# A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration

Eduard Constantin Dragut[*], Thomas Kabisch[+], Clement Yu[*], Ulf Leser[+]

[*]University of Illinois at Chicago

[+]Humboldt-Universität zu Berlin

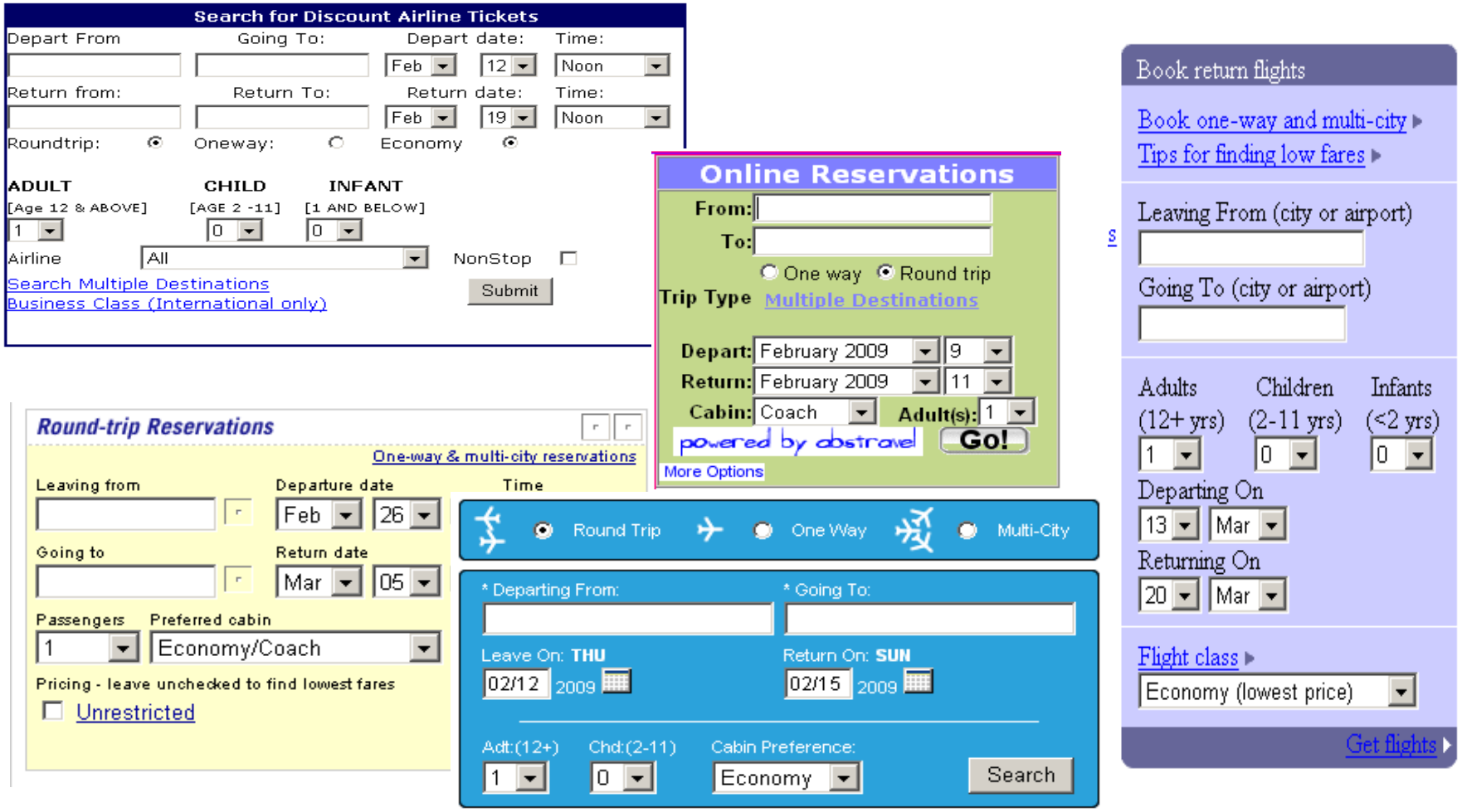
35th Intl. Conference on Very Large Data Bases

Lyon, France, August 2009

# Outline

1. **Motivation**
   - Integration of Deep Web Sources
   - Hierarchical Schema Model
2. Concepts
3. Observations on Query Interfaces
4. Extraction Algorithm
5. Evaluation
6. Conclusion and Outlook

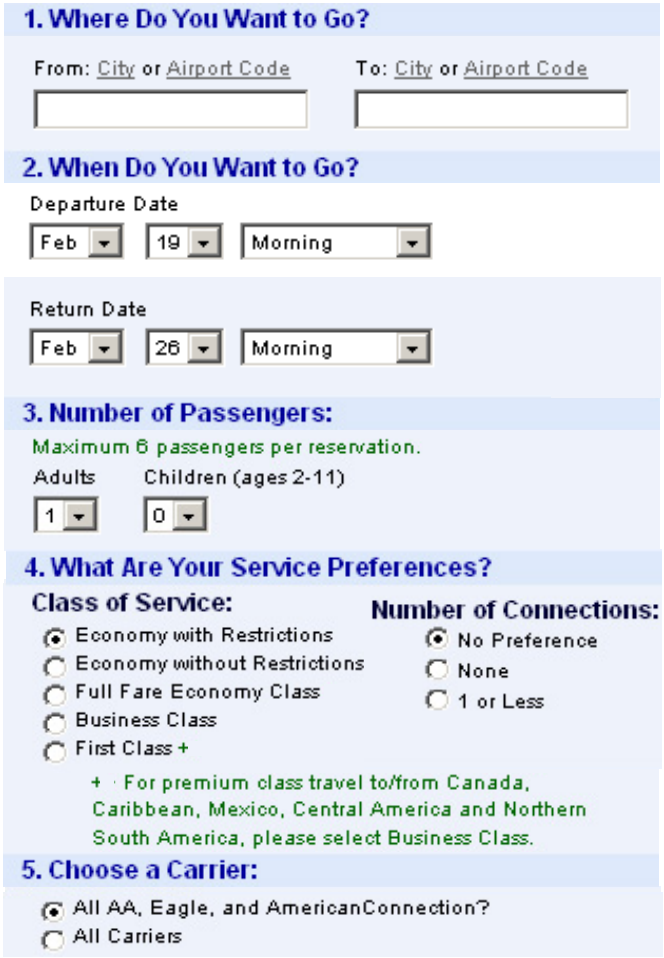
# Motivation (I): Deep Web Integration

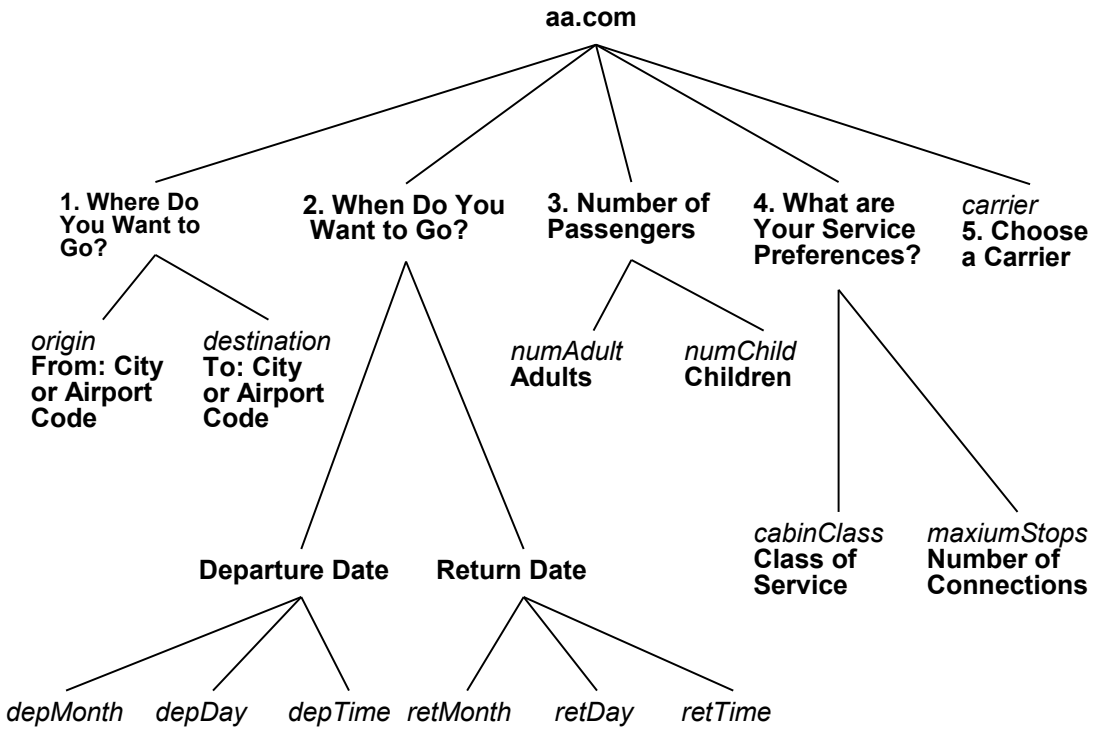


# Motivation (II): Deep Web Integration Process

1. **Extraction of Source Interface Schemata**
2. Derivation of integrated Interface Schema
3. Formulation of Queries
4. Transformation of Queries
5. Integration of Results

