# Modeling Interactive Web Sources for Information Mediation

Bertram Ludäscher and Amarnath Gupta

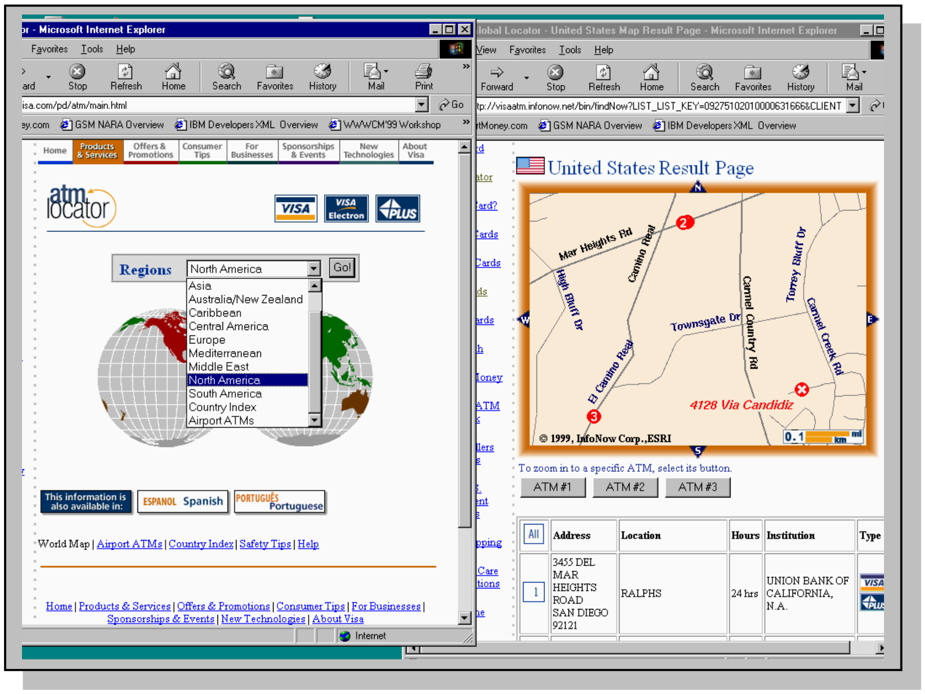
San Diego Supercomputer Center, UCSD, La Jolla, CA 92093-0505, USA
{ludaesch,gupta}@sdsc.edu

**Abstract.** We propose a method for modeling complex Web sources that have active user interaction requirements. Here "active" refers to the fact that certain information in these sources is only reachable through interactions like filling out forms or clicking on image maps. Typically, the former interaction can be automated by wrapper software (e.g., using parameterized urls or `post` commands) while the latter cannot and thus requires explicit user interaction. We propose a modeling technique for such interactive Web sources and the information they export, based on so-called *interaction diagrams*. The nodes of an interaction diagram model sources and their exported information, whereas edges model transitions and their interactions. The paths of a diagram correspond to sequences of interactions and allow to derive the various query capabilities of the source. Based on these, one can determine which queries are supported by a source and derive query plans with minimal user interaction. This technique can be used offline to support design and implementation of wrappers, or at runtime when the mediator generates query plans against such sources.

## 1 Introduction

The mediator framework [Wie92] has become a standard architecture for information integration systems. Whenever the integration involves large or frequently changing data, like e.g. Web sources,[1] a *virtual* integration approach (as opposed to a *warehousing* approach) is advantageous [FLM98]: at runtime, the user query against a mediated view is decomposed at the mediator and corresponding subqueries are sent to the source wrappers. The answers from the sources are collected back at the mediator which, after some post-processing, returns the integrated results to the user. When implementing such a framework, one has to deal with the different *capabilities* of mediators and wrappers: Mediators are the main query engines of the architecture and thus are usually capable to answer arbitrary queries in the view definition language at hand. In contrast, wrappers often provide only limited query capabilities, due to the inherent query restrictions induced by the sources. For example, when a book-shopping mediator generates query plans, it has to incorporate the limited query capabilities

[1] Since we are focusing on Web sources, we often use the terms *source* and *Web page/site* interchangeably.



Microsoft Internet Explorer
Favorites Tools Help
Stop Refresh Home Search Favorites History Mail Print
isa.com/pd/atm/main.html
GSM NARA Overview IBM Developers XML Overview WWWCM'99 Workshop
Home | Products & Services | Offers & Promotions | Consumer Tips | For Businesses | Sponsorships & Events | New Technologies | About Visa
atm locator
VISA | VISA Electron | PLUS
Regions
North America
Go!
Asia
Australia/New Zealand
Caribbean
Central America
Europe
Mediterranean
Middle East
North America
South America
Country Index
Airport ATMs
This information is also available in:
ESPANOL Spanish
PORTUGUÊS Portuguese
World Map | Airport ATMs | Country Index | Safety Tips | Help
Home | Products & Services | Offers & Promotions | Consumer Tips | For Businesses | Sponsorships & Events | New Technologies | About Visa
Internet
lobal Locator - United States Map Result Page - Microsoft Internet Explorer
View Favorites Tools Help
Forward Stop Refresh Home Search Favorites History Mail
tp://visaatm.infonow.net/bin/findNow?LIST_LIST_KEY=09275102010000631666&CLIENT
GSM NARA Overview IBM Developers XML Overview
United States Result Page
Mar Heights Rd
Camino Real
High Bluff Dr
Townsgate Dr
Carmel Country Rd
Torrey Bluff Dr
Carmel Creek Rd
El Camino Real
4128 Via Candidiz
© 1999, InfoNow Corp.,ESRI
To zoom in to a specific ATM, select its button.
ATM #1
ATM #2
ATM #3

