FI MU

Faculty of Informatics
Masaryk University

Logical Markup from RTF

by

Michal Chocholáč

FI MU Report Series FIMU-RS-2000-02

February 2000

Logical markup from RTF

Michal Chocholáč

Abstract

The paper is an overview of ways how to get logical structure from documents created by proprietary word-processors and stored into an RTF file. It describes and evaluates all steps of conversion process from RTF to SGML/XML. One of the submitted ways is demonstrated on a simple example. All used tools exist and are publicly available.

Contents

1 Introduction

The Rich Text Format (RTF) is a format for text and graphics interchange that can be used with different output devices, operating environments, and operating systems. RTF uses the ASCII (or other) character set to control the representation and formatting of a document. Thus, documents created under different operating systems (MS-DOS, Windows, UNIX, OS/2, Macintosh, Power Macintosh) and with different software (T602, Microsoft Word 6.0, …) applications can be transferred between those operating systems and applications.

RTF Version 1.5 has been updated to include all new control words introduced by Microsoft Word for Windows 95 version 7.0 and Word 97 for Windows. For more information about RTF see [1].

2 Motivation

All documents stored on computer systems have instructions or codes embedded in the text that indicate how the text should be processed. These instructions are called markup. The basic types of the markup are as follows:

- **Procedural markup** supplies detailed instructions for actions that software must follow in processing the data. It says, "Do x".
- **Declarative markup** (or descriptive or logical markup) supplies only high-level logical descriptions of the data's role or purpose, expecting that separate processing software will map the markup to the precise actions to be performed as well as actually perform the actions. It says, "I am a y".

Procedural markup is efficient because it allows a computer to follow the supplied instructions without doing additional interpretive work. However, it binds the data closely to a single kind of manipulation.

Declarative markup requires an additional interpretation step, but allows document data that is stored in a single form to be formatted, analyzed, and manipulated many different ways, increasing the value of the data once it has been described thoroughly and abstractly.

RTF is procedural because it represents only a visual form of the document. By contrast, The Standard Generalized Markup Language (SGML) can be highly declarative. If an SGML markup language is well designed and properly used, it is independent of procedures and processing and does not allow the data's value for multiple purpose to be compromised.

So, let we try to find ways how to convert an RTF document to an SGML (or XML) document with logical markup using existing and free tools.

3 Validation tool

Software, which reads SGML document and recognizes the markup in them so that other software components can process the markup and data is called parser. Validating SGML parser is a special kind of parser that reads DTDs[1] and document instances and finds any markup errors in them. It must be able to find and report a reportable markup error if (and only if) one exists.

Several public-domain validating parsers are available.

3.1 Nsgmls

Nsgmls is a component of an SGML Parser (SP) [8]. It is a free, object-oriented toolkit for SGML parsing and entity management created by James Clark. Included parts are as follows:

- **Nsgmls** – a parser and validator of SGML documents
- **Spam** (SP Add Markup) – an SGML markup stream editor
- **Sgmlnorm** – a normalizer of SGML documents
- **Sx** – a converter from SGML to XML
- **Spent** (SGML Print Entity) – concatenator of SGML entities

The source codes are available. Thus, it is possible to compile the SP for any operating system. However, you can get binaries for DOS, Windows 95/NT, OS/2 and all major Unix variants.

Usage: SP (version 1.3.3) is installed on all faculty UNIX platforms (Irix, Solaris and Linux) in the module `sp`.

```
$ module add sp
```

[1] Document Type Definition. See [3].

```
$ nsgmls sysid...
```

Nsgmls parses and validates an SGML document whose document entity is specified by the system identifiers[2] `sysid...` and prints on the standard output a simple text representation of its ESIS[3]. If more than one system identifier is specified, then the corresponding entities will be concatenated to form the document entity. A command line system identifier of `-` can be used to refer to the standard input. For using nsgmls's options see [8].

The output is a series of lines. Each line consists of an initial command character and one or more arguments. There is no space between the command character and the first argument. Complete set of command characters and its arguments are listed in [8]. If the last command on the output is `C`, it indicates that the document was a conforming SGML document.

4 Conversion

A mechanism of conversion is composed of two phases:

The first step is to read the source document, decode whatever pattern of markup there is and translate it into an intermediate conversion DTD. The conversion DTD might also include presentation-related elements and elements that hold conversion data, which don't appear in the target DTD. It allows clarification of the original markup (without loosing any of the source information) by translating it into a simple SGML form, which is much easier to process than source markup in further conversion.