# Motivation (III): Extraction of Interface Schema Trees



# Motivation (IV): Hierarchical Representation

## Prior Work [BEP04, WISE05]

- flat representation
- no order
- no context

## Improvements with ordered Schema Trees

### Query Interface Matching

- handling of unlabelled fields
- resolving of homonymy problem
- identifying complex (1:n) field-mappings

### Deep Web Crawling

- enrich annotations to fields
- identify interrelated fields

# Outline

1. Motivation

**2. Concepts**

- **Tokens**
- **Fields**
- **Labels**

3. Observations on Query Interfaces

4. Extraction Algorithm

5. Evaluation

6. Conclusion and Outlook

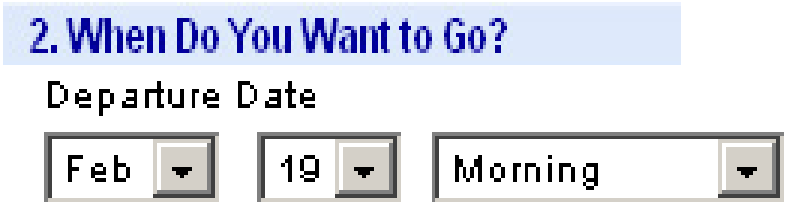
# Concepts (I): Tokens and Bounding Boxes

## Token

- Def: visible piece of HTML code with layout properties
- Attributes: style Information, bounding box, content
- Types: text token, field token, image token

## Bounding Box

- rectangular area that describes position and size of token
- pixel coordinates given by browser rendering engine



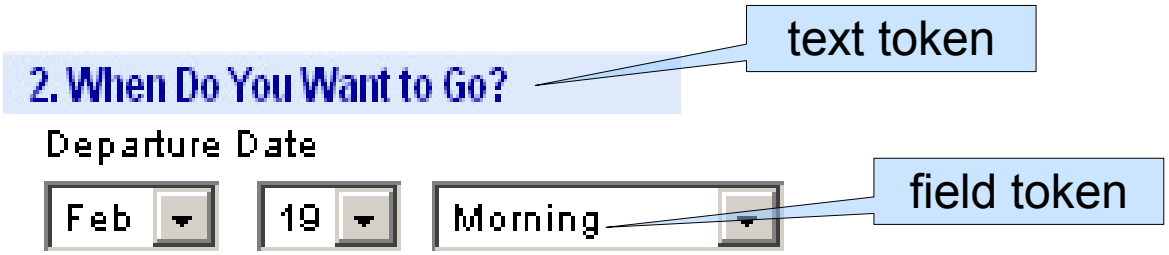
# Concepts (I): Tokens and Bounding Boxes

## Token

- Def: visible piece of HTML code with layout properties
- Attributes: style Information, bounding box, content
- Types: text token, field token, image token

## Bounding Box

- rectangular area that describes position and size of token
- pixel coordinates given by browser rendering engine



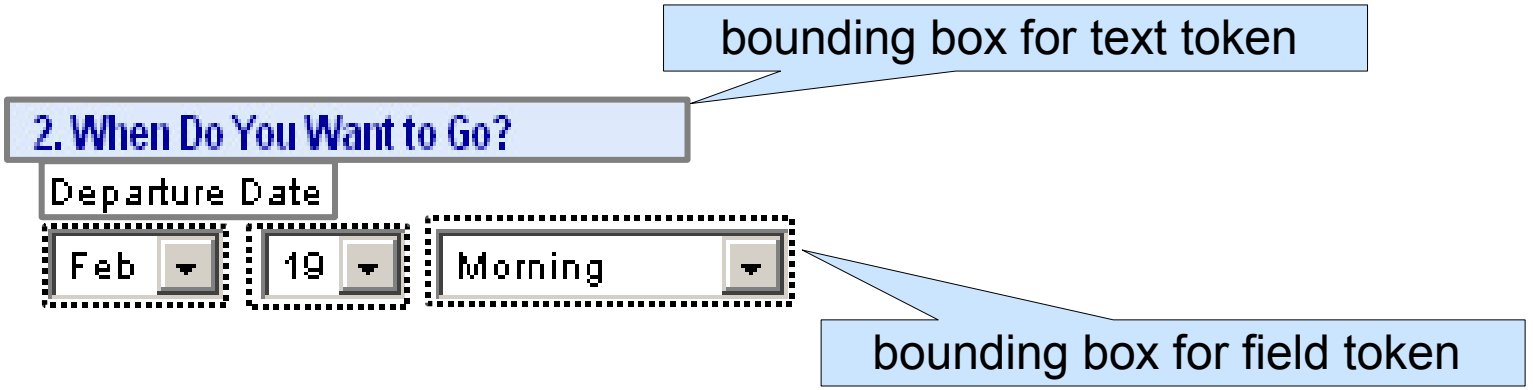
# Concepts (I): Tokens and Bounding Boxes

## Token

- Def: visible piece of HTML code with layout properties
- Attributes: style Information, bounding box, content
- Types: text token, field token, image token

## Bounding Box

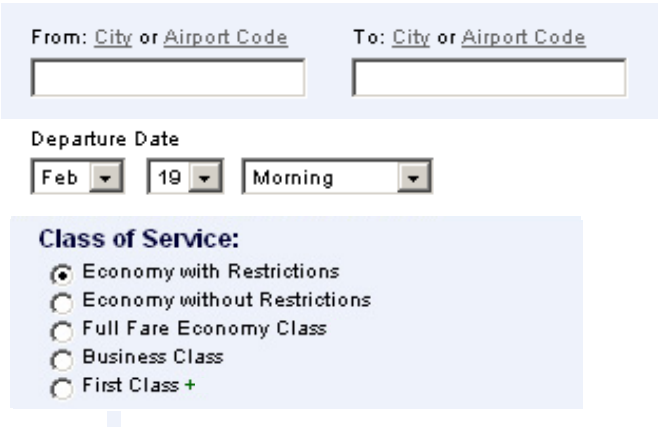
- rectangular area that describes position and size of token
- pixel coordinates given by browser rendering engine



# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values



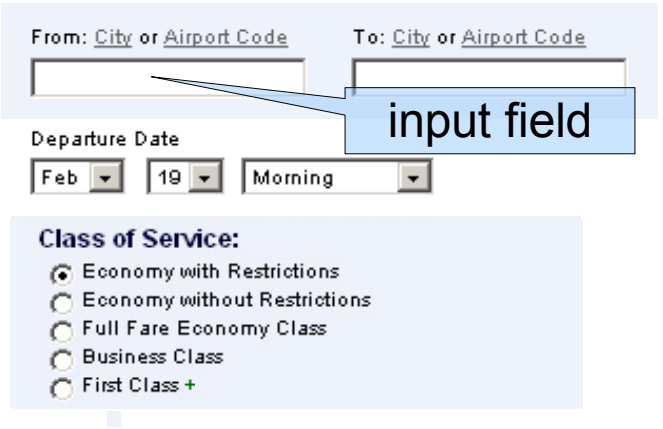

## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values



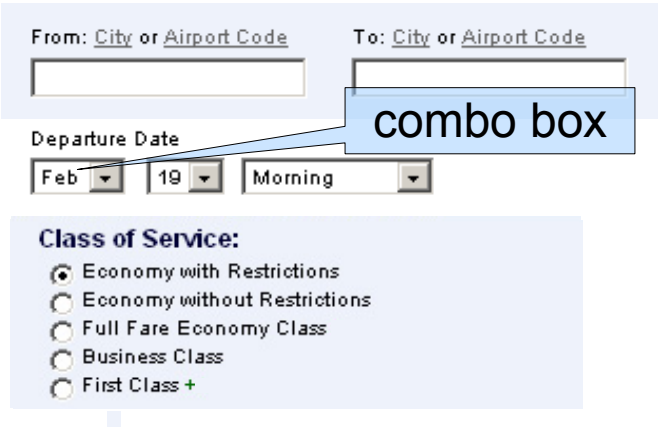

## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values




## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values

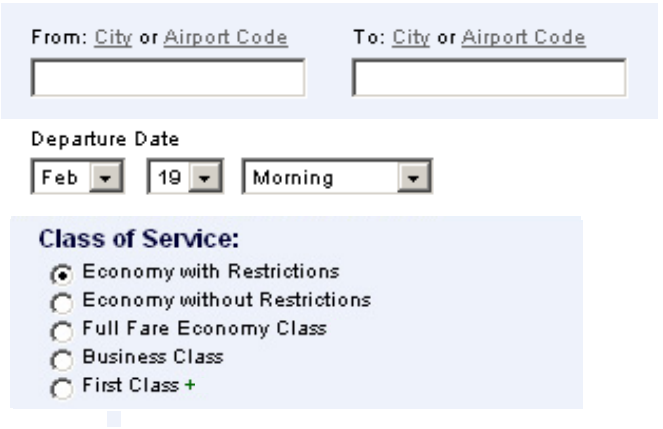

## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values



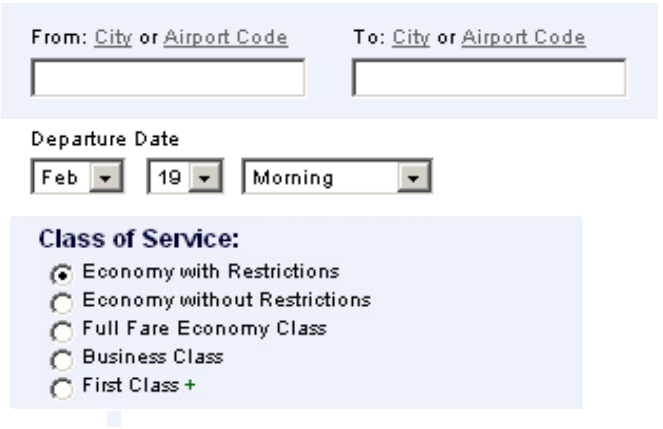

## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values



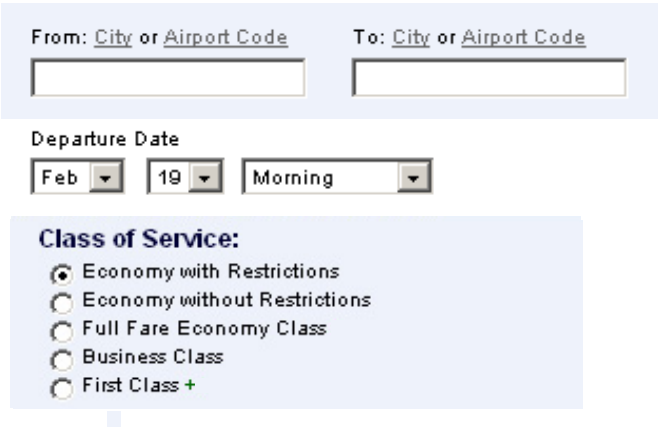

## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation

# Concepts (II): Fields and Labels

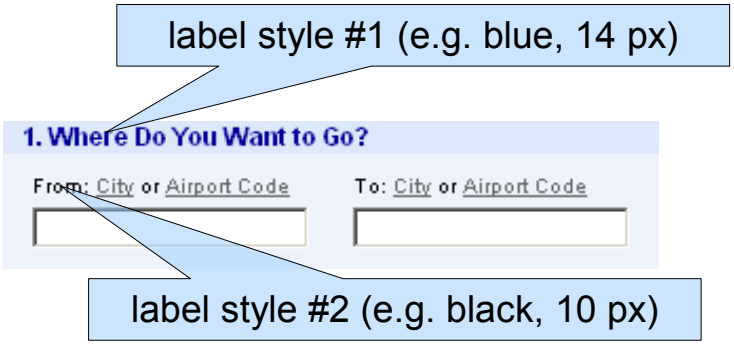
## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values




## Label

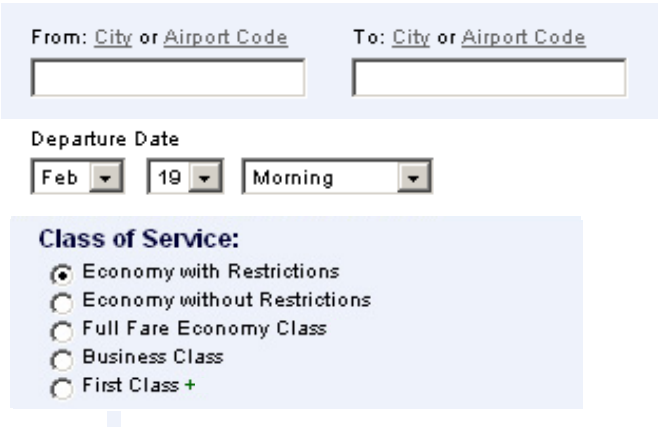
- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation



# Concepts (II): Fields and Labels

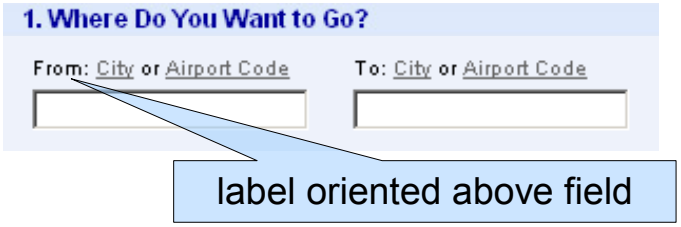
## Field

- Def: interactive HTML form elements
- Important Types: input field, combo box, radio button...
- Attributes: name, values, default values




## Label

- Def: texts denoting fields or groups of fields
- Types: field label, group label
- Properties: style, orientation



# Outline

1. Motivation

2. Concepts

3. **Observations on Query Interfaces**
   - **Order and Alignment**
   - **Field Scope**
   - **Label Scope**

4. Extraction Algorithm

5. Evaluation

6. Conclusion and Outlook

# Observations (I): Order and Alignment of Fields

## Order

- defined by their fill-in order
- given by designer (tab-order,code order)


## Alignment

- hint for semantic relation
- horizontal / vertical
- between consecutive fields only

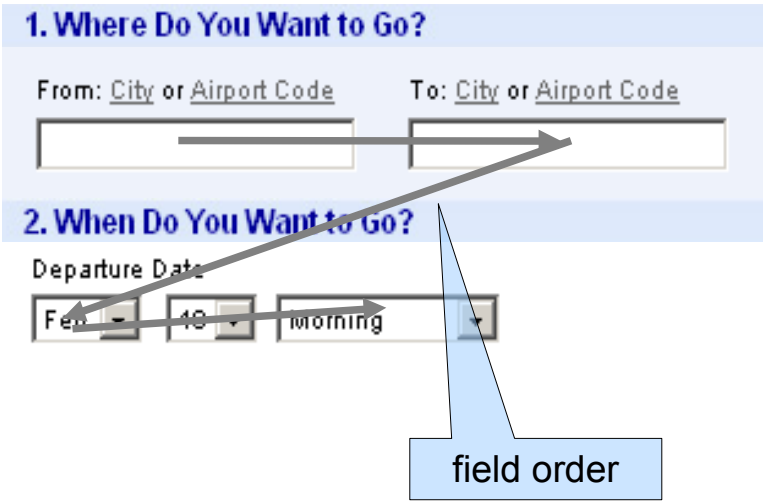
# Observations (I): Order and Alignment of Fields

## Order

- defined by their fill-in order
- given by designer (tab-order,code order)

## Alignment

- hint for semantic relation
- horizontal / vertical
- between consecutive fields only



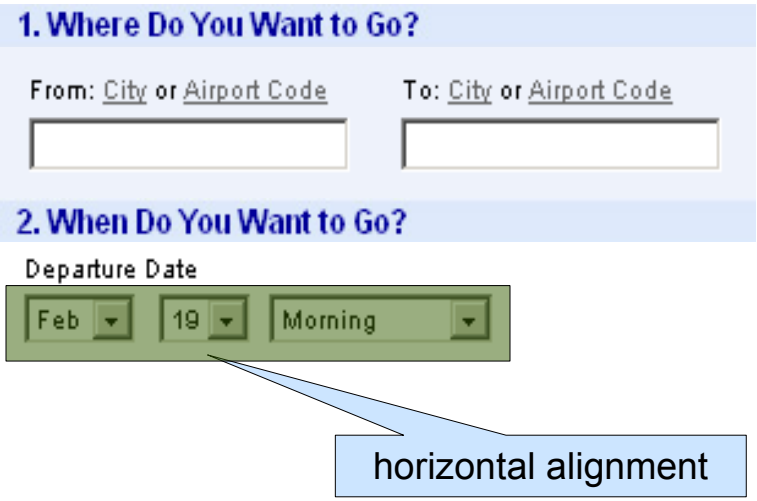
# Observations (I): Order and Alignment of Fields

## Order

- defined by their fill-in order
- given by designer (tab-order,code order)

## Alignment

- hint for semantic relation
- horizontal / vertical
- between consecutive fields only