| All | Address | Location | Hours | Institution | Type |
|---|---|---|---|---|---|
| 1 | 3455 DEL MAR HEIGHTS ROAD SAN DIEGO 92121 | RALPHS | 24 hrs | UNION BANK OF CALIFORNIA, N.A. | VISA PLUS |

**Fig. 1.** An interactive Web source (ATM locator)

of wrappers say for `amazon.com` or `barnesandnoble.com` in order to send only *feasible* queries that these wrappers can process. As shown in [GMLY99] query processing on Web sources can greatly benefit from "capability-aware" mediators.

In order to model source capabilities, various mechanisms like query templates, capability records, and capability description grammars have been devised and incorporated into mediator systems. An implicit assumption of current mediator systems is that once the sources have exported their capabilities to the mediator, all user queries can be processed in a *completely automated* way. In particular, there is *no interaction* necessary between the user and the source while processing a query. Specifically for Web sources, there is a related assumption that the Web page which holds the desired source data is reachable, e.g., by traversing a link, composing a url, or filling in a form.

While it is clearly desirable to relieve the user from interacting with the source directly, it is our observation that for certain sources and queries, these assumptions break down and explicit user interaction is inevitable in order to process these queries.

**Example 1 (ATM Locator).** Assume we want to wrap an ATM locator service like the one of `visa.com` into a meditor system. There are different means to locate ATMs in the vicinity of a location: Starting from the entry page, the user may either select a region (say North America) from a menu or click on the corresponding area on a world map (Fig. 1, left). At the next stage the user fills in a form with the address of the desired location. The source returns a map with the nearby ATM locations and a table with their addresses (Fig. 1, right). Thus, for given attributes like *region*, *street*, and *city*, one can implement a wrapper which automatically retrieves the corresponding ATM addresses.

However, if the task is to select one or more specific ATMs based on properties *visible from the map only* (e.g., select ATMs which are close to a hotel), or if the selection criterion is imprecise and user-dependent (*"I know what I want when I see it"*), then an explicit user interaction is required and the source interface has to be exposed directly to the user. □

In this paper we explore the problem of bringing such sources, i.e., which *may* require explicit user interactions, into the scope of mediated information integration. To simplify the presentation, we use a relational attribute-specification model for modeling query capabilities. The outline and contributions of the paper are as follows:

- In Section 2 we show how the capabilities and interactions of Web sources can be modeled in a concise and intuitive way using *interaction diagrams.* To the best of our knowledge, this is the first approach which allows to model complex Web sources with explicit user interactions and brings them into the realm of the information mediation framework.
- In Section 3 we show how to derive the combined capabilities of the modeled sources given the capabilities of individual interactions. This allows us to distinguish between automatable (*wrappable*) queries and queries which can only be supported with explicit user interaction. Based on this, a "diagram-enabled" wrapper can suggest alternative queries to the mediator, in case the requested ones are not directly supported.
- In Section 4 we sketch how interaction diagrams can be embedded into the standard mediator architecture and discuss the benefits of such a "diagram-enabled" system. We give some conclusions and future directions in Section 5.

**Related Work.** Incorporating capabilities into query evaluation has recently gained a lot of interest, and several formalisms have been proposed for describing and employing query capabilities [PGGMU95,LRO96,PGH98,YLGMU99]. Especially, in the context of Web sources, query processing can benefit from a capability-sensitive architecture as noted in [GMLY99]. Their work focuses on efficient generation of feasible query plans at the mediator, i.e., plans which the source wrappers can support. Target queries are of the form $\pi_{\boldsymbol{b}}(\sigma_{\psi(\boldsymbol{a})}(R))$ where $\psi(\boldsymbol{a})$ is a selection condition over the input attributes $\boldsymbol{a}$, and $\boldsymbol{b}$ are the output attributes. Their capability description language SSDL deals with the different forms of selection conditions $\psi(\boldsymbol{a})$ that a source may support. In contrast, we address the problem of "chaining together" queries like $\pi_{\boldsymbol{b}}(\sigma_{\psi(\boldsymbol{a})}(R))$ which allows us to model *complex sequences of interactions.*

Several formalisms for modeling Web sites have been proposed which more or less resemble interaction diagrams: For example, the *Web skeleton* described in [LHL$^+$98] is a very simple special case of diagrams where the only interaction elements are hyperlinks. Other much more versatile models have been proposed like the Web schemes of the ARANEUS system [AMM97,MAM$^+$98], which use a page-oriented ODMG-like data model, extended with Web-specific features like forms. Another related formalism are the *navigation maps* described in [DFKR99]. The problem of deriving the capabilities and interaction requirements of *sequences of interactions* (i.e., paths in the diagram) is related to the problem of propagating binding patterns through views as described in [YLGMU99]. There, an in-depth treatment on how to propagate various kinds of binding patterns (or *adornments*) through the different relational operators is provided.