The second step is to interpret the results of the first step and generate an instance conforming to the target DTD, i.e. to transform the intermediate document from the intermediate conversion DTD to a target DTD.

4.1 From RTF to SGML

Here we present the tools that allow document conversion from RTF to an intermediate DTD. All tools listed below are installed on faculty computers and many others are available on the Internet.

[2]The following system identifiers are available: filename, file descriptor and URL (only the http scheme is currently supported).

[3]Element Structure Information Set, ISO 13673.

4.1.1 Rainbow Maker

Rainbow Maker is a software program designed to convert documents in a proprietary word-processor (WP) format to Rainbow documents.

The Rainbow format [4] is actually an SGML DTD. It represents a unification of the wide spectrum of proprietary formats. It was developed by Electronic Book Technologies (EBT), in conjunction with several other key SGML vendors and promoters. The Rainbow DTD is publicly available, and can be used and modified by organizations and individuals freely.

EBT has developed public-domain Rainbow Makers for some key WP formats (RTF, MIF, ...). Unfortunately, EBT will distribute only executable versions (PC and several UNIX platforms) of these Makers, but when the versions become more stable, EBT will make the source code available.

Usage: Rainbow Maker is installed on Solaris platform in the module `rbmaker`. The program requires two directories called `fcm` and `pem`. The Sun version looks for them in the "current directory" at launch time. If it cannot find these directories, it crashes! Thus, you must specify path to these directories in the command line:

```
$ module add rbmaker
$ rbmaker -in in.rtf -out out.rbw\
-figdir /packages/share/rbmaker-2.5/data\
-datadir /packages/share/rbmaker-2.5/data\
-tempdir /tmp
```

Shell script `gorbmaker` can simplify the notation of this command. Thus, you can use it as follows:

```
$ gorbmaker in.rtf out.rbw
```

This Rainbow Maker release supports RTF documents from MacWord 5.x, WinWord 2.x/6.0. AmiPro RTF from any other source is subject to data/formatting loss, or even premature program termination.

4.1.2 rtf2xml

Filter rtf2xml is an OmniMark's[4] script which translates RTF to a paragraph structured XML document. It was created by Rick Geimer.

[4]OmniMark is a streaming programming language. See [6].

This program is free distributed and portable across all operating systems where OmniMark available.

Usage: The OmniMark is installed on Solaris and Linux platforms in the module `omnimark`.

```
$ module add omnimark
$ cp -r /packages/share/omnimark-5.1/rtf2xml/ .
$ cd rtf2xml/
$ omnimark -s rtf2xml.xom -of ../out.xml ../in.rtf
```

You can pass multiple RTF files as input, and they will be concatenated into a single XML file.

4.2 SGML to SGML transformations

The transformation process is based on the parsing of the SGML instance and the use of application languages. The parser reads the instance and transforms it into a sequence of events consisting minimally of the ESIS. This sequence of events is then read by an application language that is programmed to recognize each markup and to trigger the appropriate action.

The difficulty of transformation lies in the convergence or divergence of the markup models.

4.2.1 Jade

James' DSSSL Engine (Jade) is an implementation of the DSSSL[5] style language. It was created by James Clark. The program is distributed for UNIX and MS/Windows (32-bit) operating systems, the source code is available.

With Jade SGML documents can be formatted to the users needs using DSSSL-stylesheets. The output documents can be in HTML, in RTF and in $\TeX$. Jade can also perform SGML to SGML and SGML to XML transformations.

Usage: The Jade is installed on all faculty UNIX platforms (Irix, Solaris and Linux) in the module `jade`.

```
$ module add jade
```

[5] Document Style Semantics and Specification Language, ISO/IEC 10179:1996.

```
$ jade -t sgml in.sgml
```

By default, the DSSSL specification is expected in the file with the same name as the input file and with `dsl` extension instead of the original (i.e. `in.dsl`). If you want use some other name of the DSSSL specification file, use `-d` option. For example:

```
$ jade -t sgml -d spec.dsl in.sgml
```

Only a limited set of DSSSL flow objects are implemented in the Jade[6], so you would not always be able to get everything exactly as you want it.

