Grammar-Based Visual Software Modeling and Conversion

Jüri Kiho
University of Tartu
Institute of Computer Science
J. Liivi 2, 50409 Tartu, Estonia
Jyri.Kiho@ut.ee

Abstract

The problems of software conversion and automated converter construction have been investigated. It has been pointed out that conversion procedures involve some kind of modeling: any particular set of conversion rules constitute a conversion model. The suggested idea of automation of converter building is to separate the conversion model from source languages structures (grammars). If for each source language its grammar is defined and a parser (automatically) generated, then the conversion of parse trees (instead of source texts) to target texts can be described by the model. It allows to specify a general convert procedure which yields the result target text given a uniform parse tree (generated by the source text parser) and a model. A visual (sketchy) modeling implementation of the converters building process has been explained.

1. Introduction

In this paper the problem of software conversion is considered. Thereat no essential difference between program source texts and any other textual data can be observed. Conversion of a source text means giving it another form in accord with a set of conversion rules. The conversion rules define how and where the source primitives will be placed in target texts. It would be reasonable to consider any particular set of conversion rules as a (conversion) model.

Any particular conversion method addresses (1) a source language, (2) a target language and (3) a conversion model. Since conversion means just format transformation, both the source and the target language have the same set of primitives.

Often (but not always) the model defines the *two-way conversion*: from source to target and *vice verse*.

One of the important conversion types is Some-to-XML, for instance, Flat-to-XML [11]. Effective converters may impose rather strict constraints on source languages, some of them accepting, for instance, only so-called comma separated values (CSV). Obviously, restrictions of such kind are not tolerable in the case of many common software structures.

Conversion of type Some-to-XML combined with the conversions of type XML-to-Some provides the conversion Some-to-Some.

It should be noticed that using XML as an intermediate target language in some SL-to-TL converter is not always reasonable. Generally speaking, it involves the need to build two converters, SL-to-XML and XML-to-TL.

Some real examples where the straight converter would be preferred:

1. SL is the LinkGrammar language [8], where a text is a sentence in English with assigned to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. TL is a linkage diagram language expressing explicitly the structures assigned to source sentences.
2. SL is the language in which annotated dialogues in Estonian are written down [3]. TL is a visual language for source annotation schemes.

The need for converters of many different types, each performing conversion from one specific language to another one, raises the problem how to automate the process of converter building. Some of the options for automated content conversion into XML format are addressed in [9].

It is desirable, indeed, to build converters as general as possible. The ideal aim would be a converter for Any-to-Any. At a first glance this aim seems practically unreachable, but after certain reformulation of the problem, real steps in this direction can be made. The problem can be posed as follows.

Given a source language, a target language and a conversion model, automatically generate the corresponding converter. Some attempts to solve this problem have been made for conversions of type Any-to-XML, e. g. [12]. In fact, the problem has been further reduced: instead of the converter generator the model-dependent converters are developed which perform conversions according to given conver-

sion models.
In the following, a realistic process of constructing such converters is outlined. This approach assumes that a parser for any particular source language can be easily generated on the basis of the source language grammar. First, a general schema of building grammar-based converters is given. Further, a particular implementation of the process as sketchy modeling in the system Amadeus-fRED [6] is described.

2. General process of building grammar-based converters

Let SL be a source language, SG its grammar and TL a target language. Suppose that a conversion model M has already been defined.
To build the SL-to-TL converter according to M the following steps are to be performed.

1. Build the SG-based parser (SP) for SL which constructs parse trees for texts in SL.
2. Develop the method $convert$ which takes two arguments, a parse tree (of a concrete source text) and a conversion model, and yields the target text (the converted source text).
3. Specify the converter:

   - - - Let $SLtext$ be an arbitrary text in SL

   $parseTree = SP(SLtext);$

   $TLtext = convert(parseTree,\ M);$

   - - - Result: $TLtext \in TL$

Obviously, the third step is trivial and the first step can be quite easily automated using some parser generator. The most critical one is the second step: if dependent on SL and/or TL, a lot of programming work would be needed to develop the convert method anew for each next converter.

So it would be extremely desirable to have a universal convert method independent of source and target languages, i. e. which accepts models M for any SL and TL. In turn, it leads to the need for making the model structure independent of SL and TL.

