewer tools are capable of modifying their behavior to respond also to the desires and capabilities of the end-users.

The most notable outcome of this thread of research is represented by the design of aural Web browsers; in these tools the screen reader is an integral part of a browser and obtains direct access to the internal structure of the documents recovered from the Web. The knowledge of the structure of the documents is employed in novel ways, to drive the generation of an accessible version of the document, e.g.:

- providing a top-level presentation of the main document titles and headings [13]
- creating summaries of the document [23]
- restructuring the document to place emphasis on its main components [1]
- using sound cues to identify the different components of the document [11]

1.3 Relevance to E-Commerce

In this paper we concentrate on the development of autonomous agents that assist visually-impaired users in navigating complex documents – thus directly tackling the central problems highlighted in recent surveys about impact of e-commerce on people with disabilities [25]. This is not the only application of the techniques developed herein. With the development of Web Based Intermediary (WBI), this type of agent can be directly incorporated into a Web-Browser to assist visually-impaired users in navigating the Web. Such a browser is not only useful for visually-impaired users but also for normal users. For example, this type of browser can be used as a navigation-assistant, that allows users to access business information on the road (e.g., while driving). It can also be used as a personal assistant who reads the current stock information from a mobile phone or PDA. In general, this type of browser is most appropriate in applications where visual-display is undesirable or impossible. When the Semantic Web [4] is in place, the techniques used in developing the navigation agents can be employed in developing navigation agents that can discover services (e.g., navigation strategies for specific documents), invoke services, or combine existing services to create new services (e.g., coordinate access to different documents).

2. The CAVIN Project

The CAVIN project – Computer-Aided Visually Impaired users in Internet Navigation – is a comprehensive effort developed over the last couple of years at New Mexico State University. In the current work, we have focused on improving the accessibility of the World Wide Web for blind and severely visually impaired in general, but more specifically in the educational setting. In this work, we face two basic problem areas: the accessibility offered by the popular web browsers and the effect of the design of the actual web pages. In the former, we are not interested in making changes to the web browser's code, while in the latter, we are constrained to making accessibility recommendations to web page designers.

Given these constraints, our project has identified three of the most difficult areas of web access through current browsers: tables, frames, and XML fragments. While other groups are working on more general problem areas of web browsing [1,22,23] we have confined our research to these three aspects of the problem because they represent the most difficult to solve in a general way. The general reading of a web page by the visually impaired can be implemented by presenting a totally text-based version of the page, while this is not possible in the areas of tables, frames and XML [9,13,7]. Text is linear in nature while tables, frames and XML are inherently non-linear and multidimensional, and their structure and layout define much of their semantic content. These are also the core problem areas in the accessibility of e-commerce Web services.

The problem is broad in scope for the visually impaired. Lack of access to these structures has a negative impact not only on everyday life decision making, but also on access to educational and business materials which are now being presented in larger measure via the web, leading to another instance of the “digital divide”.

In the last year we have initiated a comprehensive project aimed at improving accessibility of those HTML components that are inherently multi-dimensional, such as tables, frames, and forms. Preliminary results from this work have been previously presented [12,16,17]. The key behind our original proposal is the adoption of knowledge representation methodologies to capture the *navigational semantics* of complex HTML structures. The navigational semantics complements the traditional semantic description of a document; the navigational semantics is aimed at capturing those aspects of the semantics of the document that a visually-able individual gains from being able to see the layout of the information on the screen. Cognitive studies [16] have indicated that the lack of visual cues significantly decreases the ability to retrieve, understand, and use information in tabular and multi-dimensional form. Thus, these results show a disadvantage for table users when they are given information in a procedure that simulates the flow of information to blind users using a screen reader. Not only that, but the cognitive studies have also indicated that conveying this missing knowledge is not straightforward: providing a table user with additional structural cues was helpful only when the cues were *spatial* in nature; adding auditory cues made learning performance worse [16].

The presence of an explicit representation of navigational knowledge allows us to reduce the process of navigating a document to the process of *reasoning* on a formally represented body of knowledge. In our previous work [16,17], the reasoning process was limited to the ability of *traversing* an abstract representation of the document – in the form of an annotated graph structure. The navigation process is directed by the final user and expressed as a *sequence of commands* in a properly designed language – a *Domain Specific Language (DSL)* designed to describe traversals of semantic descriptions. The graph structure derived from each document provides the *semantics* for the DSL. Our solution is novel in the sense that it provides the ability to customize the navigation process to the semantics of each specific HTML component. Furthermore, the use of a DSL for the navigation process offers the ability to dynamically reconfigure the semantic description (e.g., to add new components to the semantic description – used, for example, to allow the user to “keep a finger” on the table) and provides designers of the semantic description with the ability to provide *scripts* to direct the navigation of a specific document component. The advantage is that the DSL reduces the management of a semantic description to an explicit procedure, which can be reused and parameterized to accommodate orthogonal changes in documents' content (e.g., dynamic pages) [17].

3. Building Accessibility Agents

The experience acquired in the CAVIN project has prompted us to move our project to the next level. The ultimate objective is to develop a software agent that can represent a visually-disabled user in the process of navigating complex documents obtained from the Web – i.e., an *Accessibility Agent*. The accessibility agents are aimed at providing the user with the navigation capabilities that diversify him/her from visually-able users – i.e., the agent will provide the user with a pair of “intelligent eyes”. The basic design of our accessibility agent builds on the work previously conducted in the CAVIN project. Accessibility agents provide visually impaired users with the ability to interactively navigate complex HTML documents, producing a dynamic aural presentation. The navigation process is conducted at a very high level of abstraction – the impaired user provides his agent with *goals* that he intends to accomplish from the navigation process. Goals range from simple requests to move inside document's components – e.g., move from one cell of a table to another – to more complex queries requiring sophisticated reasoning about the content of the document – e.g., determine the collection of table cells containing expenses occurred on June 15th. The task of the agent is to interface the user with the World Wide Web and conduct the reasoning steps required to accomplish the requested goals.