4.2.2 sgmlspl

A simple post-processor for Nsgmls sgmlspl is a sample application distributed with the SGMLS.pm perl5 class library. It can be used to convert SGML documents to other formats by providing a specification file detailing exactly how you want to handle each element, external data entity, subdocument entity, CDATA string, record end, SDATA string, and processing instruction.

Sgmlspl using SGMLS.pm repackages the ESIS output of Nsgmls into perl5 objects. It exports a single subroutine `sgml(event, handler)` from a specification file into the main package and execute `handler` every time when the `event` occurs. You may use `sgml` to declare a handler for a generic event, like `'start_element'`,or a specific event, like `'<TAG>'`. A specific event will always take precedence over a generic event.

Usage: Filter sgmlspl is installed on all faculty UNIX platforms (Irix, Solaris and Linux) in the module `sgmlspl`.

```
$ module add sp perl5 sgmlspl
$ nsgmls in.sgml | sgmlspl specification.pl >out.sgml
```

5 Example of usage

This example presents a conversion way from RTF to Rainbow DTD to target DTD using Rainbow Maker and sgmlspl (Figure 1). I chose the Rain-

[6]Complete set of of DSSSL flow objects supported by Jade are listed in [7].

bow DTD because it has better support as the Rick Geimer's DTD. Since my knowledges of the DSSSL are only on the basic level, so I decided to use the sgmlspl tool for SGML transformations.

The example is only demonstration of the usage all necessary tools. So, I will try to do it simply and synoptical. Real use will require more complicated solutions (especially in error correction domain).

Two-lined content of the source document is used because more lines imply only longer contents afterwards generated files and it reduce the lucidity of this paper.

5.1 Source document

The source document was written by Microsoft Word 97 and exported into an RTF file named `addressbook.rtf`. It represents an easy address book. A line contains:

```
id-number, name, address, phone-number
```

Standard options are: Times New Roman font, 10 pt. The `id-number` is written by Arial font, `name` is bold, `address` is italic and `phone-number` uses 8 pt size of font.

Content:

01234, **Antonov Adam**, *Aladinova 12, 432 10 Adamov*, 01/123456
12340, **Bojarova Blanka**, *Bayerova 23, 333 22 Brno*, 02/234561

5.2 Intermediate document

```
artemis$ module add rbmaker
artemis$ gorbmaker addressbook.rtf rb.sgml
```

Shell script `gorbmaker` was created for easier usage of `rbmaker`. File specified by the first parameter will be converted to Rainbow DTD conforming document and stored into file specified by the second parameter.

Content:

```
<!DOCTYPE rainbow PUBLIC "-//EBT//DTD Rainbow 2.5//EN"
[]>
```

```
<RAINBOW>
  <FILEINFO ORIGIN="WinWord-RTF1" DTDVER="2.5">
  <STYINFO>
    <PARATYPE FONT-FAMILY="Times New Roman"
              FONT-SIZE="10"
              LINE-SPACING="12"
              CHARSET="ISO-8859"
              JUSTIFICATION="LeftJust"
              FIRST-INDENT="0"
              LEFT-INDENT="0"
              RIGHT-INDENT="0"
              SPACE-BEFORE="0"
              SPACE-AFTER="0"
              FONT-WEIGHT="Medium"
              FONT-SLANT="Roman"
              NAME="Normal">
    <CLFTYPE NAME="Default Paragraph Font">
  </STYINFO>
  <DOC>
    <WPLOC wp-addr="1">
    <PARA FONT-SIZE="10"
          LINE-SPACING="12"
          PARATYPE="Normal">
      <PARACONT>
        <CLF FONT-FAMILY="Arial">01234</CLF>,
        <CLF FONT-WEIGHT="Bold">Antonov Adam</CLF>,
        <CLF FONT-SLANT="Ital">Aladinova 12, 432 10 Adamov</CLF>,
        <CLF FONT-SIZE="8">01/123456</CLF>
      </PARACONT>
    </PARA>
    <WPLOC wp-addr="2">
    <PARA FONT-SIZE="10"
          LINE-SPACING="12"
          PARATYPE="Normal">
      <PARACONT>
        <CLF FONT-FAMILY="Arial">12340</CLF>,
        <CLF FONT-WEIGHT="Bold">Bojarova Blanka</CLF>,
        <CLF FONT-SLANT="Ital">Bayerova 23, 333 22 Brno</CLF>,
        <CLF FONT-SIZE="8">02/234561</CLF>
      </PARACONT>
    </PARA>
  </DOC>
</RAINBOW>
```

5.3 Target document

5.3.1 Target DTD

The target DTD is stored in the external file `target.dtd`.