One solution to this problem is to introduce a special model description language (yet another new language together with its GUI), such as the data extraction language DEL [10]. However, the sketchy modeling technique described in the following sections offers an alternative solution which is more uniform and user-friendly. Namely, the same visual diagram representation views are used for target languages as well as for conversion models. So, the conversion models become independent of source languages, though staying dependent on the (single) target sketchy language.

3. Sketchy modeling

The sketchy modeling method [7] [6] implemented in Amadeus-fRED offers a uniform and, at the same time, highly flexible framework for graphical representing, processing and editing software structures. It also serves as a basis for rapid development of specialized and structure-driven *ad hoc* editors for software of various kind. Because of the multi-language nature of Amadeus-fRED the conversions between various text formats are inherent to the system.
All texts processed are represented in an inner format (as linked Java objects) called the sketchy text language. Sketchy texts are assigned WYSIWYG display views (view attributes) from a list of available ones. Below, only one particular view (the sketchy view) is used for sample sketchy texts.
In figure 1 the sketchy text structure is acquainted. The *sketchy text* is a *sketch* (denoted by prolonged left bracket in the sketchy view). A sketch may have *primitive heads* and consists of one or more *branches*. A branch may have primitive heads and contains *primitive members* and/or sketches (the sub-sketches).
A sketch with just one branch is called the *mono-sketch*. In the following examples, mainly the mono-sketches are in use. Any sketchy text element (except comments) may have a comment.
Every sketchy text is assigned a *base language* (the base language attribute) – the source language the sketchy text (as conversion target) has been converted from.
A number converters have already been implemented for

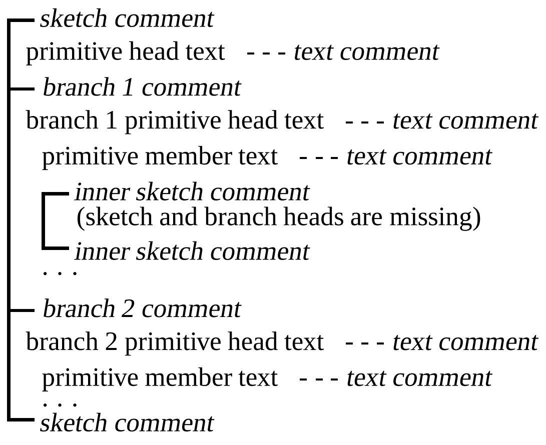

sketch comment
primitive head text - - - *text comment*
branch 1 comment
branch 1 primitive head text - - - *text comment*
primitive member text - - - *text comment*
inner sketch comment
(sketch and branch heads are missing)
inner sketch comment
. . .
branch 2 comment
branch 2 primitive head text - - - *text comment*
primitive member text - - - *text comment*
. . .
sketch comment

Figure 1. Sketchy text elements.

Amadeus-fRED, including Java-to-Sketches converter [2].
In the following, a simple source language SL is introduced for illustration purposes. Any text in SL describes a descending family tree (see example in figure 3).
The following description of converter development

process will be based on the sample case of converting family trees to sketchy text. Corresponding trial version of Amadeus-fRED is available [1].
It should be stressed that although the given example addresses some textual data conversion, exactly the same modeling scheme is valid for any kind of hierarchically organized software. In authors's opinion, a toy programming language example would have been much less informative.
Some approximate time estimations for converter building steps are included in square brackets [te: . . .]. The estimations are derived from author's personal experience while developing a converter for annotated dialogues [3].
To give the reader some flavor of the GUI used some menu commands in square brackets [mc: . . .] are pointed out too.