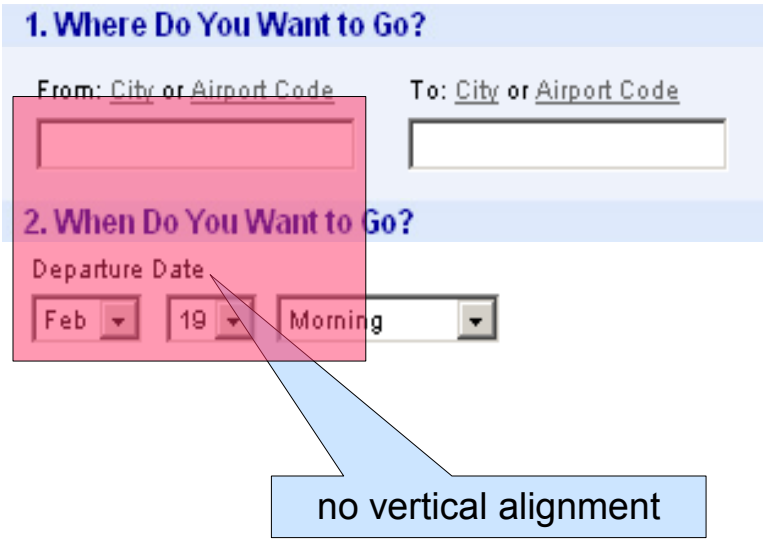
# Observations (I): Order and Alignment of Fields

## Order

- defined by their fill-in order
- given by designer (tab-order,code order)

## Alignment

- hint for semantic relation
- horizontal / vertical
- between consecutive fields only



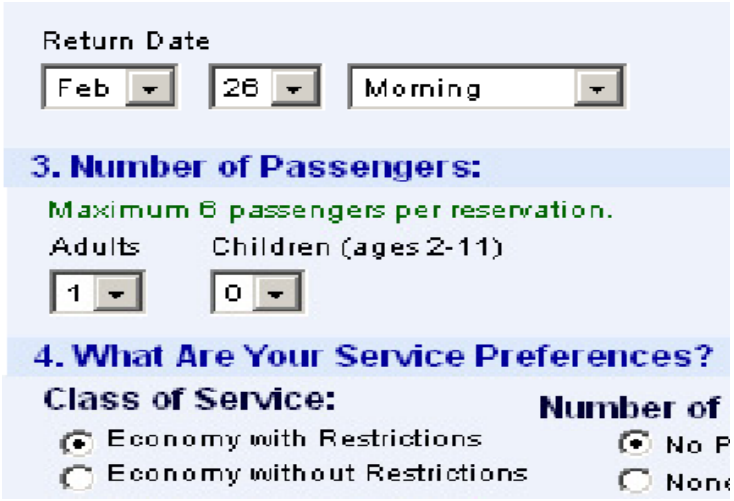
# Observations (II): Field Scope

## Field Scope

- area where a field candidate label may be placed
- contains four scope sectors

## Scope Sector

- ranges from field to the left, to the right, upwards or downwards
- bound by other fields or interface boundary



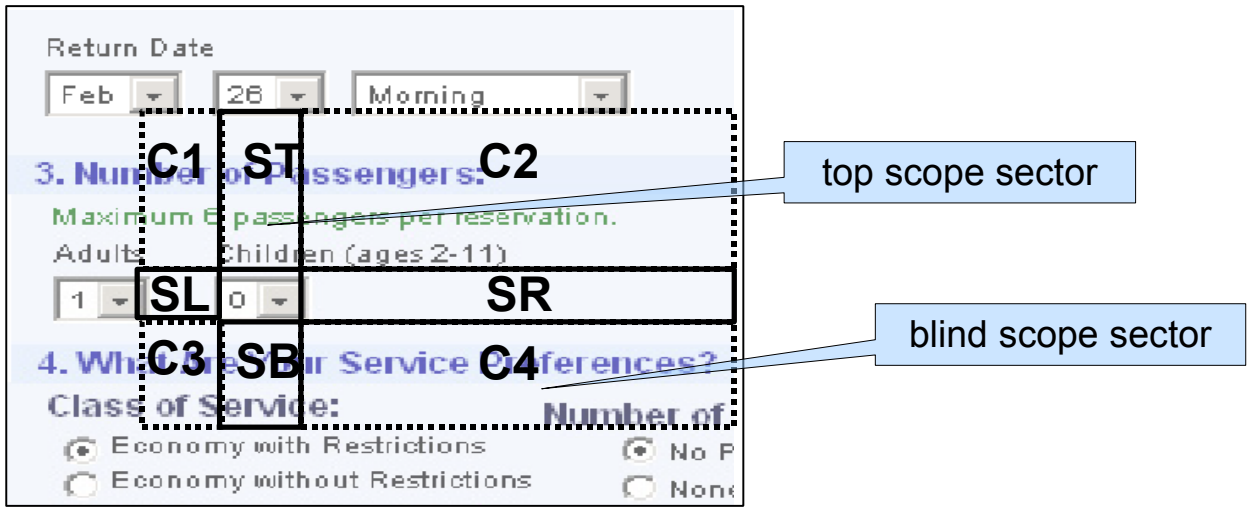
# Observations (II): Field Scope

## Field Scope

- area where a field candidate label may be placed
- contains four scope sectors

## Scope Sector

- ranges from field to the left, to the right, upwards or downwards
- bound by other fields or interface boundary



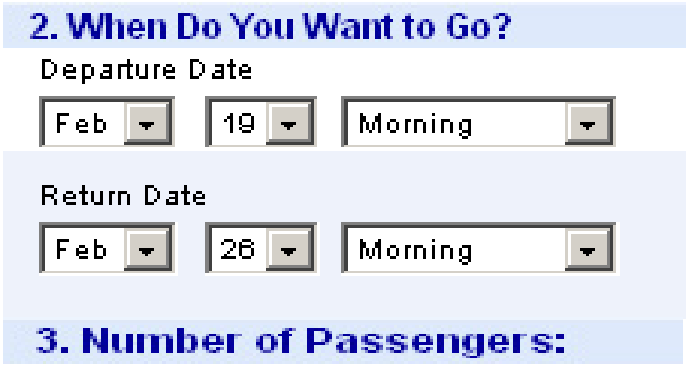
# Observations (III): Label Scope

## Intuition

- query interfaces are organized top-down and left-rigth
- labels have area where they are visible

## Definition

- rectangular area ranging from the (top,left)-corner of the label bounding box to the right and to the bottom until the next label of the same text style or the interface boundary
- never intersect other label's scope



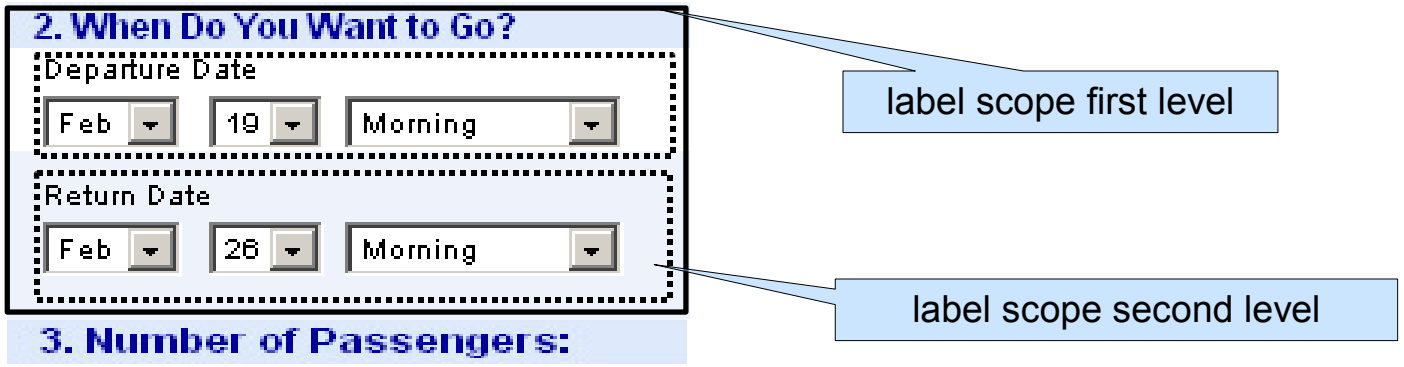
# Observations (III): Label Scope

## Intuition

- query interfaces are organized top-down and left-rigth
- labels have area where they are visible

## Definition

- rectangular area ranging from the (top,left)-corner of the label bounding box to the right and to the bottom until the next label of the same text style or the interface boundary
- never intersect other label's scope