However, the approaches mentioned above do not address the problem of modeling sources with *explicit user interactions* and how to incorporate such sources into a mediator architecture.

## 2 Modeling Interactive Sources

As illustrated above, there are user interactions with Web sources which can be wrapped (e.g., link traversal, selections from menus, filling of forms), and others which require explicit user interaction (e.g., selections from image maps, GUI's based on Java, VRML, etc.). Before we present our main formalism for modeling such sources, *interaction diagrams*, let us consider how the typical interaction mechanisms found in Web sources relate to database queries.

### Modeling Input Elements

Below, for an attribute $a$, we denote by $\$a$ the *actual parameter value* for $a$ as supplied by the corresponding input element. In first-order logic parlance, we may think of $a$ as a *variable* which is *bound* to the *value* $\$a$. Hence, we often use the terms *attribute* and *variable* interchangeably. We denote vectors and sequences in ***boldface***. For example $\boldsymbol{a}$ stands for some attributes $a_1, \ldots, a_n$.

We can associate the following general query scheme with the input elements discussed below:

$$\pi_{\boldsymbol{b}}(\sigma_{\psi(\boldsymbol{a})}(R)).$$

Here, $\boldsymbol{a}$ are the *input parameters* to the source, $\boldsymbol{b}$ are the desired *output parameters* to be extracted from the source, and $\boldsymbol{a}, \boldsymbol{b} \in \mathit{atts}(R)$, i.e., the relation $R$ being modeled has attributes $\boldsymbol{a}, \boldsymbol{b}$. The first-order predicate $\psi$ over $\boldsymbol{a}$ is used to select the desired tuples from $R$. With every input element we will associate a *default semantics*, according to their standard use.

**Hyperlinks** are the "classical" way to provide user input. We can view the traversal of a link $\mathsf{href}(a)$ as providing a value $\$a$ for the single input attribute $a$. The value $\$a$ is given by the label of the link. For example, by clicking on a specific airport code, we can bind *apc* (*airport-code*) to the label of the link. Assume we want to extract the zip code from the resulting page. We can model this according to the above scheme as $\pi_{zip}(\sigma_{apc=\$apc}(R))$. Since equality is the most common selection condition for links, we define it as the default semantics for $\mathsf{href}(a)$:

$$\psi(a) := \quad (a = \$a).$$

Other meanings are possible. For example, a Web source my contain a list of maximal prices for items such that by clicking on a maximal value $\$price$, only items for which $price \leq \$price$ are returned. Thus, $\psi(price) := (price \leq \$price)$. Note that the domain of $a$ is finite, since the source page can have only finitely many (static) links.

**Forms** can be conceived as *dynamic links* since the target of "traversing" such a link depends on the form's parameters. In contrast to hyperlinks, forms generally involve *multiple input attributes*, ranging over an *infinite* domain (think of a form-based tax calculator). For example, given a street and a city name, the extraction of zip codes from the result page can be modeled by the input element $\mathsf{form}(street, city)$ and the query $\pi_{zip}(\sigma_{street=\$street \wedge city=\$city}(R))$. Similar to $\mathsf{href}$, the most common use and thus our default semantics is to view forms as conjunctions of equalities:

$$\psi(\boldsymbol{a}) := \quad (a_1 = \$a_1 \wedge \ldots \wedge a_n = \$a_n).$$

In contrast, by setting $\psi(low, high) := (\$low \leq price \leq \$high)$ we can model a *range query* which retrieves tuples from $R$ whose *price* is within the specified interval. *Optional parameters* can be modeled as well: let $\psi(a, b) := ((a = \$a \wedge b = \$b) \vee a = \$a)$. This models the situation that $b$ is optional: if $b$ is undefined (set to NULL), the first disjunct will be undefined while the second can still be true.

**Menus** are used to select a *subset of values* from a predefined, *finite* domain. For example, the selection of one or more US states from a menu is denoted by $\mathsf{menu}(state)$. When modeling a menu attribute, it can be useful to include the attribute's domain. For example, we may set $dom(state) := \{\text{ALABAMA}, \ldots, \text{WYOMING}\}$. In contrast to $\mathsf{href}(a)$ and $\mathsf{form}(a)$, *multiple values* $\$a_1, \ldots, \$a_n$ can be specified for the *single* attribute $a$. The default semantics for menus is

$$\psi(a) := \quad (a = \$a_1 \vee \ldots \vee a = \$a_n).$$

**Non-Wrappable Elements.** Conceptually, the elements considered above are all *wrappable* in the sense that the user interaction can be automated by wrappers, which have to fill in forms (via http's `post`), traverse links (`get`), possibly after constructing a parameterized url[2], etc. We call an interaction *non-wrappable*

[2] E.g., `search.yahoo.com/bin/search?p=a+OR+b` finds documents containing `a` or `b`. Other elements like radio buttons and check boxes can be modeled similarly.

if, using a reasonable conceptual model of the source, an explicit user interaction is inevitable to perform that interaction (cf. Section 3). Note that technically, selections on clickable image maps could also be wrapped, say using a url which is parameterized with the xy-coordinates of the clicked area. However, this is usually not an adequate conceptual model of the query, since the xy-coordinates do not provide any hint on the semantics of the query being modeled. Thus, for sources where automated wrapping is impractical or inadequate from a modeling perspective, we "bite the bullet" and ask the user for explicit interaction with the source. To model such a user interaction, we use the notation

$$!\mathsf{ui}(a_1, \ldots, a_n)$$

where $a_1, \ldots, a_n$ are input parameters on which the user interaction depends. Consider, e.g., an image map on which the user clicks to change the focus or pan the image. We can regard this as an operation with some input parameter *loc*, representing the current location, and some otherwise unspecified, implicit input parameter $x$ which models the performed user interaction.[3] In this case, we can model the interaction as $\pi_{loc'}(\sigma_{loc=\$loc, x=\$x}(R))$ where $R$ has input attributes *loc* and $x$ and an output attribute *loc*$'$.

## Modeling Complex Sources with Interaction Diagrams

To describe the query capabilities of and dependencies within complex interactive sources, we propose a formalism in the spirit of state-transition diagrams, called *interaction diagrams*, or *diagrams* for short. Conceptually, a *node* of a diagram is viewed as an individual source. A source can be a single Web page or comprise several pages which exhibit the same input/output behavior. With every node we associate an *export schema*, representing the information provided by that source. The possible *transitions* between (the states of) sources are modeled by *labeled edges*. The edge label specifies the type of interaction (form, menu, !ui, etc.) which is required to perform the transition and determines the capabilities of the source wrt. this transition.

**Diagrams.** More precisely, an *interaction diagram d* for a source is defined over a set of attributes *atts* (=*atts*($d$)) and consists of *labeled nodes* and *labeled directed edges*. Attributes are used to describe the modeled entities of the source and thus are the basic building blocks of the source description. In the sequel, let $a, a_1, a_2, \ldots \in atts(d)$.

**Sources.** The nodes of $d$ are called *sources* and model identifiable units within the complex source, most notably Web pages. Sources have an identifier (node id), typically a url and an *output schema* of exported attributes $a_1, \ldots, a_k$. These attributes model all relevant information exported by the source.

[3] E.g., $x$ could be an encoding of xy-coordinates. However, we do not provide or need a way to dissect $x$ since it acts merely as an identifier.

We distinguish different ways in which attributes can be exported in an output schema:

- $(a_1, \ldots, a_n)$: the source exports these attributes *tuple-at-a-time*,
- $\{(a_1, \ldots, a_n)\}$: the source exports a *set* of such tuples,
- $[(a_1, \ldots, a_n)]$: the source exports an *ordered list* such tuples.

This structural information can be used to derive additional query properties of sources like (non-)uniqueness of results and availability of order, thereby supporting the wrapper design.

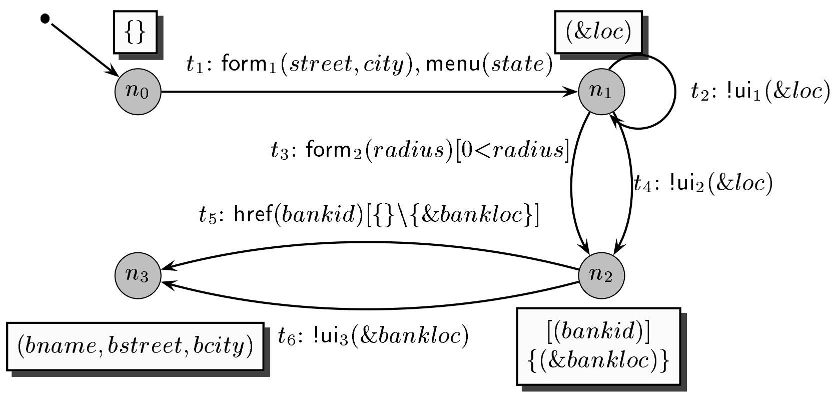
**Internal Attributes.** Sometimes, attribute values are not explicitly provided by the source (e.g., the location corresponding to the center of the image in Fig. 1, or the attribute *loc* in Example 2). Although such a source may be "stateful" and remember the current value of *loc*, this value is implicit and cannot be directly extracted by a wrapper. Nevertheless, the output of subsequent interactions may very well depend on the value of the latent *loc* attribute, so we cannot ignore it. We say that an attribute $a$ is *internal*, denoted by $\&a$, if the actual value of $a$ is not directly extractable.