3.1. Grammar and parse trees

First of all, the grammar for SL should be specified, rewritten in JJTree/JavaCC [4] and debugged. At this stage cooperation between domain and grammar experts is necessary. It may take a considerable amount of time [te: up to 3 man-days] to specify the grammar. The grammar specification could be in EBNF or XBNF etc. and not necessarily describe all minor details. In figure 2 a sample grammar is presented.
Next, the grammar is detailed and given the JJTree/JavaCC

```
s ::= person
person ::= ”{” name (data)* (partnership)* ”}”
name ::= (<IDENTIFIER>)+
data ::= ”(” (˜[”)”]) ”)”
partnership ::= ”[” name (data)* (child)* ”]”
child ::= person
```

Figure 2. Grammar for family trees.

form and syntactically debugged [te: up to 1 man-day] until the sequential application JJTree-JavaCC-javac runs correctly and produces the first version of the parser (e.g. $Family00Parser.class$ in our sample case).
Independent of the last activity, the new base language is installed into Amadeus-fRED [te: up to 1 hour (if manually)]. Basically it involves making a new Java class ($BaseLanguageFamily00.java$ in our sample case) according to a pattern and including the new base language name and few related actions into the parent class.
At this point, Amadeus-fRED is ready to support further debugging the grammar. One can open [mc: *File+Read text*] a concrete source text (e.g. family tree in figure 3) and get [mc: *Tools+Parse* and *Tools+Reduce*] its reduced parse

```
{Jaan King (1743 – 1817)
   (Was born in Tartu. Moved to Edinburgh in 1780. )
   [ Mari Kass (1750 – 1777) {Kai King (1770 – ? )} ]
   [ Mary McQueen (1760 – 1835)
     (Church wedding in June 1783.)
   ]
}
```

Figure 3. Sample family tree (FT) as plain text.

tree (figure 4) or an error message. In the case of error, the grammar (or, perhaps, the source text) needs to be corrected. Developing the grammar ends, when a sufficient number of sample source texts have been accepted by the parser.
JJTree [5] enables to suppress generation of some nodes in parse trees. In the sample grammar the node ”IDENTIFIER” is suppressed as meaningless from the family tree point of view. So, for instance, in the parse tree (figure 4) the terminal symbols ”Jaan” and ”King” both become successors for the non-terminal node ”name”.
Parse trees are also displayed in the visual sketchy form: each non-terminal node, say nt, is represented by a sketch containing all descendant nodes of nt as its sub-sketches (figure 4). The name of the non-terminal is prefixed by ^{0} and shown as the sketch comment.
Non-terminal nodes which do not have non-terminal successors, i.e. have only terminal successor(s), are called the *primitive non-terminals*.
The primitive non-terminals in the sketchy parse tree are reduced to sketch primitive members. In such case the terminal values (e.g. ”Jaan” and ”King”) are listed in the primitive text and separated by ^{1} (e.g. ”Jaan^{1}King”); the name of the primitive non-terminal is prefixed by ^{0} and shown as the text comment (e.g. ”- - - ^{0}name”).

3.2. Primary model

Amadeus-fRED supports conversions of type Any-to-Sketches. Therefore it is natural to represent conversion models also in the sketchy form.
In Amadeus-fRED, the *sketchy model* is a sketchy text which serves as a template for target sketchy texts. It also includes ^{0}-prefixed non-terminal names (see figure 5) to establish the connections to parse trees of source texts.
In particular, if a non-terminal nt occurs in a comment of a sketch s in the sketchy model, every sub-tree with root nt in the parse tree will be converted to a sketch, having the same attributes as the sketch s. Likewise, if a primitive non-terminal pnt occurs as a primitive member p in the sketchy

