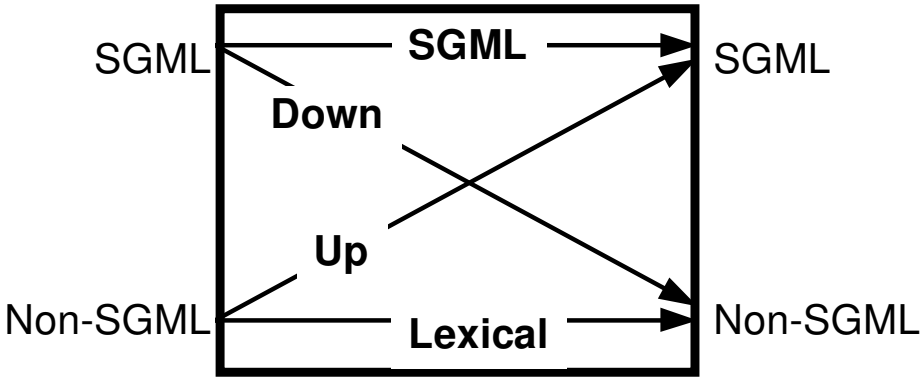
Types of Conversions/Transformations



My personal nomenclature:

Conversion: Up and Lexical

source is not structured, content-identified data

100% accuracy not easily obtained

Transformation: SGML and Down

source is structured, content-identified, validated data

100% accuracy possible

This presentation deals solely with SGML transformations.

Why Bad DTDs Happen to Good People

The dreaded "chasm":

Optimal in-house DTD:

Content-driven design

Formatting-free

Omission of auto-generate-able data

Processor Needs:

Hardcopy generators / On-line publishing

Young industry: processors have limitations

Distribution DTD:

Industry-standard DTD

DTD required by major customer

Typical result: In-house DTD is compromised to bridge the gap

Example Processor Limitations

Example processor: DynaText on-line presentation system

Can't do:

auto-numbering of heterogeneous element types

E.g. numbering PARA0 and SUBPARA1 elements as a single sequence

Inefficient at:

calculating list-adornment type based on complex context-sensitive algorithms

Stylesheet language is powerful enough, but performance issues arise on low-end platforms.

Shouldn't be used to do:

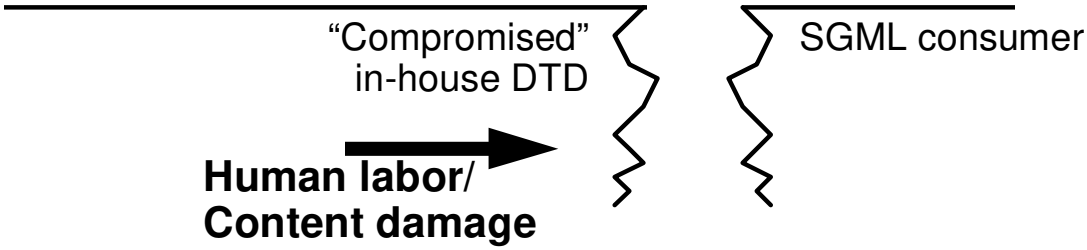
auto-generate text

Stylesheets can generate a section title at point of XREFerence.

But... generated text is unknown to the search engine.

Wrong Solution: Compromise your DTD

Extend your side of the cliff to bridge the chasm:



Costs:

Disruption of current authoring/conversion processes

DTD design is "compromised"

DTD's portability/lifespan is shortened

Future processor choices/enhancements

Future distribution DTDs

Authoring process becomes more difficult

Typically, the compromise places greater burden on author.

Typically, additional "mindless" tasks are added to authors' role.

Solution distasteful for philosophical reasons:

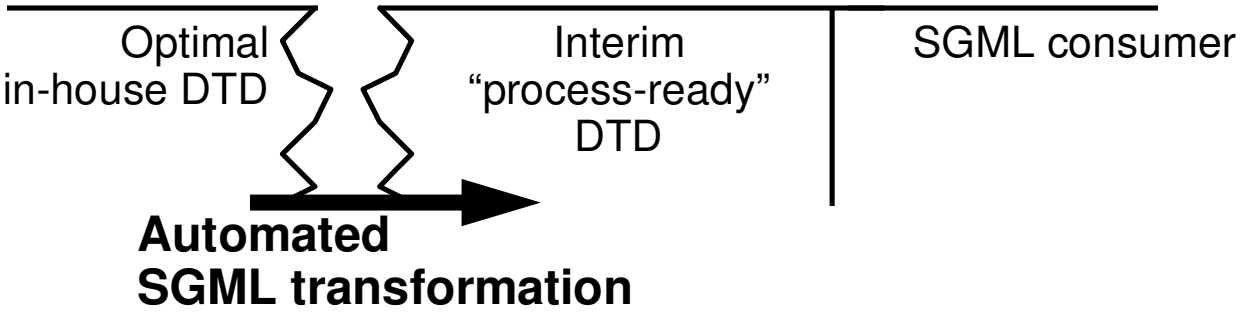
Don't change a long-term strategy to make up for temporary deficiencies in SGML processors.

