Serdica
Bulgariacae mathematicae publicationes

Сердика
Българско математическо списание

SERDICA — Bulgaricae
mathematicae publicationes
20 (1994) 219-239

DOCUMENT SYNTHESIS BASED ON AN ANALOGY BETWEEN DOCUMENT DESCRIPTIONS AND HIGH LEVEL LANGUAGE PROGRAMS*

PETER H. BARNEV, VESSELIN N. IGRACHEV, VLADIMIR S. SHKOURTOV

ABSTRACT. The present paper deals with an approach to document synthesis based on combining text fragments prepared in advance. The fragments are arranged according to a scheme (skeleton) which defines the possible combinations. Skeletons resemble algorithm flowcharts. The characteristic features of the documents processed through this approach are defined. Tools of describing the generic logical structure of such documents are proposed. The problems related to the parametrization of fragments are discussed in detail. The general structure of a system named Syd2 based on the proposed approach is described as well. Some data about the Syd2 efficiency are given too.

1. Introduction. It is well known that functionally close control structures are used in conventional algorithmic languages. At first control structures were designed so as to describe different numeric calculations. The same control structures were successfully used later for describing a number of other routine activities, for example, sequencing of computer jobs, manufacturing technologies for CAM systems, drawing procedures for CAD systems, etc. Special-purpose languages were created (e.g. advanced job control languages such as that of the standard UNIX shell [Bour78]). Our investigations have shown that control structures can be used for describing the process of synthesis of documents created through combining preliminarily prepared text fragments.

An approach to automating office clerks' routine activities related to document synthesis is proposed in this paper. The documents operated with are called t-documents (from typed documents). T-documents counter some constraints given in Section 1. The basic notions related to t-documents are defined in Section 2. The problem of specifying the logical structure of t-documents is discussed in Section 2 as well.

*The study described in the present paper has been supported by the Bulgarian Academy of Sciences (project No 1001003)

A method for the parametrization of t-documents is proposed and studied in Section 3. The general structure of a programming system intended for t-document processing is outlined in Section 4.

A number of authors believe that clerks' everyday work in most offices consists of creating documents of similar structure and that activity takes about 19-29% of their working time [Kell85]). Our study on the work in some Bulgarian offices confirmed this statement.

The most frequently used documents are blank forms which are filled in by employees when necessary. Forms are a special kind of documents and they are characterized by:

- a standard part(s) of the text printed in advance;
- a fixed logical structure;
- a uniform layout.

Usually forms are provided for frequently created documents of relatively simple contents.

Forms aim at:

- saving employees' time due to reducing the amount of information filled in by them;
- limiting the probability of making errors in the process of document creation;
- facilitating the non-automated screening of documents (an employee can easily read particular fields).

The form-oriented systems are widespread and well known software products in office automation.

Our study is focused on documents which though often used in office practice have no preliminary designed forms. The analysis of such documents shows that in most cases they are particular instances of a document class (in terms of ODA standard [ISO86]) which differ in their logical structure. The layout of these documents is rather uniform. They merely look like standard free text formatted into lines, paragraphs and pages. Designing forms for such documents is either inconvenient (most fields are rarely filled in, the contents of fields slightly varies in instances) or impossible (for example, often a free text of unpredictable length should be inserted into the documents). Examples of such classes of documents are:

- minutes of sessions of scientific, administrative, etc. councils;
- notarial acts (muniments), sentences and other documents from judicial and legal practice;
- business letters referring to standard occasions.

Main properties of these documents are:

- They are routinely created by a specialist defining a set of text fragments and rules of document constructing by these text fragments;

– Documents consist of natural language sentences. The layout structure defines a standard free text (formatted into lines, paragraphs and pages);

– Some of the documents have similar generic logical structure in the sense that the text fragments they are built of appear in a partially predefined order.

Documents satisfying the above mentioned constraints are called *t-documents*. Like forms, t-documents can also be well standardized, but in contrast to forms they deal with deviations due to the generic logical structure of documents.

Our study has shown that 80% of the documents used in some Bulgarian offices are forms, about 10% are t-documents and the remaining 10% are impossible-to-standardize documents. These percentages may vary considerably in different offices.

2. T-document Model. The investigations described in the present paper are focused on the problem of modelling the process of t-document creation. We decided to deal with t-documents because forms have been studied for a decade [Tsic82] and a number of commercial form-oriented office systems are available now. Even specialized form processors (e.g. Form Base of Xerox) have already been developed. But on the other hand there exist documents which cannot be standardized. Their individual structure makes them quite unappropriate for automated synthesis.

Next in the text for brevity we shall use the word document(s) instead of t-document(s). The investigations treated in this paper are based on text documents only. We believe that our approach could be extended to multimedia documents [Tsic85] as well. However, corresponding study has not been initiated because our observations have shown that no-text documents are not intensively used in Bulgarian offices.

2.1. T-document Structure. The main property of t-documents is that they can be divided into *document classes*. Each class is based on the set of text *fragments* and it comprises documents consisting of elements of this set (precise definitions will be built in the text iteratively). So, documents with one and the same purpose (reason for appearance) and similar contents form a class. The notion of the document class introduced corresponds to that of ODA standard.

Documents belonging to a given class are referred to as *instances* of that class. At that early stage of the exposition we can only say that the instances of a class can vary in the number and ordering of their text fragments.

The process of selecting fragments from real documents of one and the same class is illustrated in Fig. 1*

The requirement that the texts of the fragments should be identical every time they are used in a document seems to be quite restrictive and unnatural. That is why

*The lack of original documents in English hampered us in finding examples for the present paper. The direct translation of Bulgarian documents is inadequate for the peculiarities of Bulgarian as a language. Therefore we had to use some fabricated examples in order to illustrate our method. The example is taken from everyday life of researchers. It could hardly be accepted as a typical office document but we have chosen it for its comprehensibility.

23.02.1990	Date	01.03.1990
Poland Warsaw IPI - PAS Prof. Miroslav Dombrovsky	Recipient address	USSR Kiev Gorsystemotechnica institute Prof. A. Stogniy
Dear Prof. Dombrovsky,	Salutation	Dear Prof. Stognij,
Thank you for your kind invitation. It is a big honour for me to participate in the conference you organize. It would be a pleasure for me to meet my colleagues in IPI-PAS again and to learn more about the achievements and problems of informatics in Poland.	Confirmation	
	Excuse & replacement	I would like to thank you for your kind invitation and to inform you that unfortunately I am not in a position to participate in the conference you organize because of engagements already taken. In our institute several colleagues of mine work in the field you are interested in. I would like to recommend some of them.
After I have been acquainted with the topics of your conference I think that a suitable subject of my lecture might be "Methods for Document Synthesis". I would ask you to inform me how you find the proposed subject till April 6 1990 in order to have enough time to prepare the lecture.	Offered lecture	
	Info about substitute	Vasil Slavchev was born in 1949. In 1977 he received his Doctor's degree in the University of Sofia. Dr. Slavchev has worked in the IEI - CNR in Pisa - Italy. He has been working with us since 1977. The main field of his interest is fuzzy sets. Peter Ivanov was born in 1948. In 1981 he received his Doctor's degree in the University of Sofia. Dr. Ivanov has worked in the Faculty of Mathematics and Informatics at the University of Sofia. He has been working with us since 1976. The main field of his interest is object-oriented systems.
I am looking forward to our meeting. Sincerely yours.	Conclusion C	
	Conclusion R	My colleagues Dr. Slavchev and Dr. Ivanov have not any engagement during the time in question. Each of them has agreed to come to your conference. You may contact them at the same address. I would like to express my regret again but I hope another opportunity will arise soon to meet you and your colleagues. Sincerely yours.
	Enclosure	

The fragments used in these two instances of one and the same class are defined. Letters intended to invite participants in a certain scientific event are related to that class. Some of the fragments can be used in these two instances, others – only in one of them. There exist some fragments which might be inserted several times in one and the same instance.

Figure 1. Segmentation of Two Document Class Instances (Letters) into Fragments

partitions may be marked in the text of each fragment which will be formed in the process of instance synthesis. Partitions of that type will be referred to as *fragment parameters*. An example of a fragment and fragment parameters are given in Fig. 2.

```
<<Info about substitute>>

     ... was born in ... . In          Prof. Ivan Penev    1949
... he received his ... degree         1986    Doctor's
in the ... . He has worked in          University of Moscow
... . He has been working with         University of Sofia
us since ... . He is interes-          1979
ted in ... .                           office automation
```

Each fragment has a unique name. The locations where parameter values should be inserted are marked by dots. Sample parameter values are given on the right hand side of the fragment. Substitutes of theirs have to be inserted by replacing the corresponding dots so as to obtain the whole text

Figure 2. An Example of a Fragment with Fragment Parameters

Now we can define more precisely instances of a given class as documents that vary in number, ordering and parameter values of their fragments.