# Outline

1. Motivation

2. Concepts

3. Observations on Query Interfaces

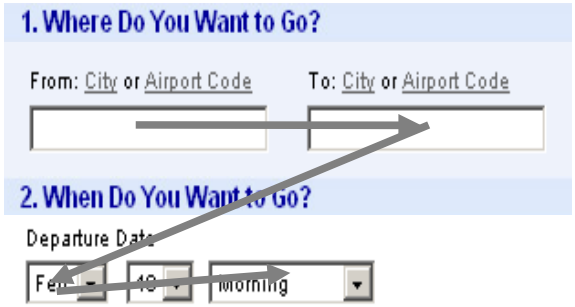
4. **Extraction Algorithm**
   - **Field Tree Construction**
   - **Label Tree Construction**
   - **Tree Integration**

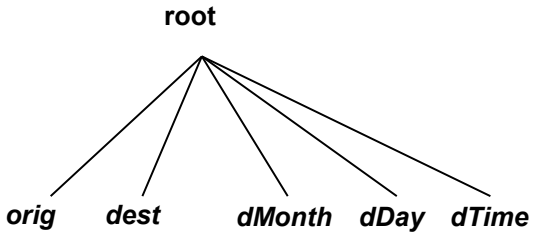
5. Evaluation

6. Conclusion and Outlook

# Algorithm (I): Tree of Fields Construction

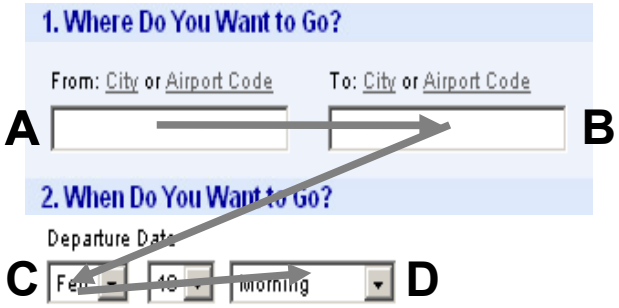
1. **add for each field a node into the empty tree *FT***
2. estimate inflection points of field order
3. group all fields between two consecutive inflection points
4. insert an internal node *N* for each group of fields into tree of fields *FT* such that *N* becomes the direct parent node of all field nodes in the group

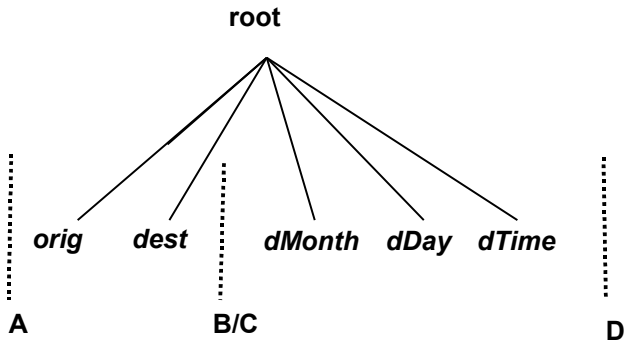


# Algorithm (I): Tree of Fields Construction

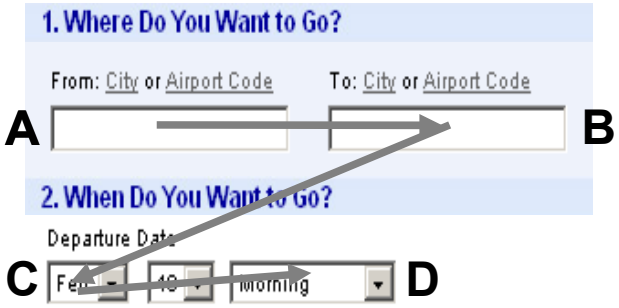
1. add for each field a node into the empty tree *FT*
2. **estimate inflection points of field order**
3. group all fields between two consecutive inflection points
4. insert an internal node *N* for each group of fields into tree of fields *FT* such that *N* becomes the direct parent node of all field nodes in the group

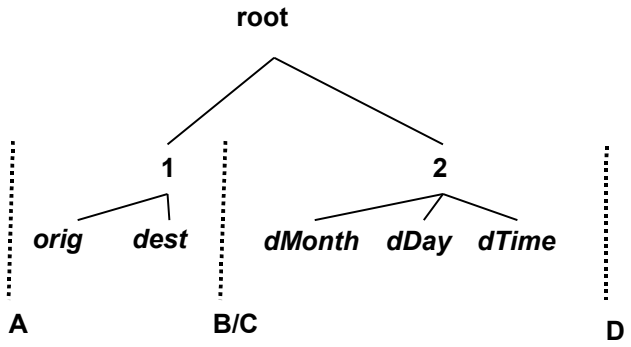


# Algorithm (I): Tree of Fields Construction

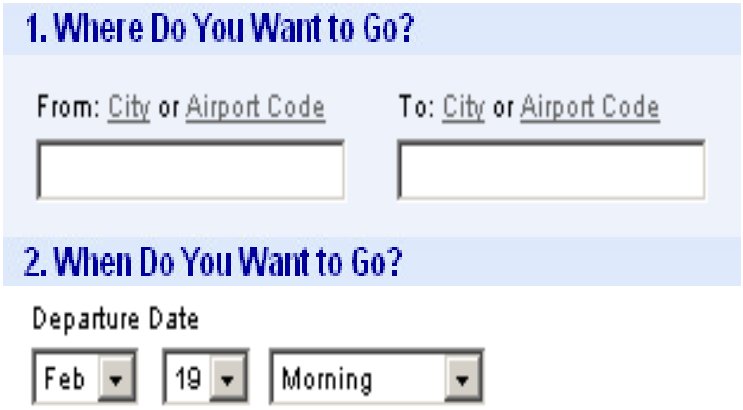
1. add for each field a node into the empty tree *FT*
2. estimate inflection points of field order
3. **group all fields between two consecutive inflection points and insert an internal node *N* for each group into *FT* such that *N* becomes parent node of all field nodes in the group**



# Algorithm (II): Tree of Labels Construction

1. initialize label scope
2. get top cluster: smallest cluster of labels that covers all fields of current window
3. insert node for each element of top cluster as children of current root
4. recursively repeat for next lower cluster of labels



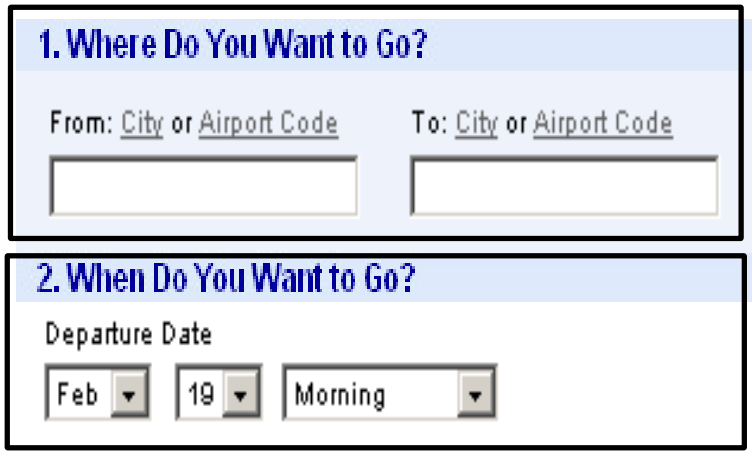
# Algorithm (II): Tree of Labels Construction

**1.initialize label scope**

2.get top cluster: smallest cluster of labels that covers all fields of current window

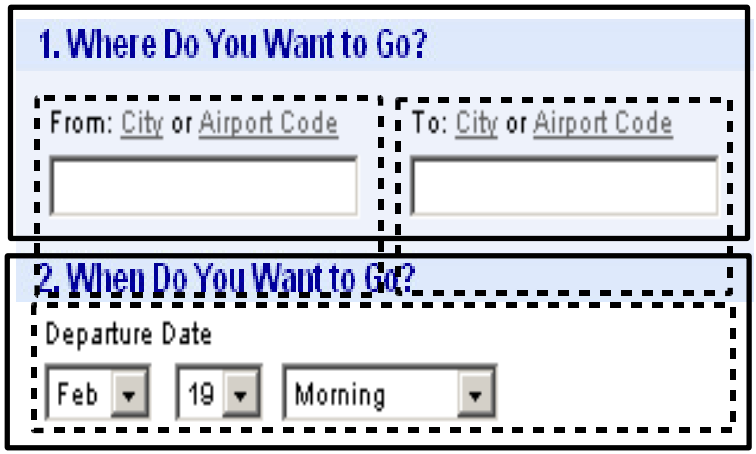
3.insert node for each element of top cluster as children of current root

4.recursively repeat for next lower cluster of labels



# Algorithm (II): Tree of Labels Construction

1. **initialize label scope**
2. get top cluster: smallest cluster of labels that covers all fields of current window
3. insert node for each element of top cluster as children of current root
4. recursively repeat for next lower cluster of labels