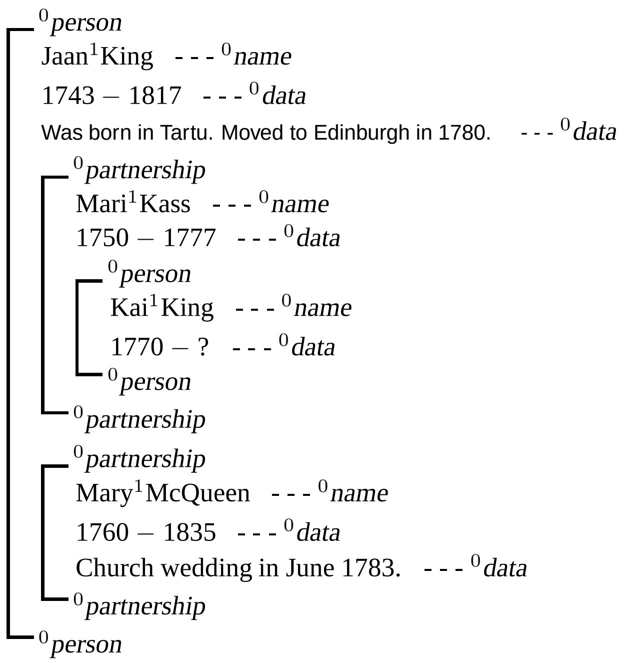

^{0}person
Jaan^{1}King - - - ^{0}name
1743 – 1817 - - - ^{0}data
Was born in Tartu. Moved to Edinburgh in 1780. - - - ^{0}data
^{0}partnership
Mari^{1}Kass - - - ^{0}name
1750 – 1777 - - - ^{0}data
^{0}person
Kai^{1}King - - - ^{0}name
1770 – ? - - - ^{0}data
^{0}person
^{0}partnership
^{0}partnership
Mary^{1}McQueen - - - ^{0}name
1760 – 1835 - - - ^{0}data
Church wedding in June 1783. - - - ^{0}data
^{0}partnership
^{0}person

Figure 4. Parse tree for FT as sketchy text.

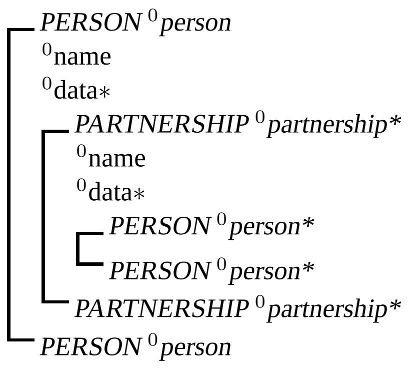


Figure 5. Primary model (M0) for family trees.

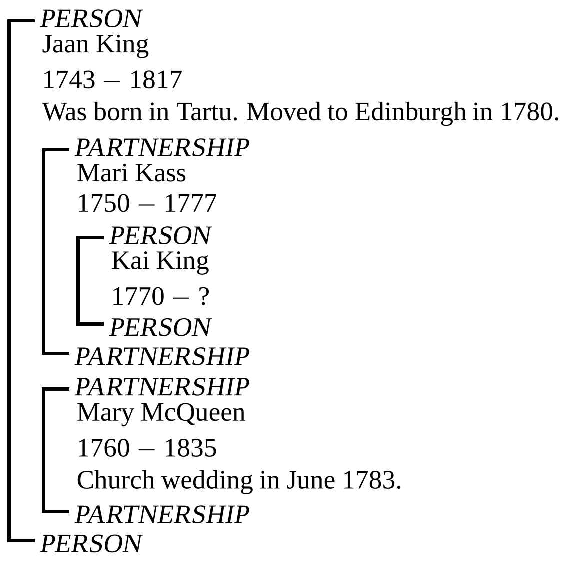


Figure 6. FT as sketchy text modeled according to M0 (in figure 5).

model, every sub-tree with root pnt in the parse tree will be converted to the primitive p. The primitive text is composed from the terminal successors of the node pnt.
In the model, the possible repetition of a construct in the target text is shown by an asterisk at the end of corresponding non-terminal name.
A primary model (figure 5) can be quickly obtained from a particular parse tree in which all (or most) of the non-terminals occur. For that, the parse tree should be simply modified (changed into the model) and saved like any other sketchy text in Amadeus-fRED.
When the sketchy model is complete, Amadeus-fRED is ready to perform conversions. One can open a source text [mc: *File+Read text*] and get [mc: *Tools+Sketchify*] the target sketchy text (like in figure 6) converted according to the model.
Usually, the primary model is not adequate and satisfactory for the domain. Nevertheless, it serves as a good starting point for further conversion model design.