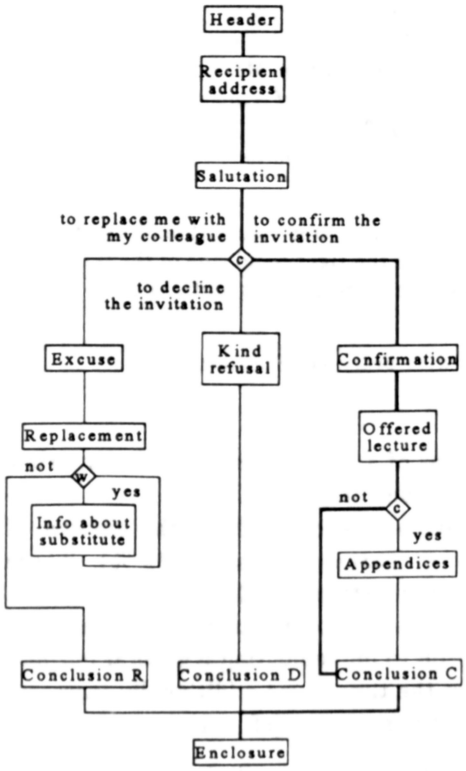
The fragment text consists of a constant part (about 61% of the number of characters at the average) and a variable part, i.e. parameters (about 39 %). These data have been found out due to studying about 300 instances of 15 document classes. In other words, the use of the fragment parameters approach can result in saving a lot of time for preparing a document in comparison to implementing a general purpose text editor. As the reader will be convinced later the time saved additionally increases with the use of a mechanism for naming fragment parameters (cf. 4.1).

2.2. Skeletons. As mentioned the instances of a document class are formed by one and the same elements – fragments. Each fragment can be inserted into an instance as many times as necessary (if at all inserted). A fragment is inserted into the text of an instance according to a certain scheme. The scheme describes a set of modes of combining fragments. Each mode is characterized by a certain degree of freedom which varies from strictly fixed forms to free hypertext systems. We refer to the scheme that determines the way of combining fragments as a *skeleton*. The use of skeletons requires fragments to be arranged in a way similar to that of actions in flowcharts (Fig. 3). Skeletons provide an intermediate choice between forms and hypertext systems, but are more close to forms. Our experience gained has shown us that such an approach is quite useful for processing t-documents that can strictly be standardized. The main advantages of skeletons are:

— users are supported at each step and thus a number of probable errors can be avoided;

— fragment selection manipulations can be reduced;

— instances can be standardized.



February 23 1990
Sofia

Poland
Warsaw
IPI - PAS
Prof. Miroslav Dombrovsky

Dear Prof. Dombrovsky,

I would like to thank you for your kind invitation and to inform you that I will participate in the conference you organize with a great pleasure. It will be a big honour for me to meet my colleagues in IPI - PAS again and to learn more about the achievements and problems of informatics in Poland.

After I have been acquainted with the topics your conference I think that a suitable subject my lecture might be "Methods for Document Synthesis". I would ask you to inform me how you find the proposed subject till April 6 1990 in order to have enough time to prepare the lecture.

I am looking forward to our meeting.

Sincerely yours.

An instance of a letter (confirming the participation of a lecturer in a conference) is formed by using a document class skeleton according to which the fragments needed are inserted where necessary. The locations where a selection is made are denoted: by **c** – a case type selection and by **w** – a while type selection. The traversed path, corresponds to the relevant instance marked by bold lines.

Figure 3. An Example of a Document Skeleton and a Document Class Instance

On the basis of observations on the documents of the same class the following variants for combining fragments have been chosen:

– obligatory insertion of a fragment or a fragment sequence in all the instances of a given class;

– selection of a fragment sequence out of several predefined ones and inserting it in a certain instance;

– insertion of fragment sequences for a non-predefined number of multiple insertions;

– specification of a fragment text through the peculiarities of each of its instances;

– iterative insertion of a fragment sequence for a number of iterations given in advance or calculated in the process of synthesis;

– selection of one of several predefined sequences of fragments with or without a capability of multiple selection.

Initially [BaNg86] the first four options have been used. They sufficed to describe the sequence logic of each t-document. However, processing turned out to be quite clumsy. That is why the last two options were added later [BaIS89].

A graphical example of a skeleton is given in Fig. 3. The well-known notations of flowcharts are used. This skeleton corresponds to the document class considered in Fig. 1.

Each skeleton can be presented through a directed graph which resembles the well-known flowcharts used for describing algorithms. Each fragment corresponds to a graph node. Each graph has an initial node and one or more terminal nodes. Each path from the initial node to a terminal one defines a valid (permissible) way of combining fragments.

Two sequences of fragments are given in Fig. 4 so as to illustrate some valid ways of combining fragments.

```
<<Header>>                      <<Header>>
<<Recipient address>>           <<Recipient address>>
<<Salutation>>                  <<Salutation>>
<<Excuse>>                      <<Kind refusal>>
<<Replacement>>                 <<Conclusion D>>
<<Info about substitute>>       <<Enclosure>>
<<Info about substitute>>
<<Conclusion R>>
<<Enclosure>>
```

Figure 4. Two Valid Fragment Combinations

In short we can make the more formal conclusion that each document class can be defined by an ordered couple *<set of fragments, skeleton>*. In addition we can say that through the class definition we specify the generic logical structure of the class in terms of the ODA standard. Some aspects of instance formatting (e.g. date, address and salutation, Fig. 3) have been specified in the texts of the corresponding fragments.

So, the generic layout structure is also specified by using the set of fragments. This way of specifying is consistent with the simplified layout structure of t-documents.

The common syntax orientation makes our approach similar to the hierarchical document model [Rabi85]. Our approach does not claim to be universal, but provides specific means of describing the generic logical structure of t-documents.

2.3. A Skeleton Description Language. A specialized skeleton description language can be created on the basis of the above mentioned adopted variants of combining fragments. A language, named *EMO* was created and implemented in the process of the experiments. The syntax of that language is given below*:

```
(1)  Skeleton          ::= Sequence
(2)  Sequence          ::= Directive | Sequence Directive
(3)  Directive         ::= Prompt | Directive_REPEAT | Directive_SELECT
                           Directive_REPSEL | Directive_CYCLE
(4)  Prompt            ::= <Text> | <<Fragment_name>>
(5)  Directive_REPEAT  ::= REPEAT (Alternative)
(6)  Alternative       ::= Prompt | Alternative Directive
(7)  Directive_SELECT  ::= SELECT (Alternatives)
(8)  Alternatives      ::= Alternative | Alternatives ; Alternative
(9)  Directive_REPSEL  ::= REPSEL (Alternatives)
(10) Directive_CYCLE   ::= CYCLE (Enumerated_type_name ; Sequence)
```

The number of alternatives in rule (7) should be at least two. Also only one "empty" alternative at the most is permitted. An "empty" alternative is an alternative which results in no other action except for leaving the current `Directive_SELECT`.

The description of the sample skeleton through our specialized language is shown in Fig. 5.

Since a skeleton is prepared before creating any instance of a relevant class, it is convenient to reduce the skeleton description by compiling it to a simple **intermediate language**. The intermediate language resembles low level programming languages. It consists of a small set of instructions suitable for a compact description of skeletons. In the process of compilation the text of fragments is processed and a **fragment table** is obtained as a result. It consists of some fragment characteristics (name, length, code, etc.). In the process of synthesis the skeleton description in the intermediate language is interpreted by making use of the fragment table. The synthesis process is considerably speeded up due to the simplicity of the intermediate language. In the process of synthesis a path is formed (traversed) by connecting the initial node and a terminal one as shown in the graph depicted in Fig. 3. If a certain node is a fragment, then the relevant text is inserted into the instance. Fragment parameters are set up too (this process is explained in detail in Section 3).

*The angle brackets are used here as terminal symbols.

```
<<Header>>
<<Address>>
<<Salutation>>
SELECT (
  <to confirm the invitation>
     <<Confirmation>>
     <<Offered lecture>>
     SELECT (
       <without any papers applied>
        ;
       <with some papers applied>
        <<Appendices>>
            )
     <<Conclusion C>>
     ;
  <to replace me with my colleagues>
     <<Excuse>>
     <<Replacement>>
     REPEAT (
       <<Info about substitute>>
            )
     <<Conclusion R>>
     ;
  <to refuse the invitation>
     <<Kind_refusal>>
     <<Conclusion D>>
       )
<<Enclosure>>
```

Figure 5. A Skeleton written in EMO Language

2.4. Analogy Between T-document Descriptions and Programs. One can easily notice the close resemblance between the above discussed variants of combining the considered fragments and the control structures of a High-Level Algorithmic Language (HLAL), or more general, between a skeleton description in EMO and a program in a HLAL. As seen from Table 1, this analogy is deep and comprehensive.