Content:

```
<!ELEMENT addressbook - - (person+)>
<!ELEMENT person - - (name, surname, address, phone*)>
<!ATTLIST person
     id     NUMBER           #IMPLIED >
<!ELEMENT name - - (#PCDATA)>
<!ELEMENT surname - - (#PCDATA)>
<!ELEMENT address - - (#PCDATA)>
<!ELEMENT phone - - (forenum?, num)>
<!ELEMENT forenum - - (#PCDATA)>
<!ELEMENT num - - (#PCDATA)>
```

5.3.2 The specification file

The specification file `spec.pl` contains instructions for processing the intermediate SGML document stored in `rb.sgml`.

For easier creating specification files is available a program `skel.pl`. It is an sgmlspl specification which writes a specification.

```
aisa$ module add perl5 sp sgmlspl
aisa$ cp /packages/share/sgmlspl/doc/skel.pl .
aisa$ nsgmls rb.sgml | sgmlspl skel.pl > spec.pl
```

This command will generate a skeleton of potential events appropriate for the document `rb.sgml`. It can be used as a starting point for writing real `spec.pl`.

Content:

```
use SGMLS;
use SGMLS::Output;

#
# Initializing
#
```

```
$doc = 0;
$para = 0;
$paracont = 0;
$clf = 0;
$output[0] = "";

#
# Subprograms
#

sub begin_doc {
   print '<!DOCTYPE addressbook SYSTEM "target.dtd">' . "\n";
}

sub begin_change {
   my $hlp;
   $hlp = $_[0]->attribute('FONT-FAMILY')->value;
   unless ($hlp) { $hlp = $FAMILY[$#FAMILY];}
   push @FAMILY, $hlp;
   $hlp = $_[0]->attribute('FONT-WEIGHT')->value;
   unless ($hlp) { $hlp = $WEIGHT[$#WEIGHT];}
   push @WEIGHT, $hlp;
   $hlp = $_[0]->attribute('FONT-SLANT')->value;
   unless ($hlp) { $hlp = $SLANT[$#SLANT];}
   push @SLANT, $hlp;
   $hlp = $_[0]->attribute('FONT-SIZE')->value;
   unless ($hlp) { $hlp = $SIZE[$#SIZE];}
   push @SIZE, $hlp;
}

sub end_change {
   my $hlp;
   $hlp = pop @FAMILY;
   $hlp = pop @WEIGHT;
   $hlp = pop @SLANT;
   $hlp = pop @SIZE;
}

sub begin_rb {
   print "<addressbook>\n";
}

sub end_rb {
   print "</addressbook>\n";
```

```
}

sub begin_paratype {
   $FAMILY[0] = $_[0]->attribute('FONT-FAMILY')->value;
   $WEIGHT[0] = $_[0]->attribute('FONT-WEIGHT')->value;
   $SLANT[0] = $_[0]->attribute('FONT-SLANT')->value;
   $SIZE[0] = $_[0]->attribute('FONT-SIZE')->value;
}

sub begin_para {
   $para = 1;
   &begin_change;
}

sub end_para {
   $para = 0;
   &end_change;
}

sub begin_paracont {
   $paracont = 1;
   push @output,
      qq(<person id="#id#">\n<name>#name#</name>\n) .
      qq(<surname>#surname#</surname>\n) .
      qq(<address>#address#</address>\n) .
      qq(<phone>\n#phone#</phone>\n</person>\n);
}

sub end_paracont {
   $paracont = 0;
   my $hlp = pop @output;
   print "$hlp";
}

sub begin_clf {
   $clf = 1;
   &begin_change;
}

sub end_clf {
   $clf = 0;
   &end_change;
}

sub begin_element {
```

```
   die "Unknown element: " . $_[0]->name;
}

sub cdata {
   if ($doc && $para && $paracont && $clf) {
      if (($FAMILY[$#FAMILY] eq "Times New Roman") &&
          ($WEIGHT[$#WEIGHT] eq "MEDIUM") &&
          ($SLANT[$#SLANT] eq "ITAL") &&
          ($SIZE[$#SIZE] eq "10")) {
         my $hlp = pop @output;
         $hlp =~ s/#address#/$_[0]/g;
         push @output, $hlp;
      }