**Transitions.** Labeled edges are used to model possible *transitions* between (states of) sources. Each transition $t$ is labeled with one or more *interactions* $\boldsymbol{i} = i_1, \ldots, i_n$, where each interaction $i_j \in \{\mathsf{href}, \mathsf{form}, \mathsf{menu}, !\mathsf{ui}\}$ has input parameters from $atts(d)$. Transitions can be further constrained by attaching conditions as follows. The transition

$$t: \; x \xrightarrow{\boldsymbol{i}[\varphi, \boldsymbol{u} \setminus \boldsymbol{v}]} y$$

means that one can move from source $x$ to $y$ using the interaction(s) $\boldsymbol{i}$ only if the first-order condition $\varphi$ holds. This allows modeling of additional semantic constraints which are enforced by a source (e.g., a source having $\mathsf{form}(low, high)$ may enforce $\varphi := \$low < \$high$). Note that by default, the input parameters of $t$ are the attributes occurring in $x$ or $\boldsymbol{i}$. In order to indicate that the outcome of $t$ also depends on the attributes $\boldsymbol{u}$ but not on $\boldsymbol{v}$, the expression $\boldsymbol{u} \setminus \boldsymbol{v}$ is used.

**Example 2 (Going to the Bank).** The diagram in Fig. 2 defines the following model of a complex source: The source at the start node $n_0$ does not export information relevant to the application. By filling in $\mathsf{form}_1$ with a street and city name and selecting a state from $\mathsf{menu}$, we can move to source $n_1$, which provides an internal attribute $\&loc$ for the requested location ($t_1$). As long as we apply user interactions $!\mathsf{ui}_1$, we remain at source $n_1$, possibly updating its internal location attribute ($t_2$). We can move to $n_2$ by filling in another form $\mathsf{form}_2$ with a positive radius, or by executing some user interaction $!\mathsf{ui}_2$ (say marking a rectangular region). Source $n_2$ exports a list of bank identifiers (say as hyperlinks), and a set of *internal* bank locations (e.g., as a clickable image map). There are two ways to move to $n_3$: We may either traverse a link which



{}
$t_1$: form$_1$(*street*, *city*), menu(*state*)
(&*loc*)
$n_0$
$n_1$
$t_2$: !ui$_1$(&*loc*)
$t_3$: form$_2$(*radius*)[0<*radius*]
$t_4$: !ui$_2$(&*loc*)
$t_5$: href(*bankid*)[{}\{&*bankloc*}]
$n_3$
$n_2$
(*bname*, *bstreet*, *bcity*)
$t_6$: !ui$_3$(&*bankloc*)
[(*bankid*)]
{(&*bankloc*)}

**Fig. 2.** A diagram for an interactive source (for retrieving addresses of nearby banks)

is labeled with the bank id ($t_5$), or we use another user interaction !ui$_3$ (again by clicking on an image map). Note that we can ignore the &*bankloc* attribute in $t_5$ (as specified in brackets), since the *bankid* is sufficient to execute $t_5$. □

## 3 Deriving Source Capabilities

While interaction diagrams are a useful modeling tool in itself, their main purpose is to support the automatic derivation of capabilities of complex interaction patterns of the modeled source.

**Single Transitions.** The capabilities and interaction requirements of a single transition

$$t : x \xrightarrow{\boldsymbol{i}[\varphi,\boldsymbol{u}\backslash\boldsymbol{v}]} y$$

are modeled as follows:

- $in(t) := (atts(x) \cup atts(\boldsymbol{i}) \cup \boldsymbol{u}) \setminus \boldsymbol{v}$ are the required *input attributes*,
- $out(t) := atts(y)$ are the *output attributes* which are exported by $t$, and
- $act(t) := \boldsymbol{i} \wedge \varphi$ are the *interaction (execution) requirements* of $t$.

Here, $atts(\ldots)$ is the set of all attributes occurring in the corresponding expression. For each user interaction !ui$_k$, we also add a *new internal attribute* $\&x_k$ representing the user interaction, so

$$atts(!\mathsf{ui}_k(a_1, \ldots, a_n)) := \{a_1, \ldots, a_n, \&x_k\}.$$

**Transition Paths.** Based on the above notions, we can derive the query capabilities along paths, i.e., *sequences of transitions*. A *path* of a diagram $d$ is a sequence $t_1.t_2.\cdots.t_n$ of connected transitions, i.e., where the target of $t_i$ meets the source of $t_{i+1}$. Let $\boldsymbol{t} = t_3.\cdots.t_n$ and $t_1.t_2.\boldsymbol{t}$ be paths of $d$ (here, $\boldsymbol{t}$ may be empty). The capabilities and interaction requirements along paths are inductively defined as follows:

- $in(t_1.t_2.\boldsymbol{t}) := in(t_1) \cup (in(t_2.\boldsymbol{t}) \setminus propagate(\varphi_{con}(t_1,t_2)))$ are the *input attributes*,
- $out(t_1.t_2.\boldsymbol{t}) := out(t_2.\boldsymbol{t})$ are the *output attributes*, and
- $act(t_1.t_2.\boldsymbol{t}) := act(t_1) \otimes \varphi_{con}(t_1,t_2) \otimes act(t_2.\boldsymbol{t})$ is the *execution plan* (*interaction requirement*).