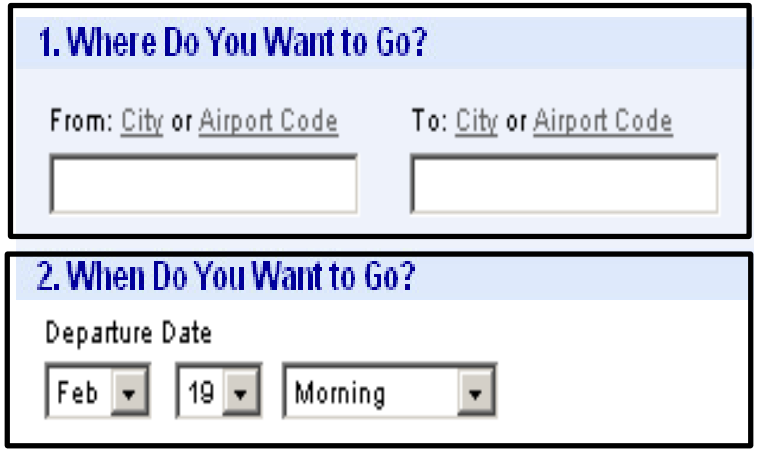
# Algorithm (II): Tree of Labels Construction

1.initialize label scope

**2.get top cluster: smallest cluster of labels that covers all fields of current window**

3.insert node for each element of top cluster as children of current root

4.recursively repeat for next lower cluster of labels



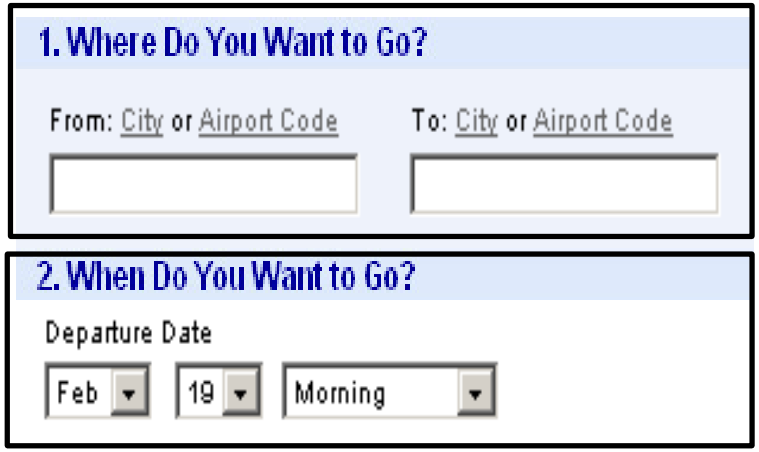
# Algorithm (II): Tree of Labels Construction

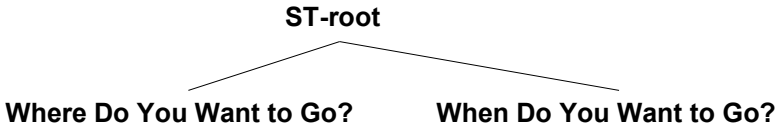
1.initialize label scope

2.estimate top cluster: smallest cluster of labels that covers all fields of current window

**3.insert node for each element of top cluster as children of current root**

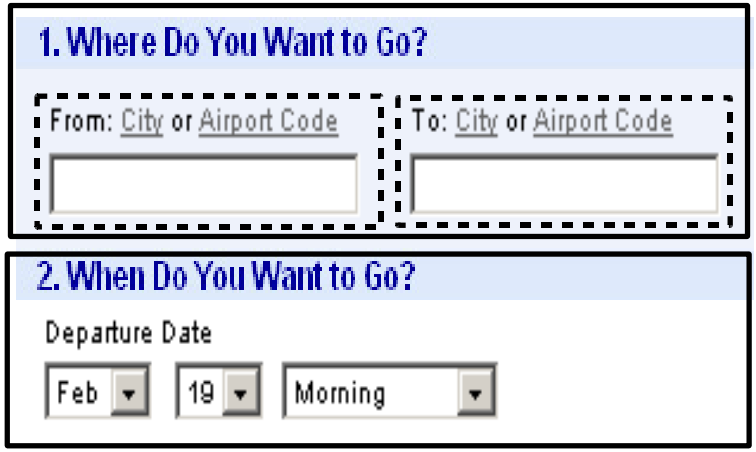
4.recursively repeat for next lower cluster of labels

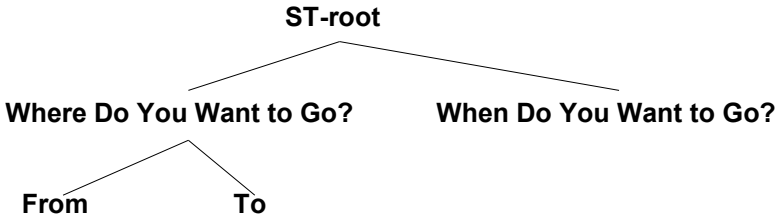


# Algorithm (II): Tree of Labels Construction

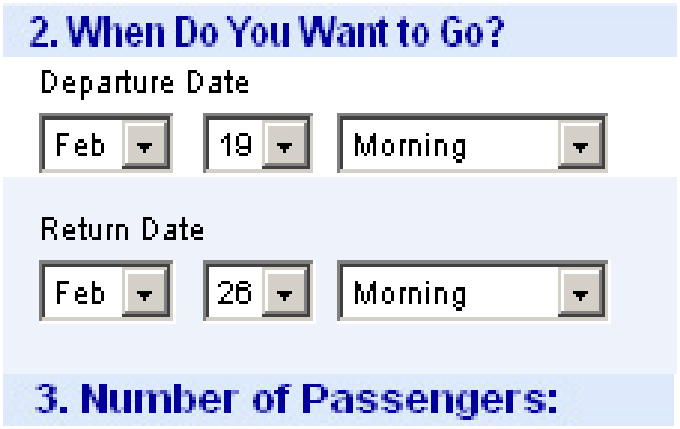
1. initialize label scope
2. estimate top cluster: smallest cluster of labels that covers all fields of current window
3. insert node for each element of top cluster as children of current root
4. **recursively repeat for next lower cluster of labels**



# Algorithm (III): Integration of Trees

1. calculate for a label *l* the list of covered fields *Df* based on the label scope of *l*
2. identify a node *lca* in tree of fields *FT* such that *lca* is the lowest common ancestor of all the fields in *Df and lca* does not have any other descending leaves
3. map label to *lca*, distinguish three cases:
   - if a node *lca* can be identified in *FT* map *l* to *lca, l* becomes label of *lca*
   - if there is no such node and no group conflict insert *lca* into *FT*
   - if there is a group conflict resolve conflict and insert *lca* into *FT*



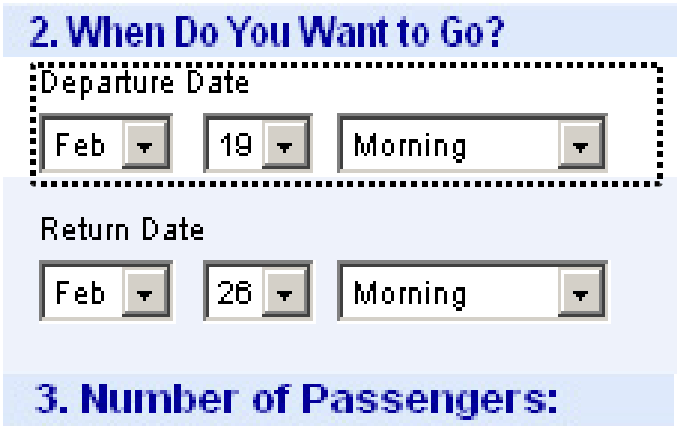
# Algorithm (III): Integration of Trees

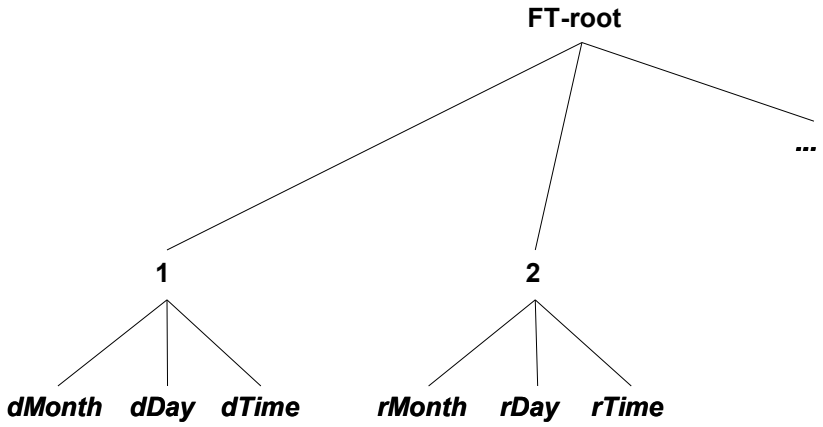
**1.calculate for a label *l* the list of covered fields *Df* based on the label scope of *l***