Figure 1 illustrates the overall structure of the accessibility agent. The agent represents the user in obtaining documents from the Web and in presenting the documents in a non-visual fashion. Documents retrieved by the agent are analyzed and a semantic representation constructed (or retrieved from a local knowledge base); the semantic representation will be used by the agent as the abstract semantics for the document and used to answer user's queries. Interaction between the user and the agent is based on *navigation goals*; these are expressed and interpreted in the context of an *action theory* [8]. This allows the user to describe the objective of his/her navigation as a goal and let automatic mechanisms (i.e., a *planner*) develop (part of) the navigation process. Furthermore, this approach allows the semantic description to predefine not only complete navigation strategies (as in our previous approach based on DSL) but also partial skeletons, making the remaining part of the navigation dependent on run-time factors, e.g., user's goals, specific aspects of the table's content, user's run-time decisions.

We will start by presenting a formalization of the problem we are focusing on. Our goal is to provide non-visual accessibility to particular classes of HTML structures. In particular, we are currently focusing on handling HTML Tables with their inherent two (or more) dimensional structure. Efforts in this direction can be easily customized to work with other non-linear HTML components, such as frames and forms [16]. Each table has a visual multi-dimensional layout, and in most cases such visual layout is essential component of the semantics of the table. Thus, any form of arbitrary linearization will deprive the non-visual user of an important key in understanding the document. In addition, in a variety of situations the visual layout is actually used to emphasize the existence of a more complex structuring of the information within the table – e.g., the spatial layout can be used to describe a 3-dimensional structure or even a tree structure (e.g., by properly combining the table structure with the use of different colors, header elements, etc.).

4. Semantic Navigation

4.1 Semantic Representation

As we mentioned earlier, in our proposal we make use of knowledge representation techniques to capture the *navigation semantics* of a table. The knowledge representation scheme that we have adopted in this project is based on *Conceptual Graphs* [21]. A conceptual graph can have two kinds of node; a concept node that represents types and objects of that type and a relation node that represents a relationship between these objects. The theory allows for a basic expressiveness that is equivalent to first-order logic as well as mechanisms for defining concepts and for representing type hierarchies of concepts and relations. Much research since Sowa's original conception has extended the formalism to allow representation and manipulation of more advanced ontologies, especially those involving actions and events, and higher level structures such as viewpoints [19], and nested contexts. Many of these are difficult to express in linear first-order logic, but have simple, easy-to-read forms as conceptual graphs.

There are two kinds of structure present in any web page – syntactic and semantic. The syntactic structure is hierarchical, as reflected in the use of HTML/XML; the semantic structure may generate a more complex structure (i.e., it is not necessarily expressible as a simple hierarchy). Before XML became popular, the only way to navigate a web page was to follow this syntactic structure. Any other navigation technique was forced to rely on search through content, especially where no meta-data was present. Since the advent of XML, however, representation of meta-data is far easier, since a web page can store arbitrary amounts of content that can be considered as additional to the main content of the document. It has been suggested that this extra meta-data content can be used for web page retrieval, but it can also clearly be used for web page navigation. It is this latter use that we are proposing as semantic navigation. Since the content of this meta-data is arbitrary, links between meta-data items and between meta-data and the main content of the page cut across the syntactic structure of the page. These links have to be represented separately from the page itself, possibly in an associated document, just as display information can be placed in a separate style sheet. We are proposing to use conceptual graphs for the representation of the semantic links, and we have started work in this direction. The creation of a semantic description is a complex problem and we have proposed different approaches to tackle it, such as:

1. syntactic analysis of the table – e.g., identification of specialized HTML tags and attributes for table structuring, such as `scope` and `axis`;

2. manual annotations – using a specialized GUI to develop fragments of a semantic representation for existing documents [16];
3. machine learning techniques – e.g., use of semantic clustering algorithms to detect patterns in previous navigations of the document. Details of these methods are described in [16].

In this paper we will often refer to the following simple example to illustrate the technology developed. One of the major obstacles encountered by visually impaired users in accessing Web sites is represented by *tables* [15,9], where it is harder to keep track of the context without additional contextual references (e.g., continuous knowledge of relevant row and column headers). Table 1 was drawn from the Italian National Railway system Web site and it displays some of the options for traveling from the city of Udine to the city of Genova.

The table is fairly simple for a visually-able user: groups of rows are meant to be read together to describe possible itineraries. This kind of table becomes virtually unintelligible when accessed through existing aural Web browsers – either using a standard browser in conjunction with a screen reader (e.g., the popular JAWS for Windows screen reader [24]) or using specialized aural browsers (e.g., IBM's HomePage Reader [1] and BrookesTalk [23]) because these approaches lead to a *linearization* of the table – the table is sequentialized and reproduced as a sequence of rows. Visually-impaired users would have extremely hard time to relate the content of a row to a particular itinerary. This example demonstrates that linearization methods are either very unsuitable to present this sort of tables, or they have to be *customized* to create a linearization that is meaningful to express the semantic of the document – e.g., in this example, a linearization should clearly separate the different itineraries, by first presenting the summary of their arrival/destinations before proceeding in the details of the connections – reinforcing the context at the beginning of each new itinerary.

Figure 2 presents a part of the conceptual graph representing the information in table 1. With this conceptual graph, we can represent to the users different views of the table. For example, the information inside the dotted line rectangle represents the first itinerary. Thus, if the user is not interested in the first itinerary, this part of the graph can be removed. On the other hand, if the user is interested in this particular itinerary, this is the only information he needs. Traversing the path consisting of the nodes labeled "Itineraries", "Departure", "21:24", "Arrival", and "08:42" is equivalent to reading the first line of the table. *This shows that the navigation of the document could be viewed as an appropriate traversal of (parts of) such conceptual graph.*

4.2 Language-based Navigation

In our previous proposal [12,17] we developed a *Domain Specific Language (DSL)* to support the navigation of tables (later extended to support frames and forms [16]). The commands of the DSL are parametric on the navigational semantics of the table – i.e., the behavior of each command will change accordingly to the table currently under consideration. The main data domain manipulated by the DSL programs is the set of *nodes* of the table representation – i.e., either locations of the HTML table or elements representing abstraction levels; for the sake of discussion, we will assume that each node has a unique identifier associated to it. Such identifier can be used to retrieve a description of the node (if it represents a level of abstraction) or an index to a table storing the actual cell content. Certain operations also manipulate *sets* of items – where the items are themselves nodes of the table representation. Sets are used to represent operations which are nondeterministic in nature – i.e., they do not identify a unique node of the representation. This is the case of operations like `abstract` (move to a higher level of abstraction in the SD) – since each node can be abstracted in different ways whenever there are different viewpoints.