Here, we have assumed that

- $\varphi_{con}(t_1,t_2)$ is a first-order predicate over $out(t_1) \cup in(t_2)$ which "connects" the output of $t_1$ with the input of $t_2$. By default, $\varphi_{con}(t_1,t_2)$ is defined as the *natural join* over the common attributes of $out(t_1)$ and $in(t_2)$. This default can be overridden by attaching an explict condition at the node connecting $t_1$ and $t_2$,
- $propagate(\varphi_{con}(t_1,t_2))$ are those attributes of $in(t_2)$ whose bindings are propagated from $out(t_1)$ using $\varphi_{con}(t_1,t_2)$. Thus, in the default case with natural joins:

$$propagate(\varphi_{con}(t_1,t_2)) := out(t_1) \cap in(t_2)$$

- "$\otimes$" is a binary, right-associative connective denoting *serial conjunction*.[4] Thus, an execution plan has two readings, both of which have to be obeyed: (i) as a logic formula where the first-order conditions $\varphi$ and $\varphi_{con}$ of the plan are viewed as conjunctively connected, and (ii) as a linear sequence of required interactions $\boldsymbol{i}$.

The *query capabilities* along a non-empty path $\boldsymbol{t} = t_1.\cdots.t_n$ is then denoted by the *binding pattern*

$$q_{\boldsymbol{t}}(in(\boldsymbol{t}), out(\boldsymbol{t})).$$

Below, we "sign" an attribute $a$ to indicate whether it is input ($+a$), output ($-a$), or both ($\pm a$).

**Example 3 (Banks Revisited).** Consider the diagram in Fig. 2. It is easy to derive the binding patterns

- $q_{t_1}(+street, +city, +state, -\&loc)$, and
- $q_{t_3}(+\&loc, +radius, -bankid, -\&bankloc)$.

We can connect $t_1$ and $t_3$ via $n_1$ by a first-order predicate $\varphi_{con}(t_1,t_3)$ which, by default, is set to the natural join over the common attributes, so $\varphi_{con}(t_1,t_3) =$

$$q_{t_1}(+street, +city, +state, -\&loc) \bowtie_{\&loc} q_{t_3}(+\&loc, +radius, -bankid, -\&bankloc),$$

from which we derive the binding pattern

$$q_{t_1.t_3} = (+street, +city, +state, +radius, -bankid, -\&bankloc).$$

[4] This notation and terminology is borrowed from Transaction Logic [BK94].

Now consider $t_5$. Since $t_5$'s only input attribute, *bankid*, is exported by the target node of $t_1.t_3$, we can connect $t_1.t_3$ and $t_5$ via $n_2$ without the need to supply additional input parameters. From this we obtain

$$q_{t_1.t_3.t_5}(+street, +city, +state, +radius, -bname, -bstreet, -bcity) \; .$$

The execution plan necessary to implement $q_{t_1.t_3.t_5}$ is

$$\begin{aligned} act(t_1.t_3.t_5) = \; & \mathsf{form}_1(street, city), \mathsf{menu}_1(state) \\ & \otimes \varphi_{con}(t_1, t_3) \\ & \otimes \mathsf{form}_2(radius) \wedge radius > 0 \\ & \otimes \varphi_{con}(t_3, t_5) \\ & \otimes \mathsf{href}(bankid) \end{aligned}$$

where $\varphi_{con}(t_3, t_5)$ is the natural join $q_{t_3}(\ldots) \bowtie_{bankid} q_{t_5}(\ldots)$. $\square$

By modeling complex interactive sources using diagrams, we can derive the different query capabilities and associated interaction requirements which are necessary to implement these queries at the sources. Indeed, from the above definitions one can easily derive an algorithm which, given a diagram $d$ and a non-empty transition path $\boldsymbol{t}$ of $d$, computes the unique binding pattern $q_{\boldsymbol{t}}$. This is an important prerequisite for enabling capability-sensitive query processing in a mediator framework (cf. Section 4) and allows us to determine the *supported* (or *feasible*) queries of a source:

We say that a binding pattern $q(in, out)$ *subsumes* a pattern $q'(in', out')$, if $in \subseteq in'$ and $out \supseteq out'$ (since we can answer $q'$ by resorting to $q$); $q$ and $q'$ are called *equivalent*, if they subsume each other.

A query $q(in, out)$ is *supported* by a source if there is a path $\boldsymbol{t}$ in the corresponding diagram such that $q_{\boldsymbol{t}}$ subsumes $q$. Finally, $q$ is called *wrappable* (or *automatable*), if it is supported by some path $\boldsymbol{t}$ such that $act(\boldsymbol{t})$ does not contain any user interaction $\mathsf{!ui}$.