      elsif (($FAMILY[$#FAMILY] eq "Arial") &&
          ($WEIGHT[$#WEIGHT] eq "MEDIUM") &&
          ($SLANT[$#SLANT] eq "ROMAN") &&
          ($SIZE[$#SIZE] eq "10")) {
         my $hlp = pop @output;
         $hlp =~ s/#id#/$_[0]/g;
         push @output, $hlp;
      }
      elsif (($FAMILY[$#FAMILY] eq "Times New Roman") &&
          ($WEIGHT[$#WEIGHT] eq "MEDIUM") &&
          ($SLANT[$#SLANT] eq "ROMAN") &&
          ($SIZE[$#SIZE] eq "8")) {
         my $hlp = pop @output;
         (my $forenum, my $num) = split /\//, $_[0];
         $hlp =~ s/#phone#/<forenum>$forenum<\/forenum>\n#num#/g;
         $hlp =~ s/#num#/<num>$num<\/num>\n/g;
         push @output, $hlp;
      }
      elsif (($FAMILY[$#FAMILY] eq "Times New Roman") &&
          ($WEIGHT[$#WEIGHT] eq "BOLD") &&
          ($SLANT[$#SLANT] eq "ROMAN") &&
          ($SIZE[$#SIZE] eq "10")) {
         my $hlp = pop @output;
         (my $surname, my $name) = split / /, $_[0];
         $hlp =~ s/#name#/$name/g;
         $hlp =~ s/#surname#/$surname/g;
         push @output, $hlp;
      }
      else {
         my $hlp = $_[0];
         $hlp =~ s/\s+/ /g;
         if ($hlp ne " ") {
```

```
            print "<unknown>$hlp</unknown>";
         }
      }
   }
}

#
# Handlers for specific events
#

# Element: RAINBOW
sgml('<RAINBOW>', \&begin_rb);
sgml('</RAINBOW>', \&end_rb);

# Element: FILEINFO
sgml('<FILEINFO>', "");
sgml('</FILEINFO>', "");

# Element: STYINFO
sgml('<STYINFO>', "");
sgml('</STYINFO>', "");

# Element: PARATYPE
sgml('<PARATYPE>', \&begin_paratype);
sgml('</PARATYPE>', "");

# Element: CLFTYPE
sgml('<CLFTYPE>', "");
sgml('</CLFTYPE>', "");

# Element: DOC
sgml('<DOC>', sub {$doc = 1; });
sgml('</DOC>', sub {$doc = 0; });

# Element: WPLOC
sgml('<WPLOC>', "");
sgml('</WPLOC>', "");

# Element: PARA
sgml('<PARA>', \&begin_para);
sgml('</PARA>', \&end_para);

# Element: PARACONT
sgml('<PARACONT>', \&begin_paracont);
sgml('</PARACONT>', \&end_paracont);
```

```
# Element: CLF
sgml('<CLF>', \&begin_clf);
sgml('</CLF>', \&end_clf);


#
# Handlers for generic events
#

sgml('start', \&begin_doc);
sgml('end', '');

sgml('start_element', \&begin_element);
sgml('end_element', '');

sgml('cdata', \&cdata);

sgml('re', '');

1;
```

5.3.3 Document entity

```
aisa$ module add perl5 sp sgmlspl
aisa$ nsgmls rb.sgml | sgmlspl spec.pl > addressbook.sgml
```

Content:

```
<!DOCTYPE addressbook SYSTEM "target.dtd">
<addressbook>
<person id="01234">
<name>Adam</name>
<surname>Antonov</surname>
<address>Aladinova 12, 432 10 Adamov</address>
<phone>
<forenum>01</forenum>
<num>123456</num>
</phone>
</person>
<person id="12340">
<name>Blanka</name>
```

```
<surname>Bojarova</surname>
<address>Bayerova 23, 333 22 Brno</address>
<phone>
<forenum>02</forenum>
<num>234561</num>
</phone>
</person>
</addressbook>
```

5.4 Validation

```
aisa$ module add sp
```

Intermediate document: `aisa$ nsgmls rb.sgml`

Target document: `aisa$ nsgmls addressbook.sgml`

If the last character on the standard output of `nsgmls` is `C`, it means that the document is a conforming SGML document.