The DSL includes various classes of instructions. *Inner Navigation Instructions* are used to navigate *within* one abstraction layer; these instructions allow the user to follow the inner links in the table representation. *Outer Navigation Instructions* allow the navigator to move between different abstraction levels. Due to the presence of different viewpoints, these operations may require specification of the particular viewpoint that the user is interested in. *Extension Instructions* are used to extend the table representation on the fly; the user can make use of these instructions to create additional viewpoints or keep track of previously accessed information – e.g., these instructions allows the user to “keep a finger” on previously encountered positions. *Inspector Instructions* are used to query the current position in the table representation during navigation; e.g., the `path` command determines the path from the root of the hierarchy to the current node – this is particular useful to provide contextual reinforcement during the navigation of the document.

The DSL has been specified and an interpreter for it has been (automatically) derived using *Horn-logic Denotations* [10] – a logic programming software engineering methodology for specification and implementation of DSLs.

5. An Action Theory Perspective

To understand a table[1], even when he is familiar with its content, a visually-impaired user has no choice but examining all its cells. Any language that supports the process of understanding a table needs to be designed to help the users to navigate in the table, perhaps at different levels of abstraction. It also needs to provide a simple way for users to find some specific information, i.e., to locate a specific cell with the content satisfying certain criteria. We argue that all of these features can be supported under an *action theory framework* [8]. Under this perspective, a visual-impaired user could be viewed as an agent that tries to explore an unknown or a partially known environment. The agent has the capability of moving around (his finger is moving from location to location) and can comprehend what is in the cell[2]. Gradually, together with his/her general knowledge about the world, our agent will arrive to conclusions that are useful to him. For instance, he/she would conclude that Table 1 contains train itineraries and, for example, he/she will try to determine if any of the itineraries is suitable to him/her.

In the action theory framework, the results of the navigation process will be described by a set of properties describing the environment, called *fluents*. The commands that allow user to navigate between table cells are called *actions*. Actions can be divided into two types, *basic actions* and *table-specialized actions*. The former is similar to that of moving the finger around the table, thus providing the basic navigation capabilities to users. Observe that the structure of the basic action is table *independent*. The latter is, in many cases, a carefully developed procedure that helps users to understand the table without

---

[1] Hereafter, the discussion is focused on tables, but the techniques developed for tables can be easily generalized for arbitrary document components.

[2] How this can be done is an interesting topic and deserves an in-depth study that is not the main topic of this paper.

having to navigate through the whole table. This is because the representation of tables (and many other similar structures, e.g., frames and forms) is highly domain-dependent. Tables containing similar information are usually organized using similar formats and, hence, allow the development of table-specific actions, that can be reused in other tables of the same domain.

5.1 Data Definition Language

The *data definition language (DDL)* is used by domain specifiers to provide the basics needed to encode the structure of a table and its content. The DDL programs, in the new framework, represent an encoding of the conceptual graph representing the navigational semantics of the table. The DDL includes the specification of fluents and actions, and any objects that will be used in describing the action theory. This step provides the basic ontologies for constructing the action theory of the table. Although a table's structure and content vary considerably, the basic navigation actions do not. This leads us to the basic ontology for describing the basic navigation actions. From our discussion in the previous section, we first associate each node of the conceptual graph with a unique *identifier*. From now on, we will use *Id* (possibly with indexes) to denote a cell identifier and whenever we say that the user is at *Id* we mean that the finger of the user points at the cell associated with *Id*. To refer to the *current position* of the agent, we use the fluent *at(Id)*. The fact that *at(Id)* is true means that the agents is at *Id*. Nodes are connected with each other but not necessarily to all (according to the semantic description mentioned earlier). To represent this information, the fluent $connected(Id_1,Relation,Id_2)$ is used. The most simple navigation action in a table is that of moving from cell to cell, and it will be called $move(Id_1,Id_2)$. The fluent *info(Id)* is used to say that the content of *Id* has been presented to the user. Additional fluents can be provided to deal with the distinction between concept and relation nodes in the semantic representation, thus allowing to "type" the current location and the possible movements. For example, nodes of the conceptual graph might be divided into groups or hierarchies. We use the predicate *type(Id,type)* to record this information. We next describe the action language for the specification of action theories encoding tables.

Example: The action theory for Table 1 (via the structure in Figure 2) will have different nodes for each itinerary, its departure and arrival time, its connections, and departure time, arrival time, station, and train for each leg of the itinerary. We note that for the first itinerary, we need around 15 nodes. Some connections are

connected(Itineraries,Departure,21:34),
connected(Itineraries,Departure,05:36),
connected(21:34,Connections,Leg#1),
etc.

