DTD limitations
Based on names

- DTDs attach a fixed meaning on the name of an element, regardless of context
- (attribute names are element-scoped)
- Either you give different names to the same type of element in different contexts, or check on application level the actual allowed structures
Example problem with names

Idea is that footnotes may contain paragraphs, *except* that they shouldn’t contain other footnotes.

- Cannot be directly expressed with an XML DTD.
- Could use another element (e.g., `fnpara`), but would have to duplicate definitions, maintenance and learning.
Not all structures possible

- DTDs use a limited form of regular expressions for content models (deterministic)
- the traditional limitation of regular expressions: inability to define nested structures is not a problem, since the tree-structure defines nesting
Example problem structure

- Want to write a DTD for a chess game: white goes first (or quits), then any alternating sequence of black and white moves

  ```xml
  <!ELEMENT game ( white, (black, white)*, black? ) >
  ```

- Hopelessly nondeterministic (cannot be fixed)

- Can only be checked on application level
No datatypes for element content

- Elements are often meant to contain certain types of data
- Both in document-oriented (dates, numbers, units etc.)
- and in data-oriented (chars, ints, number, strings, dates, binary) structures
- No way of expressing in a DTD (theoretically notations can be used to express, but not to validate)
DTD alternatives

- Alternatives basically from two directions: data-oriented companies (databases, middleware) like Microsoft, Oracle, IBM

- and the language-theoretical crowd (MURATA Makoto, James Clark)

- First resulted in W3C XML Schema, other in Relax NG (and related efforts)
XML Schema

- Detailed and straightforward datatype definitions, IMO very usable
- Very complicated structure definitions including inheritance
- Fairly good support in database interfaces, some document-oriented support
- Inheritance changes the basically string/name-oriented nature of XML: documents with and without associated schema represent very different sets of information
- Very few people understand Schema structures well
<xs:element name="recipe">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="head"/>
            <xs:element minOccurs="0" ref="notes"/>
            <xs:element ref="ingredients"/>
            <xs:element ref="directions"/>
        </xs:sequence>
        <xs:attribute name="categories" type="xs:NMTOKENS"/>
        <xs:attribute name="revision" use="required"/>
        <xs:attribute name="id" type="xs:ID"/>
        <xs:attribute ref="xml:lang"/>
    </xs:complexType>
</xs:element>
XML Schema

- XML-based syntax
- more content model possibilities, but requires determinancy
Relax NG

- Unifies attribute and element models
- No built-in text-content datatyping, but allows importing of such modules, eg. XML Schema
- Grounded in theory of hedge-languages, quite orthogonal features
- Allows eg. exclusions
- Simple
Relax NG example

menu = element menu { menu.attlist, head, description*, recipe+ }
menu.attlist &=
    attribute revision { text },
    attribute id { xsd:ID }?
recipe =
    element recipe {
        recipe.attlist, head, notes?, ingredients, directions
    }
Non-deterministic Relax example

white = element white { empty }
black = element black { empty }
game = element game { white, (black, white)*, black? }
start = game
Relax NG

- Both XML-based and compact syntax (shown)
- Developing into an ISO standard
- [http://www.relaxng.org/](http://www.relaxng.org/)
Schema or Relax NG?

- Both usable
- Relax NG significantly easier to learn and use
- Schema has good tool support for database oriented tasks and some authoring support
- Relax NG has some authoring tool support
- Tools exist to convert between the two (and DTDs)
- Maybe use Relax NG for document-oriented tasks, Schema for data-oriented
DTD miscellania
Linking external data

- You could use notations and entities
- Quite arcane, require DTD modifications for use
- In my opinion, best bet is to use HTML-like empty elements
- e.g. for images, use something like `<img href="URL" type="image-type" />`
Analyzing database schema

- Maybe the document (or a part of it) can be automatically generated from a database
- Then we can use possible existing
  - Database schema diagrams/documentation
  - UML or ER diagrams
- Giving us
  - One-to-one relations from table columns (chapter (title, ...))
  - One-to-many relations (child tables) (chapter (para+), chapter(para*))
- But not the order of elements
Standard DTDs

- "Standardized" (by whom, for who?), agreed upon; de facto or de jure?

- Benefits
  - Document exchange
  - Save DTD design work
  - Ready-made stylesheets and applications
  - May be familiar to users already
Standard DTDs

- Disadvantages
  - Two organizations may not have exactly the same needs in practice → own specializations
  - May be two simple / flexible
  - or too complex
Standard DTD or custom?

- Pragmatic: choose existing
- Make necessary changes
  - remove unnecessary elements (to simplify)
  - constrain content models (for stricter rules and easier authoring decisions)
  - add elements (to cover your specific needs)
- Still compatible?
  - Probably not with documents from others
  - If only simplifications / added constraints your documents may be processable by others
Standard DTD or custom?

- Note that many standard DTDs may already come in different flavors, or have customization features built-in
Standard DTDs, examples

- http://www.schema.net/
- http://www.xml.org/xmlorg_registry/
- http://www.xml.com/pub/rg/DTDs
- http://xml.coverpages.org/
- MathML, CML (chemistry), UXF (UML eXchange Format), SMIL (multimedia; also HyTime), RDF (Resource Description Framework), HumanML (natural languages), DocBook