5.4.1 Error correction

This example uses very simply method of error correction because of the need for lucidity of the report. By contrast, in the real use is necessary to create proper aid for finding and/or correction of maximum count of errors as possible. The subcode in the sgmlspl specification file:

```
else {
   my $hlp = $_[0];
   $hlp =~ s/\s+/ /g;
   if ($hlp ne " ") {
      print "<unknown>$hlp</unknown>";
   }
}
```

inserts any unidentified incoming non-whitespace `#PCDATA` into an element `unknown`. A parser will notify of unknown element `unknown` during validation and it will indicate an error in source document. This solution provides only elementary error prevention of input data.

General prevention should be allowed by two-level hierarchy. The first level is to embed any unidentified incoming data into an unknown element and detect this errors during validation process. The second level is to create auxiliary structures for context logging which allows to report any identified incoming data in bad context. This errors can be detected and reported during conversion process, i.e. immediately when they occur.

6 Comparison

6.1 Rainbow Maker vs. rtf2xml

The Rainbow DTD represents unification of the wide spectrum of proprietary formats. It is better supported by the authors and other SGML developers. Filter rtf2xml has insufficient documentation.

Filter rtf2xml is a script. It must be executed under OmniMark C/VM. So, its run time is several time longer[7] than run time of binary Rainbow Maker. In addition, rtf2xml generates much longer[8] output. The Rainbow Maker is easier for use and has more intuitive interface (especially `gorbmaker`).

6.2 Jade vs. sgmlspl

Jade is a good product to use. It is fast and functional. It has some drawbacks too: a lot of features which could be implemented according to the DSSSL specification are not yet implemented. Also, it takes a lot of time to learn DSSSL. Jade is a program for expert users only.

Sgmlspl will pass any additional arguments on to the specification file, it can process them in the regular perl5 fashion. While sgmlspl is fully functional, it is not always particularly intuitive or pleasant to use. But it has the virtue that perl5 is very widely supported.

7 Summary

Within the framework of this project were installed a number of free and powerfull tools for SGML documents processing. Especially, the package

[7]Conversion of an Italian - Czech dictionary (930 kB) was about three times slower.

[8]Length of output from an Italian - Czech dictionary (930 kB) was about 150%.