3.3. Meaningful model

As an example, a more meaningful model for conversion of family tree texts is presented in this section (with some minor technical details omitted). First of all, it should demonstrate the ease of experimenting with different models in the Amadeus-fRED environment.
The model in figure 7 defines the following conversion. Each ”person” in a given source text is converted to a sketch. The latter has several primitive heads commented by the person’s ”data”; in addition, the person’s ”name” takes the place of the first primitive’s text. The sketch for a person contains several branches each corresponding to a ”partnership”. The head of the branch is similar to the head of sketch. Each branch, in turn, contains several sketches. Actually, the inner sketch in the model is optional and included only for clarity and may be considered a model comment. It does not affect the conversion, because the nesting of persons into partnership is determined already by the grammar and is present in every parse tree. In figure 8 an example of a newly modeled family tree is depicted. When the modeling process is accomplished, i.e. a satisfactory conversion model is elaborated, the backward conversion method (*textualize*) may be written and included into the

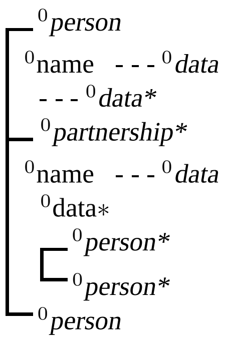
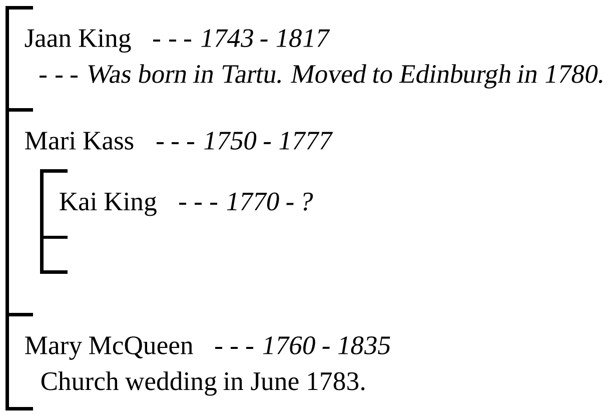


Figure 7. Another model (M1) for family trees.



Jaan King - - - *1743 - 1817*
- - - *Was born in Tartu. Moved to Edinburgh in 1780.*
Mari Kass - - - *1750 - 1777*
Kai King - - - *1770 - ?*
Mary McQueen - - - *1760 - 1835*
Church wedding in June 1783.

Figure 8. FT as sketchy text modeled according to M1 (in figure 7).

corresponding base language class [te: 2 man-hours]. It enables to convert target sketchy texts back into source language texts [mc: *Tools + Textualize*].
If the choice of views currently offered by Amadeus-fRED does not include a convenient one, a new WYSIWYG view can be programmed and included into the system [te: 2-3 man-days].

3.4. XML-oriented models

Amadeus-fRED supports the two-way conversion XML-to-SketchyXML, i.e. the base language XML has already been installed. The conversion model is rather straightforward: any XML element (say, e) converts to a mono-sketch (se) which contains all sub-elements of e as sub-skethces. The tag of e becomes the comment of se and the attributes of e are placed in the head of se.
By the way, it means that any sketchy text t which contains only mono-sketches without heads (like in figure 6) can be interpreted as an sketchy XML text as well. Consequently, t can be assigned the base language XML [mc: *Base+Set*] and thereafter converted to XML [mc: *Tools+Textualize*]. Applying these operations for the sketchy text in figure 6 yields the XML text in figure 9.
To get perhaps more meaningful sketchy XML for family trees, a more appropriate conversion model should be used. For instance, the model in figure 10 reflects the wish to use attributes for person/partner names and for the first portions of their data.
Indeed, it is quite natural to assign all sketches in the XML-oriented model to base language XML.
The sample result of such conversion is depicted in figure 11. Since the model's base language is XML, the result sketchy text also has the same base language. So the corresponding plain XML text (in figure 12) can be obtained immediately [mc: *Tools+Textualize*].