5.2 Action Theories

We use the high-level action description language ***B*** of [8] to represent action theories. In such a language, an action theory consists of two finite, disjoint sets of names **A** and **F**, called *actions* and *fluents*, respectively, and a set of propositions of the following form:

$$\textbf{caused}(\{p_1, \ldots , p_n\}, f) \tag{1}$$

$$\textbf{causes}(a,\ f, \{p_1, \ldots , p_n\}) \tag{2}$$

$$\textbf{executable}(a, \{p_1, \ldots , p_n\}) \tag{3}$$

$$\textbf{initially}(f) \tag{4}$$

where f and p_i's are fluent literals (a *fluent literal* is either a fluent g or its negation $\neg g$) and a is an action. A proposition of the form (1), or *static causal law* (also *causal law*, for short), states a constraint among the fluents of the domain. It states that whenever the fluent literals $p_1, \ldots ,p_n$ hold then f must hold also. A proposition of the form (2), referred to as a *dynamic causal law*, represents the (conditional) effect of *a*. It says that *f* is guaranteed to be true after the execution of *a* in any state of the world where $p_1, \ldots, p_n$ are true. An *executability condition* of *a* (a proposition of the form (3)) says that *a* is executable in a state in which $p_1, \ldots, p_n$ hold. Propositions of the form (4) are used to describe the initial state. It states that *f* holds in the initial state. E.g., to describe the effects of $move(Id_1,Id_2)$ and its executability condition, we write

$$\textbf{causes}(move(Id_1, Id_2), at(Id_2), \{\})$$
$$\textbf{causes}(move(Id_1, Id_2), \neg at(Id_1), \{\})$$
$$\textbf{executable}(move(Id_1,Id_2), \{connected(Id_1,\_,Id_2), at(Id_1)\})$$

The first two propositions say that moving from Id_1 to Id_2 causes the agent to be at Id_2 and no longer at Id_1. The last proposition states that the action is possible only if the two nodes are connected and the agent is at Id_1 prior to the execution of $move(Id_1, Id_2)$. To encode the fact that the agent cannot be at two places at the same time, we write

$$\textbf{causes}(at(Id_1), \neg at(Id_1)) \tag{5}$$

for every pair $Id_1 \neq Id_2$. To encode the assumption that agent can automatically understand the content of a node (or the aural output) whenever it visits the cell, we use

$$\textbf{caused}(\{at(Id)\}, info(Id)) \tag{6}$$

Finally, to say that the agent is at the node designated by 0 initially we write

$$\textbf{initially}(at(0)).$$

An *action theory* is a pair (D,Γ) where Γ, called the *initial state*, consists of propositions of the form (4) and D, called *the domain description*, consists of propositions of the form (1)-(3). For each conceptual graph *G*, with the set of identifies *ID*, let $G_{connected}$ be the set of fluents of the form $connected(Id_1,Relation,Id_2)$ such that Id_1 and Id_2 belong to *ID* and they are connected through the node *Relation* in the conceptual graph. Let $G_{actions}$ be the set of propositions describing the effects of the actions $move(Id_1,Id_2)$ for every pair of $Id_1 \neq Id_2$. Furthermore, let G_{causal} be the set of propositions of the form (5) for every pair $Id_1 \neq Id$ and (6) for every *Id*. The action theory corresponding to *G* is then given by the pair (D, Γ) where

$$D = G_{actions} \cup G_{causal}$$

and Γ consists of the propositions

$$initially(connected(Id_1,Relation,Id_2)),$$

one for each $connected(Id_1,Relation,Id_2)) \in G_{connected}$ and a proposition stating the node where the navigation starts.

A domain description given in ***B*** defines a transition function Φ that maps each pair of an action *a* and a state *s* into a set of states $\Phi(a,s)$. Intuitively, $\Phi(a,s)$ is the set of states that are reachable after executing the action *a* in state *s*. If $\Phi(a,s)$ is an empty set it means that *a* is not executable in *s*. The definition of $\Phi(a,s)$ can be found in [8,20]. Here, we illustrate it with an example.

Assume that the agent is at the node ‘*Itineraries*’. The initial state is given by

$$s = \{at(\text{‘Itineraries’})\} \cup G_{connected}.$$

Moving from ‘*Itineraries*’ to ‘*Departure’* is possible and we have that the only state belonging to $\Phi(move(\text{‘Itineraries’}, \text{‘Departure’}), s)$ is

$$s' = \{at(\text{‘Departure’})\} \cup G_{connected}.$$

On the other hand, moving from ‘*Itineraries*’ to ’*21:24*’ is impossible and hence

$$\Phi(move(\text{‘Itineraries’}, \text{’21:24’}), s) = \emptyset.$$

5.2. Reasoning about Actions and Planning using Logic Programming

Given an action theory (D, Γ) and a sequence of actions $a_0, \dots, a_k$. Let s_0 be a state that satisfies Γ. We can easily check whether a_0 is executable in s_0 and, if so, what is the set of states reachable from s_0 by a_0 by computing the set $\Phi(a_0, s_0)$. For a set of states S, let $\Phi(a,S) = \cup_{s\in S} \Phi(a,s)$. Let us denote

$$\Phi([a_0,\dots,a_i],s_0) = \Phi(a_i, \Phi(a_{i-1},\dots, \Phi(a_0,s_0))).$$

Clearly, if $\Phi([a_0,\dots,a_i],s_0)$ is not empty, a_l is executable in $\Phi([a_0,\dots,a_{l-1}],s_0)$ for $l=0,\dots,i$ and answering this question can be done by computing $\Phi([a_0,\dots,a_l],s_0)$. We solve this problem by developing a logic program Π(D, Γ) whose stable model semantics coincides with the transition function Φ [20]. The most important features of this program are

- It has two sets of rules, one is independent on how D and Γ are defined and other are the rules for encoding D and Γ in logic program terminologies.
- It has one parameter that represents the maximal number of steps our agent is willing to perform in order to achieve some goal. For this reason, we often write $\Pi_n(D, \Gamma)$ instead of Π(D, Γ) where n is the maximal number of steps. This is equivalent to say that we have a limit on the length of the plan needed to achieve some goal or a limit on the time for executing the plan.
- It uses an atom of the form occ(a,i) to represent the fact that an action a occurs at the time moment i. We call this an *action occurrence*. To represent the precondition that a can occur only if it is executable, $\Pi_n(D, \Gamma)$ uses atoms of the form *possible(a,i)*.

- It uses an atom of the form holds(f,i) (holds((¬f,i)) to represent the fact that fluent f (¬f) is true at the time moment i.
- Its stable models can be computed by using one of the efficient systems that can compute stable models of logic programs such as **dlv** and **smodels** [6,14].
- It can be used to reason about actions and their effects, and to planning to achieve some formula φ. To do the first task, we add to $\Pi_n(D, \Gamma)$ the action occurrences. To do the second task, we add to $\Pi_n(D, \Gamma)$ the constraint requiring that φ should be true at the end. This constraint is represented by the rule

$$\leftarrow \text{not } \text{holds}(\varphi, n).$$

The full detail about the program has been omitted for brevity (see [20]).

To compute $\Phi([a_0,...,a_i],s_0)$, we need to compute the stable models of the program

$$\Pi_{i+1}(D, \Gamma) \cup \{occ(a_l, l) \mid l=0,... ,i\}.$$

Whatever is true in the time moment i+1 will represent a state reachable from s_0 by executing the sequence of actions $[a_0,...,a_i]$. If the program does not have a stable model, the sequence of actions cannot be executed from s_0.

To find a sequence of actions that changes the world from s_0 to s_n where s_n satisfies the formula φ, we need to compute the stable models of program

$$\Pi_n(D, \Gamma) \cup \{\leftarrow \text{not } \text{holds}(\varphi, n)\}.$$

Each stable model gives us a trajectory $a_0s_0 \ldots a_{n-1}s_n$ such that φ is true in s_n.

5.4 A Simple Query Language

In ***B***, the query language ***Q*** allows queries

$$\varphi \ \textbf{after} \ [a_0,...,a_n] \tag{7}$$

that asks whether φ holds after the execution of $a_0,...,a_n$ from the current state. This can be directly used to perform interactive navigation, in the same style as the original DSL presented in [17]. E.g., a query such as

info(Id), at(Id) **after** $[a_0,...,a_n]$

will allow a user to access the information associated to *Id* after performing the actions $a_0,...,a_n$.

In table navigation, this type of queries is often not sufficient. We will discuss next several extensions to the query language of ***B*** that are useful for our applications.

5.4.1 Asking for a Possible Action

The first extension to ***Q*** is intended for use as a recommendation to users. Given an action theory (D, Γ) and a state s representing the current state of knowledge of our agent. It would be helpful in many situations to know what action is possible in the current state. Queries of this type are given in the following form:

possible(a) (8)

Using $\Pi_n(D, \Gamma)$, we can answer this question by checking if $\Pi_n(D, \Gamma) \cup \{occ(a, 0)\}$ has a stable model or not. To allow users to ask for all possible actions, we add to ***Q*** query of the form

possible(_) (9)

where _ stands for a directive that asks $\Pi_n(D, \Gamma)$ to compute all possible answers.

For example, a query such as

at(Id), possible(_)

allows one to determine the nodes of the semantic description that can be accessed from the current location.

5.4.2 Automatic Navigation: Agents

The basic navigation actions allow users to move from node to node. Queries of the form (7) make it possible for users to have some type of guided navigation. This is quite useful when the users are not familiar with the content of the table. However, once a user is familiar with the content of the table, the basic navigation procedure – moving from one node to the next node – could be time-consuming, and hence, is no longer appropriate. When the user is familiar with the content of the table, he might be interested in some information that is located in certain nodes. To this end, we propose to develop agents that, given a certain condition that is represented by the formula φ, automatically navigates the table until it finds the requested information, or it will inform the user that the information is not available in the table. In our system, this can be achieved by asking for a sequence of actions (or *a plan*) from the current state to a state satisfying φ and then executing it. To accommodate this, we add to ***Q*** queries of the following type:

$$\textbf{achieve}\ \ \varphi \tag{10}$$

This query asks for a plan that leads from the current state to a state which satisfies φ.

For example, it makes sense to know all possible itineraries. This can be requested by giving the query:

achieve *type(N,time), info(N), connected(Id, departure, N)*

where *Id* denotes the node *Iternaries*.

5.4.3 A Temporal Query Language

Queries of the forms (7)-(10) assume that our agent can make appropriate choices during its navigation. This means that our agent can compute all possible sequence of actions and then it is capable of selecting one of the best sequences that helps him in understanding the table. This is clearly not desirable for many reasons. First, finding all possible sequences of actions achieving a goal, even with bounded length, is computationally expensive. Furthermore, without any input from the user, the best thing the agent can do is to try each of the possible action sequences until it exhausts them. To this end, we propose an interactive mode of navigation that exploits the connection between table structure and table content, and allows users to make choices during navigation. The new navigation mode enhances the previous mode in many ways:

- It allows users to place constraints on the sequence of actions that the agent needs to compute; thus improving the efficiency of the planner. We note that empirical results in planning have proved that adding constraints to a planning problem can greatly improve the efficiency of planners [2].
- It allows the domain specifier to define table-dependent actions, procedure (or scripts) that can assist the users in understanding the table in many ways.
- It allows the user to interact with the agent in selecting the course of actions to perform.

We elaborate next on the different types of constraint that we consider and the way an user can interact with the system.

5.4.4 Temporal Constraints

Temporal constraints are used to prune the search space. Approach to planning with temporal constraints has been very successful [3]. A temporal constraint could be viewed as a condition placed on the sequence of actions (or plan) that helps the planner to remove correct but non-optimal plans from consideration. Temporal constraints can be specified using a linear temporal logic, a first order logic with the temporal connectives **next, until, eventually**, and **always**. For example, the formula

$$\mathbf{achieves}\ \forall X.connected(Leg\#1,\_,X),\ info(X)$$

represents a query that leads to the visit of all the nodes connected to the node *Leg#1*. Nevertheless, the corresponding aural presentation could be very confusing, as the nodes connected to *Leg#1* are explored in an unspecified order. This sort of problems can be solved by using temporal constraints. For instance, the temporal formula

$$\mathbf{always}\ (type(Id,time)\ \wedge move(Id_1,Id) \supset \mathbf{next}\ (move(Id\_1,Id)))$$

expresses the constraint that if the system visits the node *Id* of the type *time*, from any node Id_1, then the next move is to return to Id_1. In [20], we showed how this type of constraints can be added to $\Pi(D, \Gamma)$.

5.4.5 Procedural Constraints

Procedural constraint can be thought of as an (under-specified) sketch (or a partial plan) of the plans to be generated. This type of constraint has been used in GOLOG, an Algol-like logic programming language for agent programming, control and execution, based on a situation calculus theory of actions [18]. Interpreter for GOLOG programs can be found in [18]. For example, a simple GOLOG program

$$\text{move}(1,2);\ \text{move}(2,3);\ (\text{move}(3,4)\ |\ \text{move}(3,5)\ |\ \text{move}(3,6));\ \varphi?$$

represents plans which have move(1,2) followed by move(2,3), followed by one of move(3,4), move(3,6), or move(3,5) such that φ is true upon termination of the plan. The interpreter, when asked for a solution to this program, needs only to decide which one of move(3,4), move(3,6), or move(3,5) it should choose. The language allows complex actions to be specified as macros with the Algol-like constructors such as sequential constructor (**;**), branching constructor (**if-then-else**), repeat-until constructor (**while-do**), non-deterministic selection of action constructor (|), or non-deterministic selection of argument constructor (π). In [20], we developed an implementation of these constructors in logic programming, allowing their integration in the program $\Pi(D, \Gamma)$ to generate complete plans from plans' skeletons.

5.4.6 Table-Dependent Actions

Temporal constraints or procedural constraints require that users have some knowledge about the domain of the table. A domain specifier can also provide users with useful information that helps them in understanding the table in different ways. One possibility is to define table-dependent actions that are part of the action theory describing the table. Table-dependent actions are often careful developed procedure that will provide users with the most important aspects of the table. For example, for a table containing information about stock, the action *max_profit* that provides the user with the stock with the largest gain would be very useful. Similarly, the action of computing the average price of a stock in one day is also a useful action to understand the table.

Another possibility is to provide the user with a general ontology about the content of table that is understood by computer programs which can communicate this information to the user.

5.4.7 User Interactions

Using temporal constraints or procedural constraints, users can specify conditions or sketches of plans which are desirable to them before asking for a plan achieving their goals. This approach works well under the *completeness assumption*, that says that the planner has complete information about the world and actions never fail. This assumption has been criticized by the planning community as being too optimistic, especially in domains in which the environment might be changed by actions of other agents. To address this problem, on-line planning and execution monitoring has been proposed. In this approach, the agent has a mechanism to select the next action (it could be based on look-ahead or a set of if-then rules), commits to its choice and repeats the cycle until the overall goal is achieved. The execution monitoring component will make sure that the agent will not enter a deadlock situation. It is also used to recognize if an action has failed to achieve its effects. On-line planning and execution, however, is incomplete because the agent might select an action that never leads to the goal; and hence, the need for execution monitoring.

We argue that in our application, we can safely assume that the environment is not changed arbitrarily by other agents, and that actions never fail to achieve their effects if they are executable. Furthermore, in our application, agent can backtrack to the previous situation, simply by executing the reversing movement action[3]. However, we argue that the on-line planning and execution techniques can be applied in our application to guide the navigation and to exploit user's knowledge about the content of the table, especially, when procedural constraints are employed. This can be introduced in our framework by

---

[3] The reverse of move(a,b) is move(b,a).

adding a *wait* action to the action theory representing the table. This action makes the agent wait for a user input before continuing. We assume that this action will provide the agent with enough knowledge to make the immediate decision. The wait action can be inserted anywhere in a plan but it is most appropriate at places where an agent might not have enough knowledge to decide what to do. For example, for Table 1, it would be useful to know which itinerary the user is interested in before presenting her/him with the details. For this purpose, the following program can be used

ReadItineraries:

∀ X.[*type(X,time),connected(Iternaries,Departure,X)*

if *wait(X?)* **then** *ReadItinerary(X)*]

assuming that *ReadItinerary(X)* is defined in the system. Here, the fact that *wait(X?)* is true means that the user wants to know about the itinerary with the departure time *X*.

User interaction is one way to completing a partial plan. This task can also be accomplished by matching a given partial plan with predefined plans that have been stored. To this end, the agent needs a library of predefined plans or previously executed plans. We believe that each user, through his experience, can build a library of plans that are useful to him. The planner can also provide users with the set of possible sequence of actions and ask them for decision.

6. Implementation Issues

The infrastructure for the accessibility agents is currently under development. In particular:

- **System Interfaces**: The interfacing of the various system components with the outside world has been accomplished using standard off-the-shelf interfaces. The accessibility agent interacts with the user through a combination of aural and keyboard interfaces. The agent interacts with the outside world through the use of *transcoding proxies* – a dedicated proxy server aimed at filtering all incoming HTTP requests generated by the user. Data obtained from remote servers are exchanged with

1. The *Knowledge Manager*, to retrieve from the local knowledge bases any existing information regarding the document received (e.g., existing semantic representations, user defined annotations, connect to the related ontologies);
2. The *Document Analyzers*, to generate the semantic representation of the incoming document.

Implementation of the transcoding proxies has been accomplished through the use of IBM's WBI Development Kit (Web Based Intermediary) along with its Java APIs. WBI provides a generic proxy server infrastructure that can be customized through the integration of plug-in modules – the plug-ins are used to define the behavior of the proxy server.

- **Speech Synthesis:** The aural browser makes use of IBM ViaVoice system as engine for text-to-speech translation.
- **Reasoning:** the development of the reasoning component of the accessibility agent will rely on the use of inference engines for *Answer Set Programming* [6,14]. The mapping between action theories describing the behavior of the accessibility agent and logic programs, allows us to reduce the task of developing plans to the task of computing stable models of a logic program [8].

7. Conclusions

In this paper we propose the design of an intelligent agent to improve accessibility to the World Wide Web for visually impaired individuals. In particular, we focus on the accessibility of components relevant to the domain of e-commerce (e.g., tables). Our agent design allows us to view the process of navigating a complex HTML document – e.g., a document containing tables and frames – as solving *goals* with respect to an *action theory*. This work extends our previous successful proposal for the design of a Domain Specific Language for navigation of tables and frames [17]. The new proposal maintains the underline concept of making use of knowledge representation technology – specifically conceptual structures – to explicitly capture and represent the *navigational semantics* of each document. Such explicit semantic representation is aimed at providing to the non-visual user those components of the semantic of a document that are commonly implicit in the visual layout of the document. The novelty of this proposal is the use of an agent structure to encapsulate the task of acquiring documents, developing semantic representations, and using such representations during the navigation process. The agent provides a level of intelligence in the navigation process, allowing users to express high level navigation goals and using *planning in action theories* to solve such goals. Effectively, the agent performs navigation of the complex components of HTML documents on behalf of the visually impaired user. Viewing navigation as development of a plan offers numerous novel capabilities, such as ability to introduce temporal and procedural constraints, partial plans, and table dependent actions/procedures. The approach we have taken to solve the problem – by separating the semantics of the document from its visual layout – follows the guidelines adopted for the development of the *Semantic Web* [4]. In our future work we plan to adopt techniques from this growing field of research to help achieving our goals.

References

[1] Asakawa, C. and Laws, C. Home Page Reader: IBM's Talking Web Browser, Technical report, IBM, 1998.

[2] Bacchus, F. and Kabanza, F. Using temporal logics to express search control knowledge for planning, Artificial Intelligence, 116(1-2), 123-191, 2000.

[3] Bacchus, F. et al., AIPS-00 Planning Competition, International Conference on Artificial Intelligence Planning and Scheduling Systems, 2000.

[4] Berners-Lee, T., Hendler, J., and Lassila, O., The Semantic Web, Scientific American, 284(5), 34-43, 2001.

[5] Cattagni, A. and Westat, E. Farris, Internet Access in U.S. Public Schools and Classrooms. U.S. Dept. of Education, 2001.

[6] Citrigno, S., Eiter, T., Faber, W., Gottlob, G., Koch, C., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F., The DLV System: Model Generator and Application Frontends, In *Workshop on Logic Programming*, 128-137, 1997.

[7] Earl, C. and Leventhal, J., A Survey of Windows Screen Reader Users, Journal of Visual Impairment and Blindness, 93(3), 1999.

[8] Gelfond, M. and Lifschitz, V., Action Languages, ETAI, 3(6), 1998.

[9] Gunderson, J. and Mendelson, R., Usability of World Wide Web Browsers by Persons with Visual Impairments, RESNA Annual Conference, 330-332, 1997.

[10] Gupta, G. and Pontelli, E., Specification, Implementation, and Verification of Domain Specific Languages, Computational Logic: from Logic Programming into the Future, Springer Verlag, 2001.

[11] James, F., Presenting HTML Structures in Audio, Technical Report, Stanford University, 1997.

[12] Karshmer, A., Pontelli, E., and Gupta, G., Software Technology and Computer Interfaces for the Disabled: of the WWW, HCI International, 1999.

[13] Kennel, A., Perrochon, L., and Darvishi, A., WAB: World-Wide Web Access for Blind And Visually Impaired Computer Users, New Technologies in the Education of the Visually Handicapped, 1996.

[14] Niemela, I., and Simons, P., Smodels: an Implementation of the Stable Model Semantics, Logic Programming and NonMonotonic Reasoning, Springer Verlag, 1997.

[15] Oogane, T. and Asakawa, C., An Interactive Method for Accessing Tables in HTML, *ASSETS*, ACM, 1998.

[16] Pontelli, E., Gillan, D., Gupta, G., Karshmer, A., Xiong, W., and Saad, E., Non-visual Navigation of Tables and Frames: a Progress Report, *Proceedings of the Int. Conference on Assistive Technologies,* ACM Press, 2002.

[17] Pontelli, E., Gillan, D., Gupta, G., Karshmer, A., and Xiong, W., A Domain-Specific Language Framework for Non-visual Browsing of Complex HTML Structures, *ASSETS*, 180-187, ACM Press, 2000.

[18] Reiter, R., Knowledge in Action, MIT Press, 2001.

[19] Ribiere, M. and Dieng, R., Introduction of Viewpoints in Conceptual Graph Formalism, *International Conference on Conceptual Structures*, Springer Verlag, 1997.

[20] Son, T. C., Baral, C., and McIlraith, S., Domain Dependent Knowledge in Planning - An Answer Set Planning Approach, Logic Programming and NonMonotonic Reasoning, Springer Verlag, 226-239, 2001.

[21] Sowa, J. F., Conceptual Structures: Information Processing in Mind and Machine, Addison Wesley, 1984.

[22] Vaderheiden, G., Chisholm, W., and Jacobs, I., WAI Accesibility Guidelines: Page Authoring, W3C Technical Report, WD-WAI-PAGEAUTH-19980918, 1998.

[23] Zajicek, M. et al., Ergonomic Factors for a Speaking Computer Interface, *Contemporary Ergonomics*. Taylor & Francis, 1999.

[24] Freedom Scientific Inc., JAWS 4.5 User Manual, 2002.

[25] *E-Commerce Unfriendly for People with Disabilities,* accessibility.com.au, 2003.