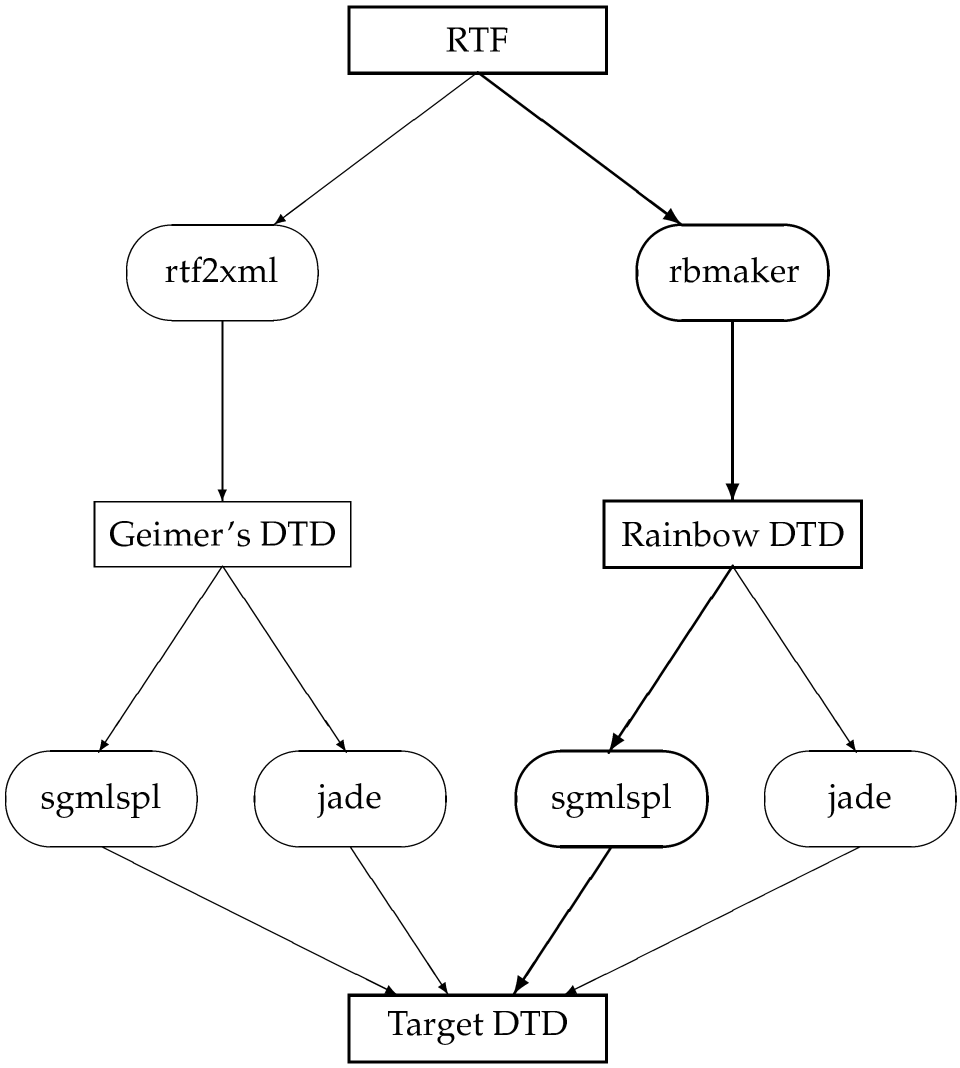

Figure 1: Data flow diagram

SP and the streaming programming language OmniMark provide wide spectrum of potential exploitation. Rainbow Maker and the filter rtf2xml allow two different view of RTF structure and their SGML or XML presentation. Jade and sgmlspl represent two different ways how to understand and to do SGML transformations.

There are two basic ways how to get a logical markup from a procedural markup of RTF documents.

The first way is to find direct relation between source and target markup. It is mostly very difficult and often impossible.

The major part of conversion engines that specialize in converting non-SGML documents to SGML documents use the second way. It is composed of two phases: the decoding of the source document, followed by its interpretation. In the concrete, the Rainbow Maker and rtf2xml allow decoding of the source markup into an existing presentation (Rainbow DTD) or specifically designed DTD (Geimer's DTD). This simple SGML or XML representation of source document allows easier processing in further conversion step. The interpretation of the first step is represented by transformation from an intermediate DTD to target DTD. It is possible using Jade (DSSSL — flow objects) or sgmlspl (Perl5 — nsgmls events).

Rainbow DTD has good support of the authors and other SGML developers. In addition, it represents universal view of several proprietary formats. Thus, it allows multiple exploitation of one and the same DTD. From this view Rainbow DTD looks as a better way. However, final choice will depend on better convergence of its markup model to target markup model.

Jade is an implementation of the DSSSL style language. If you have the knowledge of DSSSL (based on the Scheme programming language), it is easy and functional. But, it allows you to use only the implemented features of DSSSL. Sgmlspl is not always particularly pleasant to use. However, it allows you to use any programming constructions available in perl5 and only basic knowledges of programming language Perl are sufficient to understanding basic sgmlspl's functionality. Capabilities of sgmlspl are fully dependend on programmer's knowledges of Perl.

References

[1] Microsoft Technical Support: Rich Text Format (RTF) Specification and Sample RTF Reader Program, 4/97.

[2] Simon St. Laurent: Tvorba internetových aplikací v XML, Computer Press, 1999.

[3] Eve Maler, Jeanne El Andaloussi: Developing DTDs — from text to model to markup, Prentice Hall, 1996.

[4] David Sklar: The Annotated Rainbow DTD, Rainbow version 2.5, Electronic Book Technologies, 2/95.

[5] rtf2xml — The Free RTF to XML Convertor, version 0.7: `http://www.xmeta.com/omlette/rtf2xml/`.

[6] OmniMark C/VM 5.1 — An integrated OmniMark compiler and virtual machine: `http://www.omnimark.com/`.

[7] Jade — James Clark's DSSSL Engine: `http://www.jclark.com/jade/`.

[8] SP — James Clark's SGML Parser: `http://www.jclark.com/sp/`.

[9] Electronic Publishing Centre: Software evaluations `http://www.rug.nl/etc/eval/index_gb.htm`.

Publications in the FI MU Report Series are in general accessible
via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic