Part II: Document Type Definition

Imposing structure on XML documents

Document Type Descriptors

- Document Type Descriptors (DTDs) impose structure on an XML document.
- There is *some* relationship between a DTD and a schema, but it is not close – there is still a need for additional "typing" systems.
- The DTD is a *syntactic* specification.

Example: An Address Book

```
<person>
    <name> MacNiel, John </name>            } Exactly one name
    <greet> Dr. John MacNiel </greet>       } At most one greeting
    <addr>1234 Huron Street </addr>         } As many address lines
    <addr> Rome, OH 98765 </addr>           } as needed (in order)
    <tel> (321) 786 2543 </tel>             }
    <fax> (321) 786 2543 </fax>             } Mixed telephones
    <tel> (321) 786 2543 </tel>             } and faxes
    <email> jm@abc.com </email>             } As many
                                              as needed
</person>
```

Specifying the structure

• name	to specify a name element
• greet?	to specify an optional (0 or 1) greet elements
• name,greet?	to specify a name followed by an optional greet

Specifying the structure (cont)

• addr*	to specify 0 or more address lines
• tel | fax	a tel *or* a fax element
• (tel | fax)*	0 or more repeats of tel or fax
• email*	0 or more email elements

Specifying the structure (cont)

So the whole structure of a person entry is specified by

name, greet?, addr*, (tel | fax)*, email*

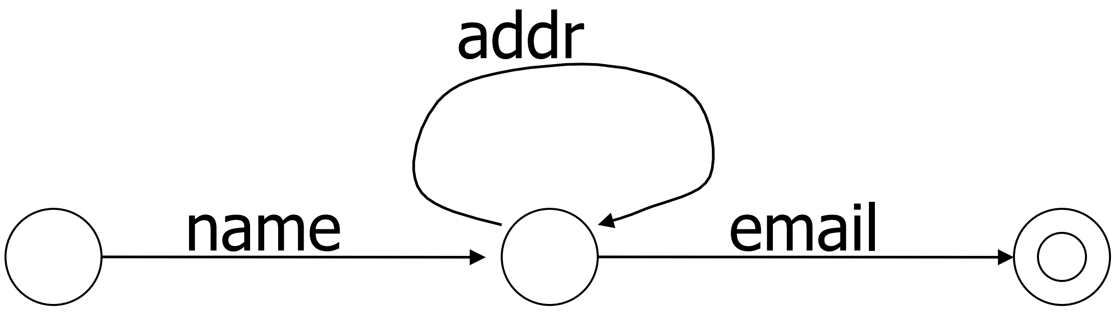
This is known as a *regular expression.* Why is it important?

Regular Expressions

Each regular expression determines a corresponding *finite state automaton*. Let's start with a simpler example:

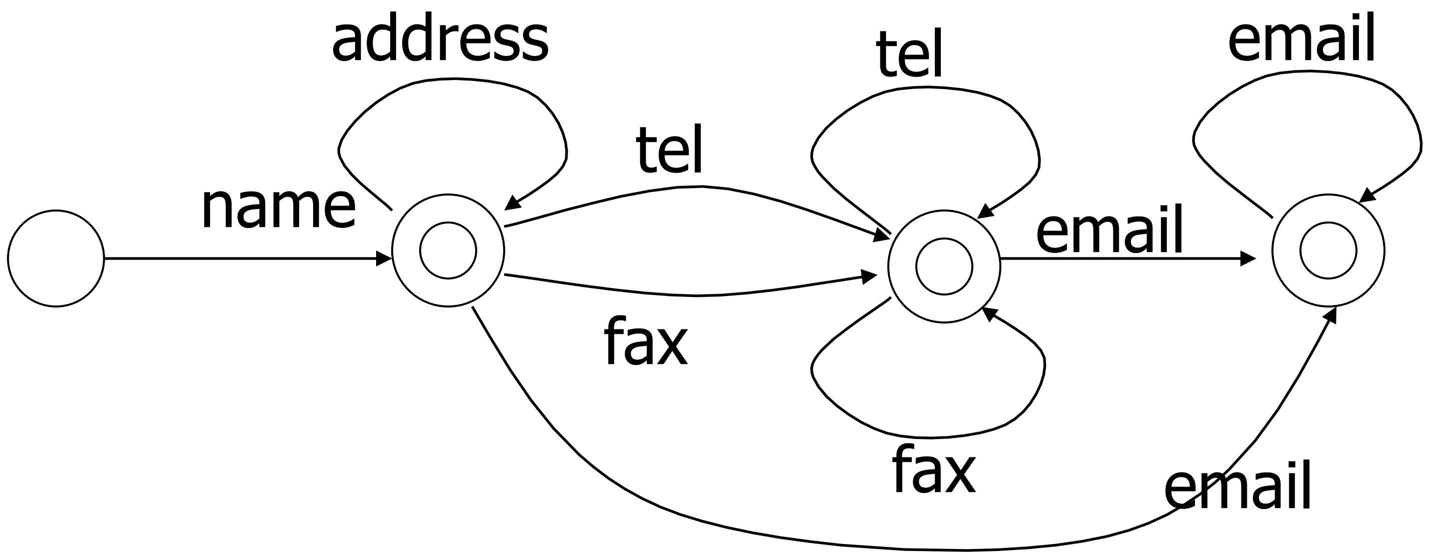
name, addr*, email

This suggests a simple parsing program



Another example

name,address*,(tel | fax)*,email*



Adding in the optional greet further complicates things

A DTD for the address book

```
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (person*)>
  <!ELEMENT person
       (name, greet?, address*, (fax | tel)*, email*)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT greet   (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT tel     (#PCDATA)>
  <!ELEMENT fax     (#PCDATA)>
  <!ELEMENT email   (#PCDATA)>
]>
```

Two DTDs for the relational DB

```
<!DOCTYPE db [
  <!ELEMENT db            (projects,employees)>
  <!ELEMENT projects      (project*)>
  <!ELEMENT employees (employee*)>
  <!ELEMENT project       (title, budget, managedBy)>
  <!ELEMENT employee      (name, ssn, age)>
  ...
]>
```

```
<!DOCTYPE db [
  <!ELEMENT db            (project | employee)*>
  <!ELEMENT project       (title, budget, managedBy)>
  <!ELEMENT employee (name, ssn, age)>
  ...
]>
```

Some things are hard to specify

Each employee element is to contain name, age and ssn elements in some order.

```
<!ELEMENT employee
    ( (name, age, ssn) | (age, ssn, name) |
      (ssn, name, age) | ...
    )>
```

Suppose there were many more fields !

Summary of XML regular expressions

A	The tag A occurs
e1,e2	The expression e1 followed by e2
e*	0 or more occurrences of e
e?	Optional -- 0 or 1 occurrences
e+	1 or more occurrences
e1 | e2	either e1 or e2
(e)	grouping

It's easy to get confused…

```
<!ELEMENT PARTNER (NAME?, ONETIME?, PARTNRID?,
    PARTNRTYPE?, SYNCIND?, ACTIVE?, CURRENCY?,
    DESCRIPTN?, DUNSNUMBER?, GLENTITYS?, NAME*,
    PARENTID?, PARTNRIDX?, PARTNRRATG?,
    PARTNRROLE?, PAYMETHOD?, TAXEXEMPT?, TAXID?,
    TERMID?, USERAREA?, ADDRESS*, CONTACT*)>
```

Cited from oagis_segments.dtd (one of the files in the Novell Developer Kit http://developer.novell.com/ndk/indexexe.htm)

```
<PARTNER> <NAME> Ben Franklin </NAME> </PARTNER>
```

Q. Which NAME is it?

Specifying attributes in the DTD

```
<!ELEMENT height (#PCDATA)>
<!ATTLIST height
     dimension CDATA #REQUIRED
     accuracy CDATA   #IMPLIED >
```

The dimension attribute is required; the accuracy attribute is optional.

CDATA is the “type” of the attribute -- it means string.

Specifying ID and IDREF attributes

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person  (name)>
  <!ELEMENT name    (#PCDATA)>
  <!ATTLIST person
                  id         ID        #REQUIRED
                  mother    IDREF    #IMPLIED
                  father     IDREF    #IMPLIED
                  children  IDREFS  #IMPLIED>
]>
```

Some conforming data

```
<family>
    <person  id="jane"  mother="mary" father="john">
        <name> Jane Doe </name>
    </person>
    <person id="john" children="jane jack">
        <name> John Doe </name>
    </person>
    <person id="mary" children="jane  jack">
        <name> Mary Doe </name>
    </person>
        <person  id="jack"  mother=”mary" father="john">
        <name> Jack Doe </name>
    </person>
</family>
```

Consistency of ID and IDREF attribute values

- If an attribute is declared as ID
  - the associated values must all be distinct (no confusion)
- If an attribute is declared as IDREF
  - the associated value must exist as the value of some ID attribute (no dangling “pointers”)
- Similarly for all the values of an IDREFS attribute
- ID *and* IDREF *attributes are not typed*

An alternative specification

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person  (mother?, father?, children, name)>
  <!ATTLIST person id ID #REQUIRED>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT mother EMPTY>
  <!ATTLIST mother idref IDREF #REQUIRED>
  <!ELEMENT father EMPTY>
  <!ATTLIST father idref IDREF #REQUIRED>
  <!ELEMENT children EMPTY>
  <!ATTLIST children idrefs IDREFS #REQUIRED>
]>
```