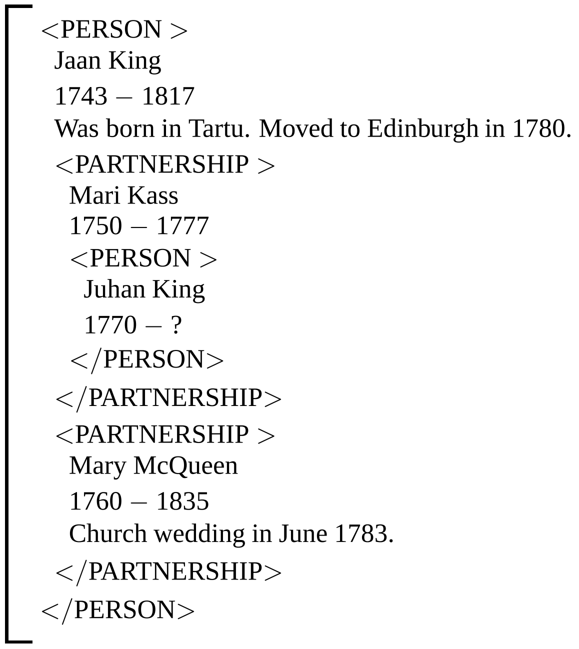
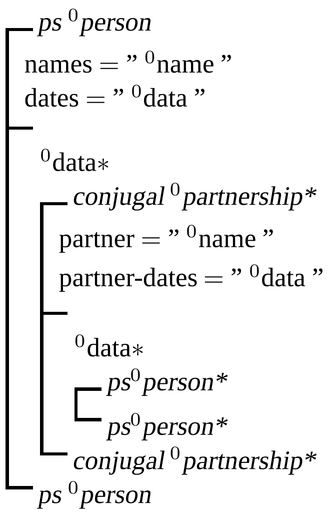

```
<PERSON >
 Jaan King
 1743 – 1817
 Was born in Tartu. Moved to Edinburgh in 1780.
 <PARTNERSHIP >
  Mari Kass
  1750 – 1777
  <PERSON >
   Juhan King
   1770 – ?
  </PERSON>
 </PARTNERSHIP>
 <PARTNERSHIP >
  Mary McQueen
  1760 – 1835
  Church wedding in June 1783.
 </PARTNERSHIP>
</PERSON>
```

Figure 9. FT from figure 6 textualized to XML.

4. Conclusion

The problems of software conversion and converters building have been investigated. It has been pointed out that the conversion procedures involve certain kind of modeling. It would be reasonable to consider any particular set of conversion rules as a (conversion) model.
There exists a real need for converters of many different types, each performing conversion from one specific language to another one. In turn, it leads to the problem how

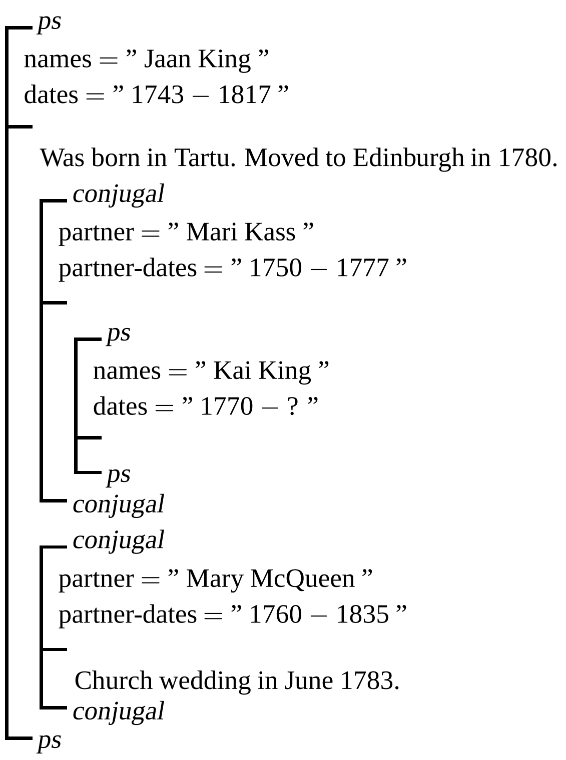


```
ps ⁰person
names = " ⁰name "
dates = " ⁰data "

⁰data∗
  conjugal ⁰partnership*
  partner = " ⁰name "
  partner-dates = " ⁰data "

  ⁰data∗
    ps⁰person*
    ps⁰person*
  conjugal ⁰partnership*
ps ⁰person
```

Figure 10. XML-oriented model (M2) for family trees.



```
ps
names = " Jaan King "
dates = " 1743 − 1817 "

Was born in Tartu. Moved to Edinburgh in 1780.
  conjugal
  partner = " Mari Kass "
  partner-dates = " 1750 − 1777 "

    ps
    names = " Kai King "
    dates = " 1770 − ? "

    ps
  conjugal
  conjugal
  partner = " Mary McQueen "
  partner-dates = " 1760 − 1835 "

   Church wedding in June 1783.
  conjugal
ps
```

Figure 11. FT as sketchy XML text modeled according to M2 (in figure 10).

to automate the process of converter building.