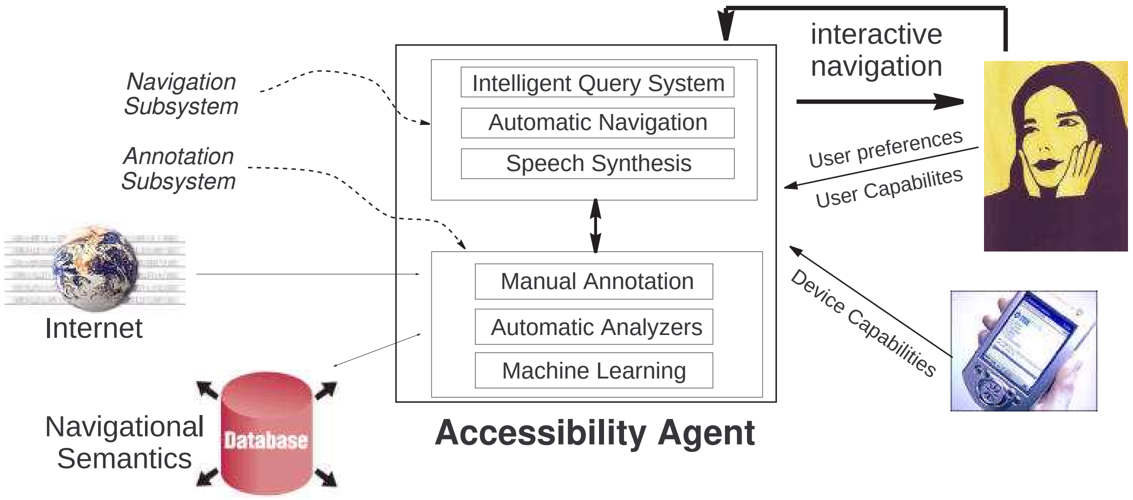

FIGURE 1: Overall Structure of the Accessibility Agent

Departure	Arrival	Connection Information		Train Information	Duration
		Station	Arrival		
21:24	8:42			ICE	11:18
03:27		Bologna	02:25	Marco Polo	
07:10		Milano	06:00	E1616	
05:36	12:42			IC	07:06
11:10		Milano	09:50	Fogazzaro	
07:29	14:42			ICIC	07:13
09:35		Mestre	08:55	Adrea Doria	
15:21	21:47			EC CIS IC	06:26
17:07		Mestre	16:33	Goldoni	
20:10		Milano	19:55	CIS38	
18:36	04:45			ICE	10:09
00:32		Verona	21:11	Giorgione	

Table 1: Sample Table: Train Itineraries

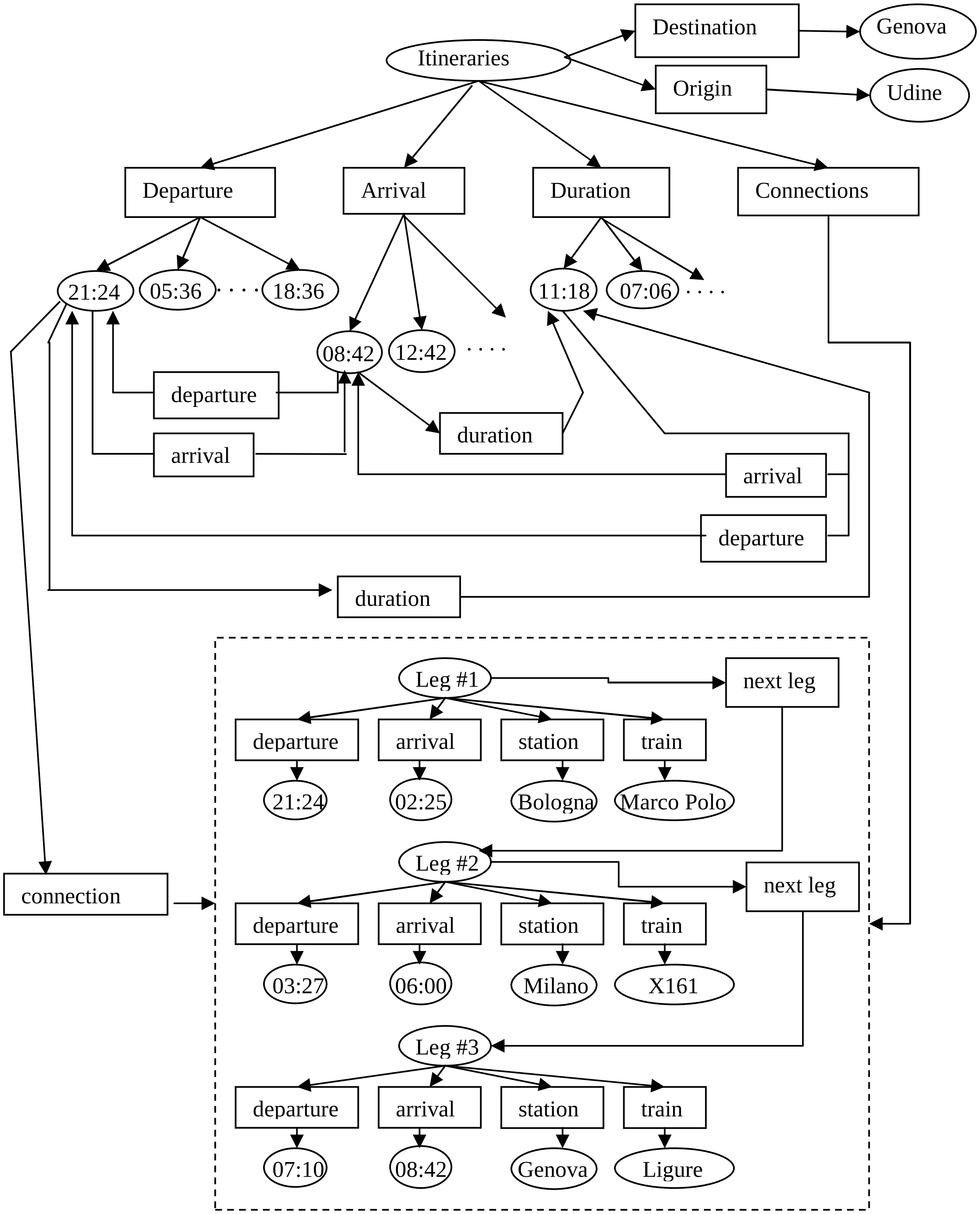

FIGURE 2: Fragment of the Semantic Description for the Table 1