The revised data

```
<family>
    <person  id = "jane”>
        <name> Jane Doe </name>
        <mother idref = "mary”></mother>
        <father idref = "john"></father>
    </person>
    <person id = "john”>
        <name> John Doe </name>
        <children idrefs = "jane jack"> </children>
    </person>
    ...
</family>
```

A useful abbreviation

When an element has empty content we can use

```
<tag blahblahbla/>     for    <tag blahblahbla></tag>
```

For example:

```
<family>
    <person  id = "jane”>
        <name> Jane Doe </name>
        <mother idref = "mary”/>
    <father idref = "john”/>
    </person>
    ...
</family>
```

An example

```
<db>
  <movie id="m1">
    <title>Waking Ned Divine</title>
    <director>Kirk Jones III</director>
    <cast idrefs="a1 a3"></cast>
    <budget>100,000</budget>
  </movie>
  <movie id="m2">
    <title>Dragonheart</title>
    <director>Rob Cohen</director>
    <cast idrefs="a2 a9 a21"></cast>
    <budget>110,000</budget>
  </movie>
  <movie id="m3">
    <title>Moondance</title>
    <director>Dagmar Hirtz</director>
    <cast idrefs="a1 a8"></cast>
    <budget>90,000</budget>
  </movie>
  :
  <actor id="a1">
    <name>David Kelly</name>
    <acted_In idrefs="m1 m3 m78" >
    </acted_In>
  </actor>
  <actor id="a2">
    <name>Sean Connery</name>
    <acted_In idrefs="m2 m9 m11">
    </acted_In>
    <age>68</age>
  </actor>
  <actor id="a3">
     <name>Ian Bannen</name>
     <acted_In idrefs="m1 m35">
     </acted_In>
  </actor>
  :
</db>
```

Schema.dtd

```
<!DOCTYPE db [
  <!ELEMENT   db       (movie+, actor+)>
  <!ELEMENT   movie    (title,director,casts,budget)>
  <!ATTLIST   movie    id   ID   #REQUIRED>
  <!ELEMENT   title    (#PCDATA)>
  <!ELEMENT   director (#PCDATA)>
  <!ELEMENT   casts    EMPTY>
    <!ATTLIST casts    idrefs  IDREFS  #REQUIRED>
  <!ELEMENT   budget   (#PCDATA)>
```

Schema.dtd (cont’d)

```
  <!ELEMENT   actor      (name, acted_In,age?, directed*)>
  <!ATTLIST   actor      id      ID       #REQUIRED>
  <!ELEMENT   name       (#PCDATA)>
  <!ELEMENT   acted_In   EMPTY>
    <!ATTLIST acted_In   idrefs  IDREFS  #REQUIRED>
  <!ELEMENT   age        (#PCDATA)>
  <!ELEMENT   directed   (#PCDATA)>
]>
```

Constraints on IDs and IDREFs

- ID stands for identifier. No two ID attributes with the same name may have the same value (of type CDATA)
- IDREF stands for identifier reference. Every value associated with an IDREF attribute must exist as an ID attribute value
- IDREFS specifies several (0 or more) identifiers

Connecting the document with its DTD

In line:

```
<?xml version="1.0"?>
<!DOCTYPE db [<!ELEMENT ...> ... ]>
<db> ... </db>
```

Another file:

```
<!DOCTYPE db SYSTEM "schema.dtd">
```

A URL:

```
<!DOCTYPE db SYSTEM
          "http://www.schemaauthority.com/schema.dtd">
```

Well-formed and Valid Documents

- *Well-formed* applies to any document (with or without a DTD): proper nesting of tags and unique attributes
- *Valid* specifies that the document conforms to the DTD: conforms to regular expression grammar, types of attributes correct, and constraints on references satisfied

Summary on XML and DTD

- XML is a new data format. Its main virtues are widespread acceptance and the (important) ability to handle semistructured data (data without schema).
- DTDs provide some useful syntactic constraints on documents. As schemas they are weak.

Shortcomings of DTDs

- Non-XML syntax
- Only one DTD referenced per document
- No support for namespace
- Useful for documents, but not so good for data:
  - No support for structural re-use such as inheritance
    - Object-oriented-like structures aren't supported
  - No support for data types
    - Can't do data validation
  - Can have a *single* key item (ID), but:
    - No support for multi-attribute keys
    - No support for foreign keys (references to other keys)
    - No constraints on IDREFs (reference *only* a Section)