In this paper, an effective process of converter building is outlined. It is restricted to the conversions of type Any-to-Skt, where Skt stands for "sketchy text language". However, the sketchy text language is rather general mean for expressing structured computer texts of different kind. In many cases Skt may serve as the target language for conversions from less structured source languages. Visual handling of sketchy texts is fully supported by the system Amadeus-fRED. The latter already offers (two-way) conversions of type Java-to-Skt, XML-to-Skt et al.

The basic idea of automation of converter building is to separate the conversion model from source languages structures (grammars). Instead of determining how particular source language syntactic constructs should be converted, the model defines the mapping of parse trees onto the target language Skt. The format of parse trees does not depend on source languages: any parse tree itself is represented in Skt. It allows to specify a general convert procedure which yields the result target text given a parse tree (of a source text) and a model. The convert procedure has been implemented and integrated into Amadeus-fRED. The needed parse trees are provided by corresponding parsers which are separately generated using JavaCC/JJTree.

If one wishes to build a converter from a source language SL to the sketchy language, the following steps are to be performed:

1. Specify a grammar (SG) for SL.
2. Build a SG-based parser (a parse tree constructor) using JJTree-JavaCC-javac.
3. Install SL as a new base language for Amadeus-fRED.
4. In Amadeus-fRED, debug the grammar SG using the visual parse facility.
5. In Amadeus-fRED, develop the conversion model using standard visual editing mode and the conversion facility (sketchifier) to test the model.

Due to the embedded general-purpose converter, supplementing Amadeus-fRED with a new convertible base language takes only numbered work-hours. Inevitably, the work with grammar (steps 1-2) remains the most time-consuming activity. Often the source language appears to be not well-defined, especially in the case of some legacy or manually prepared data.

```
<ps names = " Jaan King "
 dates = " 1743 − 1817 " >
  Was born in Tartu. Moved to Edinburgh in 1780
  <conjugal partner = " Mari Kass "
   partner-dates = " 1750 − 1777 " >
   <ps names = " Kai King "
     dates = " 1770 − ? " >
   </ps>
  </conjugal>
  <conjugal partner = " Mary McQueen "
   partner-dates = " 1760 − 1835 " >
    Church wedding in June 1783.
  </conjugal>
</ps>
```

Figure 12. FT from figure 11 textualized to XML.

References

[1] Amadeus-Famibase. *http://www.cs.ut.ee/ kiho/Amadeus-Famibase/*. (Last visited: March 2004).

[2] Amadeus-Jasovi. *http://www.cs.ut.ee/ kiho/Amadeus-Jasovi/*. (Last visited: March 2004).

[3] T. Hennoste, M. Koit, K. Strandson, A. Raabis, M. Valdisoo, and E. Vutt. Directives in estonian information dialogues. *Text, Speech and Dialogue. 6th International Conference TSD*, Springer:406–411, 2003.

[4] JavaCC. *https://javacc.dev.java.net/*. (Last visited: March 2004).

[5] JJTree. *https://javacc.dev.java.net/doc/JJTree.html*. (last visited: March 2004).

[6] J. Kiho. *SKM. Sketchy Modeling of Computer Texts*. University of Tartu, Estonia, 2000.

[7] J. Kiho. Sketchy modeling for xml. *Proceeding of the 7th Symposium on Programming Languages and Software Tools SPLST'2001*, University of Szeged, Hungary:183–197, June 15-16 2001.

[8] LinkGrammar. *http://www.link.cs.cmu.edu/link/*. (Last visited: March 2004).

[9] R. Virk. *Transforming Unstructured Content into Meaningful XML*. *http://www.cambridgedocs.com/id16.htm?#introduction_xml_for_unstructured*. (Last visited: March 2004).

[10] X-FetchWrapper(R). *http://www.x-fetch.com/wrapper.html*. (Last visited: March 2004).

[11] XFlat. *http://www.unidex.com/xflat.htm*. (Last visited: March 2004).

[12] xmlspy(R)2004altova. *http://www.altova.com/features_convert.html*. (Last visited: March 2004).