**Example 4 (Properties of Transition Paths).** Assume the mediator sends a request of the form $q(+street, +city, -bname)$ to the source modeled in Fig. 2. This query is not feasible since there is no path in the diagram which supports it. However, a "diagram-enabled" wrapper can determine close matches to this request and suggest $q_{t_1.t_3.t_5}$ which yields the desired output if the mediator can come up with a plan that provides the inputs $+state$ and $+radius$.

Another close match is $q_{t_1.t_3.t_6}$ which does not need an input value for *radius*. However, this path is not automatable since it involves an explicit user interaction $\mathsf{!ui}_3$. Thus, the first alternative is usually preferable (unless the meditor cannot provide the additional input parameters). $\square$

Note that when modeling a site with interaction diagrams, the designer of the diagram has to ensure that the binding patterns derivable from the diagrams correctly reflect what is being computed by the sources, i.e., their semantics. For example, a binding pattern $q(+a, +t, -b)$ could mean *"find books b where* *author=a or title=t"*, but it could also mean *"... where author=a and title=t"*, or even *"... where author$\neq$ a if title=t"*, etc. Dealing with these different possible semantics is beyond the scope of this paper and we just assume that attribute names and transitions between sources have been modeled accordingly.

## 4 Using Interactive Sources in the Mediator Framework

In the previous section we have defined formal properties of interaction diagrams and shown how they can be used to model complex sources involving multiple pages and sites. In this model, the capabilities of a source, i.e., the supported queries and interaction requirements, are characterized by the paths in the diagram. In the following, we explore how a wrapper using interaction diagrams can participate in the query evaluation process controlled by the mediator. We present different scenarios of interplay between the mediator and the wrapper, with increasingly complex dependence on the properties of the diagram.

**Schema Only.** First consider a scenario where the mediator is unaware of the binding patterns derivable from the interaction diagram. The mediator only keeps an account of the attribute names from a wrapper and passes a complete query to the wrapper. The wrapper uses the graph to determine if the query is supported. If the query is not feasible, the wrapper returns with an error, and the mediator tries to send a different rewrite of the query. In this case, the mediator does not have any knowledge of why the query failed, and hence cannot make any intelligent choice for the next rewrite. Although this approach makes the mediator-wrapper interactions very simple, this is potentially a very costly solution, and does not utilize any benefit of the interaction diagram.

**Summary Table.** In this scenario the mediator knows the role each attribute can play (input, output, both), but has no knowledge of the combination of attribute-role pairs that are permitted by the source. In this case the mediator maintains a summary table of the form

$$[\pm a, +b, \pm c, -d, +e, \ldots]$$

which *approximates* the capabilities of the source. The query $q(+a, +b, -c, -d)$ is supported according to this table and thus should have a set of equivalent paths by which the pattern should be satisfied. However, the source may actually support only the binding patterns $q_1(+a, +b, -c)$ and $q_2(+c, -d)$ and the mediator has to compute $q_1 \bowtie_c q_2$. To keep track of such situtations, [YLGMU99] precompute and maintain all possible binding patterns for the source at the mediator. Clearly, all such patterns can be automatically generated from the diagram, and registered with the mediator when the wrapper is first initiated. However, for a complex Web source this may be too large (in the worst case $3^n$ for $n$ attributes), and hence the solution does not scale well. We could reduce this number by eliminating subsumed binding patterns as in [YLGMU99]. In this case, one has to ensure that wrappable paths are prefered over non-wrappable ones. If the reduction by subsumption restricts the space of binding patterns to a manageable degree, then this is an acceptable solution. Since the mediator has complete knowledge of the wrapper's capabilities, it can be guaranteed to produce only feasible plans.

**Active Wrapper.** In case the previous solution still creates an unacceptable number of binding patterns at the mediator, the wrapper needs to take a more active part in query evaluation. Now, since the mediator does not have all the binding patterns, the *wrapper* has to evaluate the query by further decomposing it into subqueries. We sketch a method to accomplish this at the wrapper in the following way. Given a binding pattern $q$:

- The diagram-aware wrapper traverses the interaction diagram to find all paths that subsume the query. A path here corresponds to the execution plan defined in Section 3.
- If there is only one such path, the wrapper can immediately execute the query, since the mediator has no decisions to make in that query.
- If however there is more than one path, the wrapper has the choice to either send all of them to the mediator, or prune them by some local heuristic to reduce the number of viable execution plans. We have found the following pruning heuristic to be effective:
  - If there is a single path with no user interaction !ui, choose that path.
  - If there are multiple paths without !ui's, rank the paths by path length.
  - If all paths have one or more !ui's, rank the paths first by the length of the path, and then by the number of !ui's. The intuitive idea is first to reduce the number of pages to visit, and then reduce the number of forms or menus to fill in.
- Send the ranked list of paths to the mediator. The ranked list acts as a qualitative cost estimator for the execution plan.[5]

**Touting Wrapper.** Finally, consider a variant of the above scenario where the wrapper aids the mediator by providing additional hints (cf. Example 4). Again assume that the query is $q(+a, +b, -c, -d)$, but suppose the binding patterns supported by the source are $q_1(+a, +b, -c)$ and $q_2(+c, +e, -d)$. The query will obviously fail, but the mediator does not know that the query would have succeeded if attribute $e$ had been provided.