2.identify a node *lca* in tree of fields *FT* such that *lca* is the lowest common ancestor of all the fields in *Df and lca* does not have any other descending leaves

3.map label to *lca*, distinguish three cases:

- if a node *lca* can be identified in *FT* map *l* to *lca, l* becomes label of *lca*
- if there is no such node and no group conflict insert *lca* into *FT*
- if there is a group conflict resolve conflict and insert *lca* into *FT*



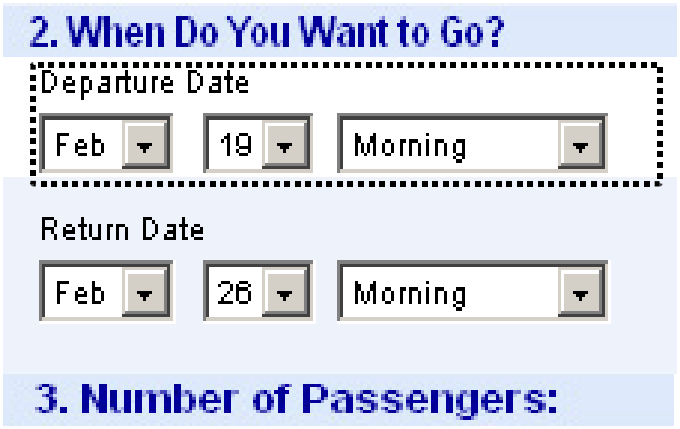
# Algorithm (III): Integration of Trees

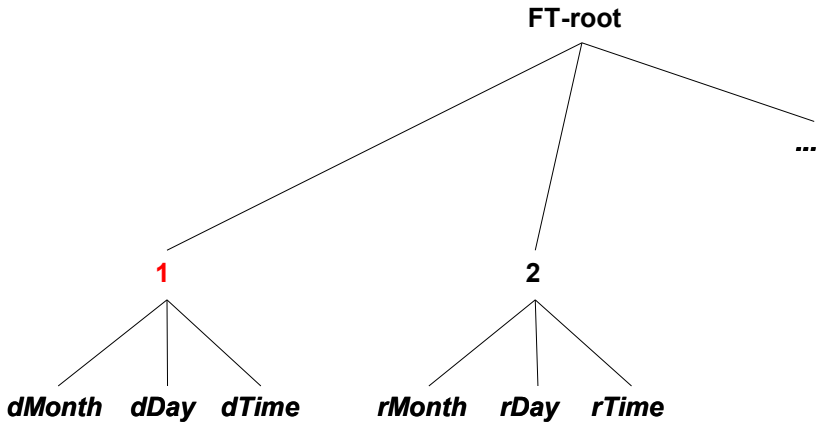
1.calculate for a label *l* the list of covered fields *Df* based on the label scope of *l*

**2.identify a node *lca* in tree of fields *FT* such that *lca* is the lowest common ancestor of all the fields in *Df and lca* does not have any other descending leaves**

3.map label to *lca*, distinguish three cases:

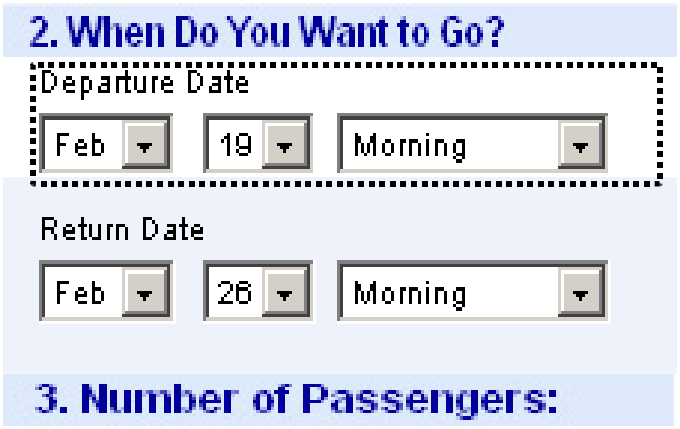
- if a node *lca* can be identified in *FT* map *l* to *lca, l* becomes label of *lca*
- if there is no such node and no group conflict insert *lca* into *FT*
- if there is a group conflict resolve conflict and insert *lca* into *FT*

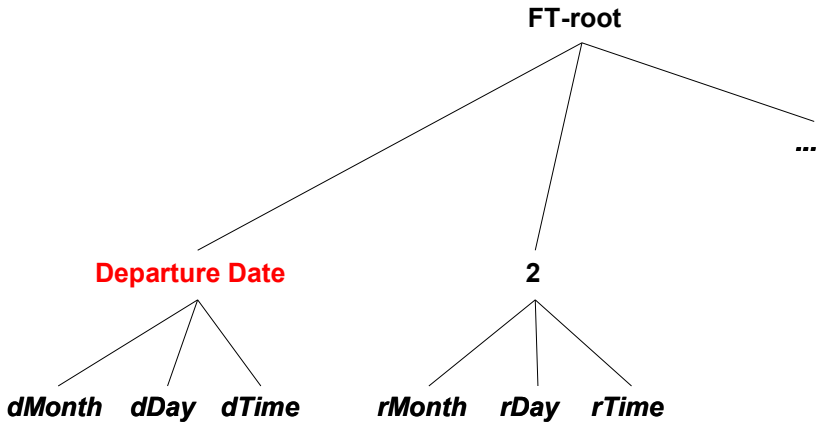


# Algorithm (III): Integration of Trees

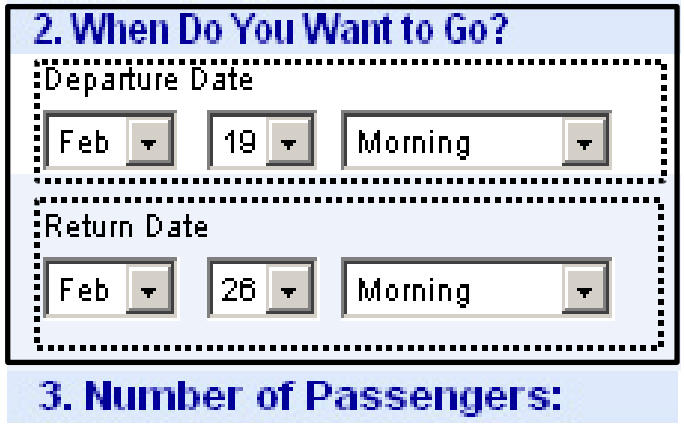
1. calculate for a label *l* the list of covered fields *Df* based on the label scope of *l*
2. identify a node *lca* in tree of fields *FT* such that *lca* is the lowest common ancestor of all the fields in *Df and lca* does not have any other descending leaves
3. **map label to *lca*, distinguish three cases:**
   - **if a node *lca* can be identified in *FT* map *l* to *lca, l* becomes label of *lca***
   - if there is no such node and no group conflict insert *lca* into *FT*
   - if there is a group conflict resolve conflict and insert *lca* into *FT*