Don't design a DTD with today's processors in mind.

Consensus: Bring the SGML consumer to the data; don't force the data to travel to the consumer!

Correct Solution: Build an "Xform Bridge"

Extend the "reach" of the SGML consumer:



Author/review environment is not affected

Transformation can be 100% automated and 100% accurate

Interim instance need not conform to an actual DTD

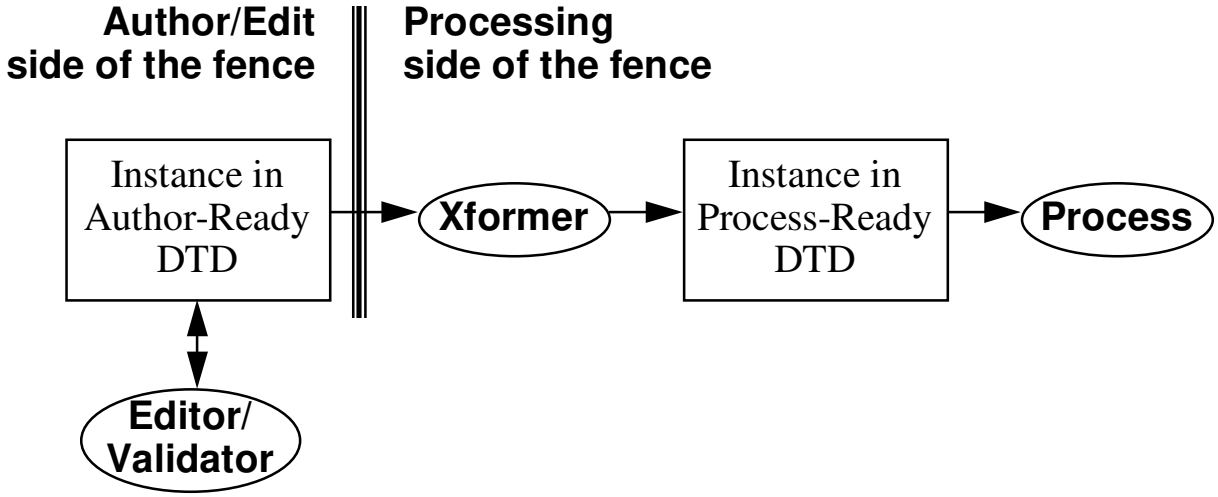
Some processors (e.g. DynaText) don't require a DTD.

Validation of interim instance is optional if the transformation process itself is "validated".

Side effect: lots of other good things

Transformations can achieve a lot more... Details later...

Author-Ready DTD vs. Process-Ready DTD



Author/Editor Ready

Contains no information that can/should be auto-generated

Let the author focus on content!

The "noble but usually not practical" form of SGML

Not riddled with "compromise" attributes needed to satisfy processor

```
<RANDLIST><ITEM>...<RANDLIST><ITEM>...
```

Process-Ready

Contains auto-generated information needed by the processor

The "real-world" form of SGML

```
<RANDLIST ADORN=BULLET>
    <ITEM ADORNTEXT=BULLET>...
    <RANDLIST ADORN=DASH>
        <ITEM ADORNTEXT=DASH>...
```

Sundry Other Uses for Transformation Engines

Import data from external sources

Generate tables from databases

Translate object IDs into "addresses" needed by processor

CALS DTD: <EXTREF DOCID=XJ8932>

DynaText: <EXTREF BOOKNAME=A7899 COLL=CD0022>

"Intelli-Minimization"

Automate "mindless" or "formatting-based" authoring tasks

Autocalculate list-adornment type

Transform content-based tables to content-generic table markup (e.g. CALS) required for hard/softcopy

"Intelli-Validation"

A DTD stores data supporting only "structure" validation.

The only content validation performed by SGML parsers concerns matching IDs and IDREFs.

An xform engine can help verify data semantics.

Perform document analysis

Produce a report (e.g. statistics, frequency counts, etc.)