There are two possible outcomes if the wrapper sends back this hint to the mediator. First, the mediator may already have a constraint on $e$, but this constraint was never passed to the wrapper, because the mediator found a second source that also uses $e$, and hence cleanly separated the attributes between sources. In this case, the mediator needs to rewrite the query by reusing the attribute $e$ for the first source also. In the second case, the original query is

[5] The mediator may also cache the alternatives returned to reduce the number of interactions with the wrapper in subsequent queries.

under-specified and the mediator returns to the user with a list of missing attributes, thus making the query evaluation more cooperative. To implement this solution we can modify the above method with a parameter $k$, such that only paths with up to $k$ missing attributes will be searched.

## 5 Conclusion and Outlook

We have proposed a method for modeling the capabilities of complex interactive Web sources using interaction diagrams and have shown how the standard input elements of Web sources (forms, menus, etc.) can be modeled as restricted relational queries with a certain input/output pattern. These elements are the basis for our formal model of interaction diagrams, where they are used to specify the interaction requirements of transitions between sources (=Web pages having the same input/output schema). A "diagram-enabled" wrapper can chain together transitions, thereby supporting more complex queries than those provided by the individual subsources. This is possible because the query capabilities and interaction requirements (i.e., plans for executing the query) of *paths* of transitions can be *automatically derived* from a diagram.[6] In particular, this allows to examine alternative execution paths and determine those with *minimal user interaction*.

The last two scenarios discussed in the previous section demonstrate the additional value of the diagram-based representation of wrapper capbilities in the mediator framework. It also shows a departure from the more commonly used thin-wrapper heavy-mediator model to a more negotiation-oriented interoperation between the two components. Such a negotiation will be useful as we move from modeling simple Web sources whose capabilities can be manually modeled to more complex, dynamic Web sources, for which creating a static exhaustive set of capabilities is impractical.

We plan to integrate our approach into the MIX[7] mediator system, which employs a virtual integration approach where queries are decomposed at runtime and sent to the source wrappers. Additionally, query evaluation in the MIX mediator system is *lazy* (or *on-demand*), i.e., driven by the clients navigation into the virtual answer view [LPV99]. Interestingly, by incorporating the proposed diagrams into this mediator architecture, user interactions and query evaluation become mutually dependent: When the user issues a query or navigates into a view, the mediator decomposes the request and send subqueries to the sources. Some of these queries may be infeasible without additional user interaction. In these cases, the source "calls back" the user and requests additional input before query evaluation can proceed.

***Acknowledgments.*** The first author thanks Birgitta König-Ries for numerous and detailed comments on the paper.

[6] A first prototype for analysing transition paths has been implemented.
[7] ***M****ediation of* ***I****nformation using* ***XML*** [MIX99,BGL$^+$99]

## References

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *Intl. Conference on Very Large Data Bases (VLDB)*, 1997.

[BGL$^+$99] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-Based Information Mediation with MIX. In *ACM Intl. Conference on Management of Data (SIGMOD)*, Philadelphia, 1999. (exhibition program).

[BK94] A. J. Bonner and M. Kifer. An Overview of Transaction Logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[DFKR99] H. Davulcu, J. Freire, M. Kifer, and I. Ramakrishnan. A Layered Architecture for Querying Dynamic Web Content. In *ACM Intl. Conference on Management of Data (SIGMOD)*, Philadelphia, 1999.

[FLM98] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3), September 1998.

[GMLY99] H. Garcia-Molina, W. Labio, and R. Yerneni. Capability-Sensitive Query Processing on Internet Sources. In *Intl. Conference on Data Engineering (ICDE)*, pp. 50–59, 1999.

[LHL$^+$98] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. *Information Systems*, 23(8):589–613, 1998.

[LPV99] B. Ludäscher, Y. Papakonstantinou, and P. Velikhov. A Framework for Navigation-Driven Lazy Mediators. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, Philadelphia, 1999.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Intl. Conference on Very Large Data Bases (VLDB)*, pp. 251–262, 1996.

[MAM$^+$98] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. The Araneus Web-Base Management System. In *ACM Intl. Conference on Management of Data (SIGMOD)*, pp. 544–546, 1998. (exhibition program).

[MIX99] Mediation of Information using XML (MIX). `www.npaci.edu/DICE/MIX/` and `www.db.ucsd.edu/Projects/MIX/`, 1999.

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. D. Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, pp. 161–186, 1995.

[PGH98] Y. Papakonstantinou, A. Gupta, and L. M. Haas. Capabilities-Based Query Rewriting in Mediator Systems. *Distributed and Parallel Databases*, 6(1):73–110, 1998.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[YLGMU99] R. Yerneni, C. Li, H. Garcia-Molina, and J. Ullman. Computing Capabilities of Mediators. In *ACM Intl. Conference on Management of Data (SIGMOD)*, Philadelphia, PA, 1999.