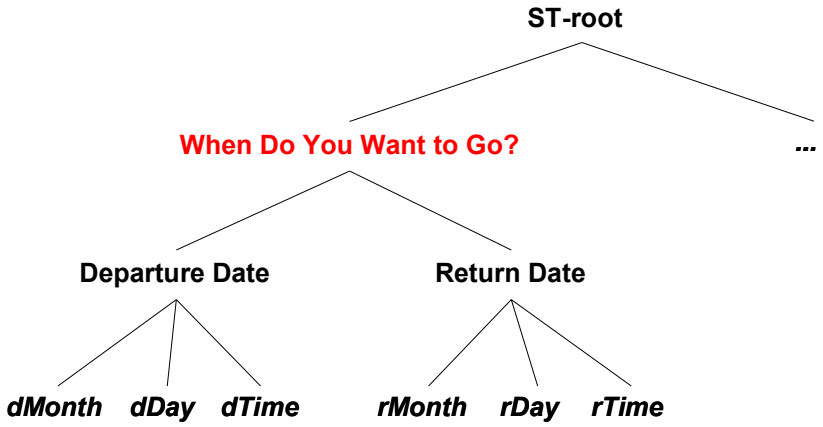


# Algorithm (III): Integration of Trees

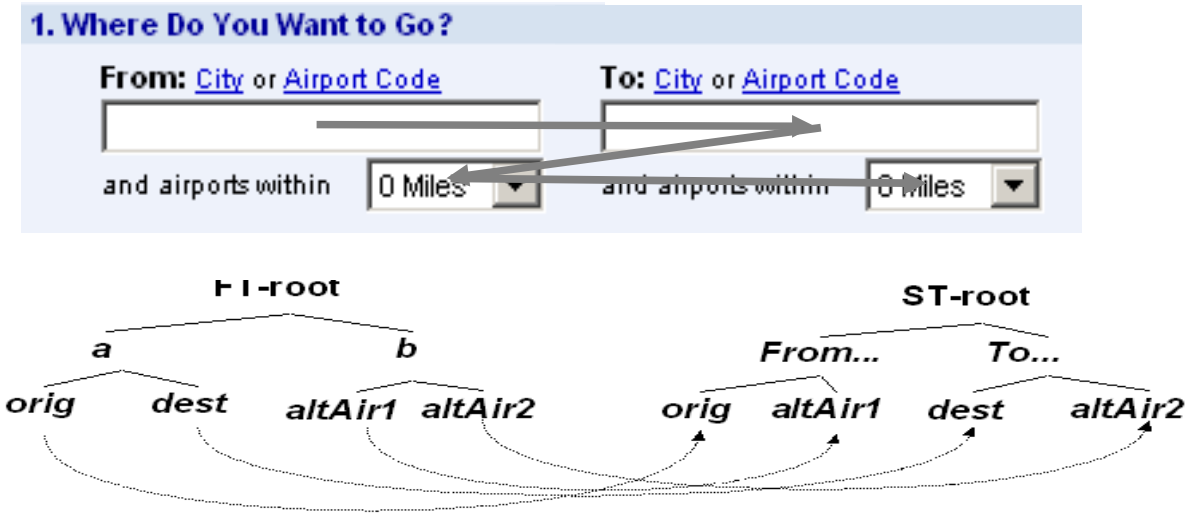
1. calculate for a label *l* the list of covered fields *Df* based on the label scope of *l*
2. identify a node *lca* in tree of fields *FT* such that *lca* is the lowest common ancestor of all the fields in *Df and lca* does not have any other descending leaves
3. **map label to *lca*, distinguish three cases:**
   - if a node *lca* can be identified in *FT* map *l* to *lca, l* becomes label of *lca*
   - **if there is no such node and no group conflict insert *lca* into *FT***
   - if there is a group conflict resolve conflict and insert *lca* into *FT*



# Algorithm (III): Integration / Conflict Resolution

- If field tree and label tree grouping conflict, the label tree grouping is given priority
- grouping in field tree may be design error (see example)




Field Tree

Schema Tree

# Outline

1. Motivation

2. Concepts

3. Observations on Query Interfaces

4. Extraction Algorithm

5. **Evaluation**
   - **Data Sets**
   - **Results**

6. Conclusion and Outlook

# Evaluation (I): Datasets

## ICQ

- totals 100 query interfaces of five domains (e.g. airlines, real estate)
- each domain contains 20 interfaces
- collected to evaluate matching techiques [CLUST04]
- provides gold standard trees
- more complex, avg. number of internal nodes in gold standard 2.35

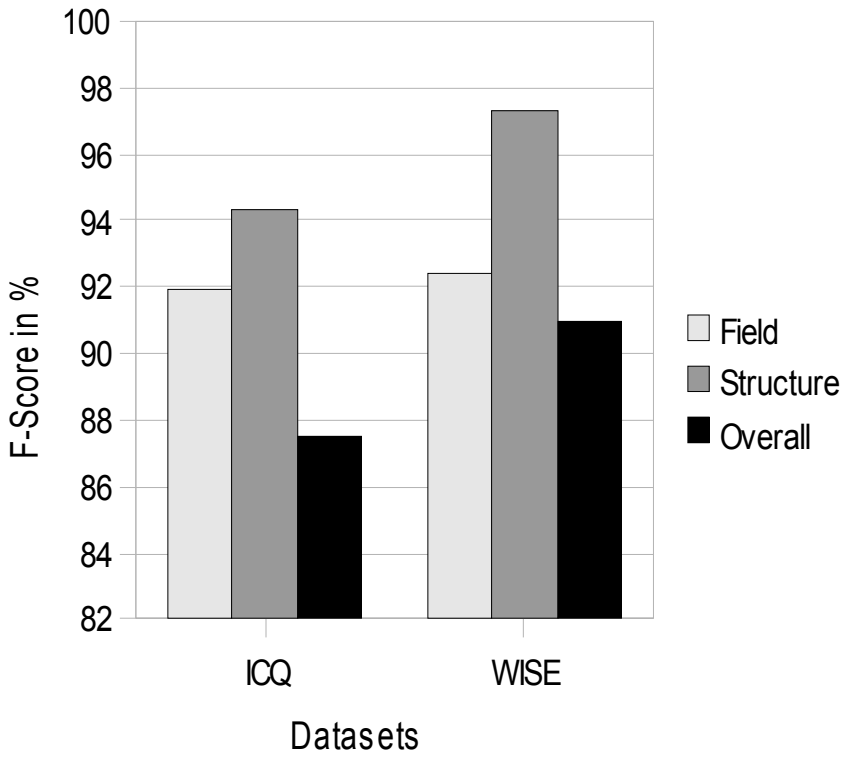
## WISE

- 147 (135 accessible) interfaces of seven domains (e.g. books, watches)
- collected to evaluate WISE Extractor [WISE05]
- gold standard trees derived manually
- less complex, avg. number of internal nodes in gold standard 1.57

## Tel8

- originally 487 interfaces of eight domains, about 50% no more accessible
- collected for extraction technique [BEP04]
- provides gold standard in flat representation, no trees

# Evaluation (II): Results & Measures




## Field

- how well are the labels for leaves extracted
- accuracy given by ratio of number of correctly labelled fields by number of fields

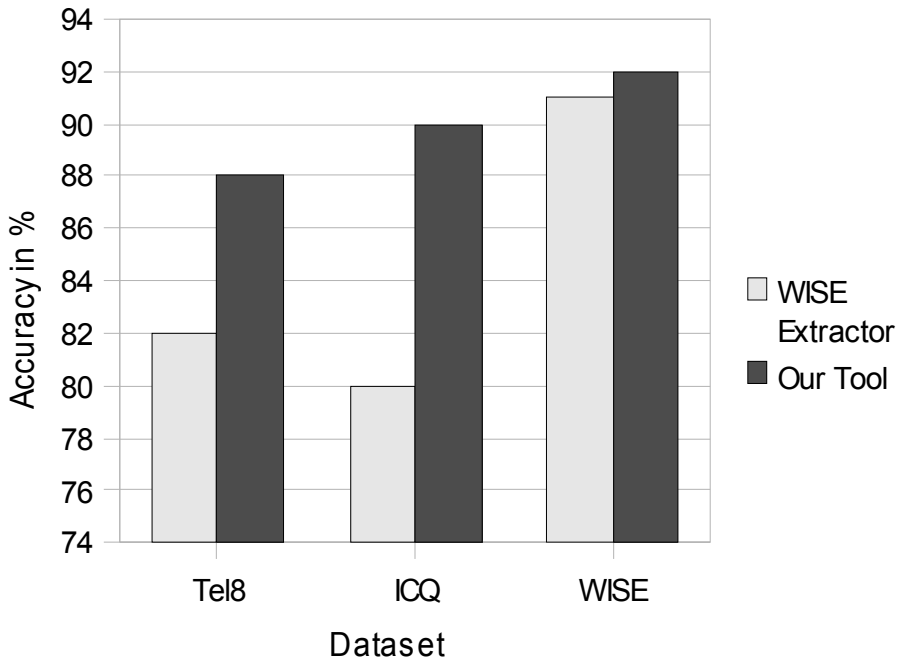
## Structure

- how well the groups of fields are identified
- how good the order of fields is obtained
- uses structural tree edit distance
- ignores labels

## Overall

- how well the trees and their labels are extracted
- utilizes tree edit distance

# Evaluation (III) : Comparison to WISE Extractor




## Node Accuracy Measure

- differs from metrics before due to comparability
- ratio of number of fields and internal nodes correctly labelled by the number of nodes
- averaged over all interfaces of dataset

# Outline

1. Motivation
2. Concepts
3. Observations on Query Interfaces
4. Extraction Algorithm
5. Evaluation
6. **Conclusion and Outlook**

# Conclusion and Outlook

## Conclusion

- query interface extraction technique for Deep Web integration
- visual approach, independent of HTML implementation issues
- represents query interface schemas hierarchically
- parallel to document headings design
- utilizes integration techiques to construct schema tree

## Outlook

- handling of images
- integration of semantic tools such as WordNet

# References

[WISE05]
Hai He, Weiyi Meng, Clement T. Yu, and Zonghuan Wu. Constructing interface schemas for search interfaces of Web databases. In WISE, 2005.

[BEP04]
Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In SIGMOD, 2004.

[CLUST04]
Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In SIGMOD, 2004.