Extract/duplicate data

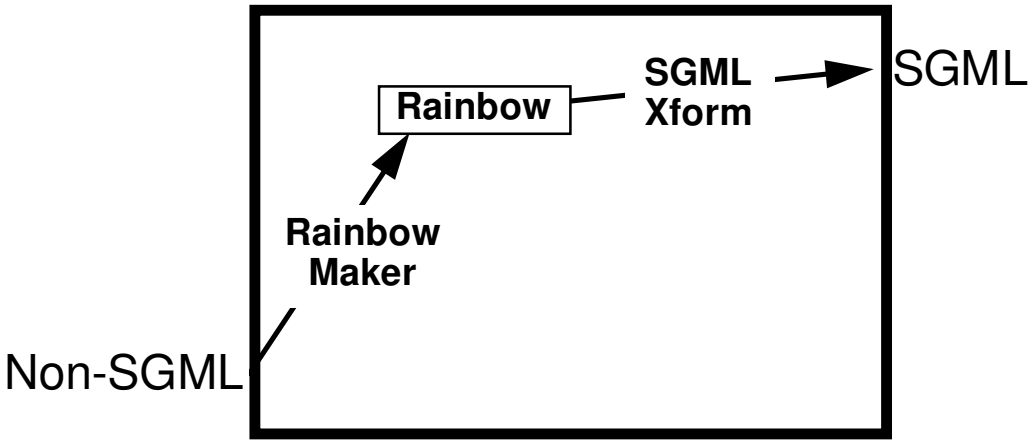
Generate a back-of-the-book index

Generate a table-of-contents/figures

Hide data

Selectively hide information based on intended audience

Up conversion via SGML transformation



Rainbow: a DTD representing word-processor documents

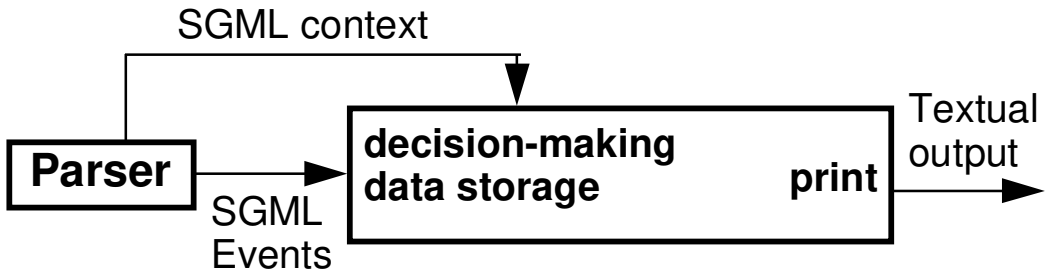
Developed by EBT

"Alpha" Rainbow Makers already exist for RTF, Frame, and Interleaf.

Transforms the notorious "Up-conversion" problem into a more-manageable SGML-transformation problem

For more information, see our poster session.

The "Parser with a Print Statement" Utility



Off-the-Shelf tools used for SGML and Down transformations:

- Avalanche's Hammer
- Exoterica's OmniMark
- AIS/MID's Balise

Limitations:

No "lookahead"

Limited "lookbehind": typically restricted to ancestors and left siblings

Not optimal (but currently the best choice) for SGML transformations

Input side has SGML intelligence

Output side has none

Thus: No verification/validation of output instance

Note: OmniMark has output-side validation, but only for non-SGML input

The "Tree Analyzer with a Print Statement"

- **EBT System Integrators Toolkit (SIT)**

  Requires C programming

  The only transformation engine that has an integrated fulltext-search capability.

  Access to entire input document

  Tree is file-based.

- **"Polypus" extension to AIS/MID Balise**

  Uses Balise language

  Access to tree fragments

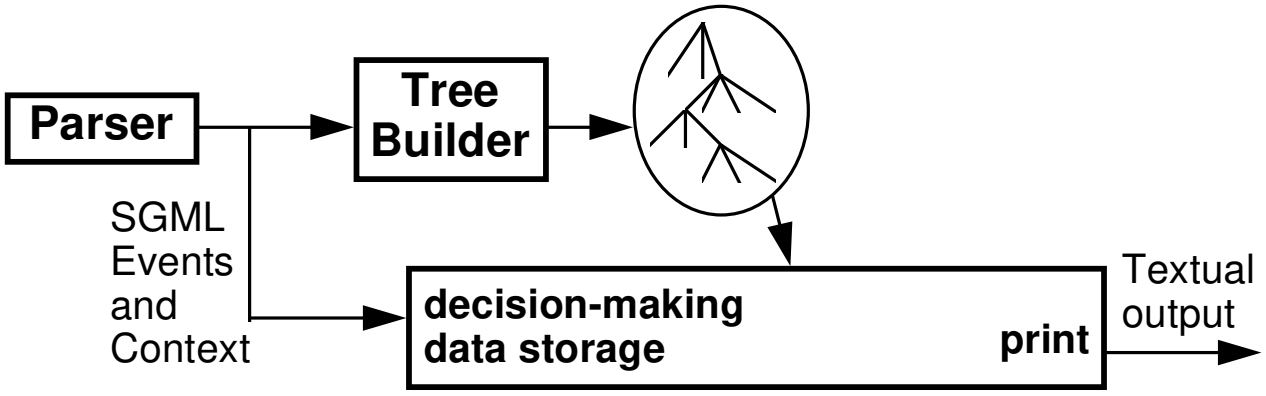
  Tree is RAM-based.

- **Unlimited tree access**

  Arbitrary look-ahead/behind

  Arbitrary re-ordering of input objects

- **"SGML-ignorant output" limitations still present**



Comparison-shopping tips

Software externals

- Range of platforms is supported
- Speed
- RAM requirements

Software internals

- Scripting language
  - *Comfortable, easy to learn?*
  - *Good diagnostics?*
  - *Extensible?*
- Script-debugging aids
- Error recovery during parsing of input document
- Amount of access to the input tree
- Pattern-matching
  - *Can patterns match arbitrary text-chunks? Or are only SGML events matchable?*
  - *Can patterns specify data types (CDATA, SDATA, etc.)?*
  - *Can patterns distinguish EMPTY elements?*