If we compare the two sides we would notice that the concept of procedure, being of great importance in a HLAL, is missing on the right hand side of Table 1. At first one might take the concept of fragment as that of procedure. However a deeper insight into the matter shows that this is not the case since fragments have the same rank in contrast to procedures in a HLAL.

t-documents	programs
instance synthesis	program execution
instance (sequence of fragments along a given branch of the skeleton)	the result of a program execution (understood as the sequence of actions along a given branch of the algorithm built in the program)
skeleton	flowchart of the program
fragment	one or group of unconditional statements
parameter	variable
functional dependency between parameters*	expression
synthesizing directive	control statement
`directive_SELECT`	case statement
`directive_REPEAT`	while statement
`directive_CYCLE`	for statement

Table 1. Correspondence of Notions of t-documents and Programs

The first purpose of the HLAL procedure technique is to enable users to decompose complex algorithms into simpler ones. Since the documents dealt with are not very complex, to provide such tools seems to be useless.

The second purpose of the HLAL procedure technique is to enable users to distinguish problems met often in different programs. Our experience gained has shown that some items of a certain document class often become items of another. We believe that the development of our approach in this aspect is interesting and promising.

Our study has been based on the analogy expressed in Table 1. As commented above this analogy may be a a further source of ideas of enriching our approach with other well examined techniques from the field of programming automation.

*This notation will be introduced later (cf. 3.3.)

2.5. Document Filing. The specific character of our approach allows the instances of a document class to be stored in a compact way. For that purpose it is sufficient to store information only about the order of sequencing the participating fragments and the values of parameters received. We call all this information *track* of the instance. With respect to filing we distinguish short-term and long-term instances. The tracks of short-term instances are stored in the work area which is shared by all the document classes. Long-term instances are those which have to be accessed afterwards. It is the user who explicitly transfers instances from the work area to the archive. The archive is divided into subarchives. Each subarchive comprises the instances of a given class. The organization of the archive and the retrieval techniques will not be discussed in the present paper in detail.

3. Fragment parameters. Parameters provide information typical of an instance. Each fragment can be tuned through them and inserted into a certain instance. Each parameter can be characterized by the following optional attributes:

- parameter name;
- mode of parameter value setting;
- parameter type;
- parameter prompt.

3.1 Naming parameters. The aim of naming parameters is to ensure the duplication of an entered parameter value at another location within a given instance. Our practice has shown that this happens quite often. For this reason it is enough such a parameter to be named and the parameter value saved. Obviously only parameters which will be cited afterwards should be named. The value of a named parameter has to be entered only when met for a first time. In each next coming across with this parameter name the already known parameter value is to be used.

```
<<Info about substitute>>*
        <name> was born in .. .  In .. he received
 his .. degree  in the .. .  <name> has worked in
 .. . He has been working with us since .. .  The
 main fields of his interests are .. .
```

The name of the substitute is denoted by <name>. It has to be entered only when first found. Its second appearance will result in copying the value entered at first.

Figure 6. An Example of Named Parameter

*Different aspects of the parametrization of fragments are illustrated by using one and the same fragment. In order to emphasize a certain aspect of parametrization, fragment texts have been slightly changed.

3.2 Parameter Value Setting. Three kinds of parameters can be distinguished with respect to their value setting:

– input parameters;

– parameters whose values is a combination of already known parameter values;

– global parameters whose value can be obtained by synthesizing program environment data (e.g. date, clock, etc.).

If we go back to the analogy made between t-documents and programs, we can notice that the enumerated three kinds of parameters correspond to the different ways of variable setting in a HLAL, namely: through an input statement, through an assignment statement and through the operating system.

3.3. Functional Dependencies of Parameters. A *functional-dependent parameter* is called a parameter whose value is calculated through the value(s) of other parameter(s) – argument(s). The relationship between a functional-dependent parameter and its arguments will be referred to as *functional dependency* (FD). The mechanism of naming parameters plays an important role in specifying FD. All parameters which are arguments in an FD have to be named. If a functional-dependent parameter has to be used later, it must be named too.

A FD can be defined for parameters within a certain fragment and in different fragments.

Some of the FD arguments might be results of the same or another FD. Nesting of FDs is permitted. FDs can be assumed to be relevant to HLAL expressions. The difference is that the only possible operation is the superposition of FDs. Our experience has shown that the depth of FDs nesting is not great.

Taking into account the analogy made with a HLAL, the set of FDs of parameters can be treated as a set of standard functions in a HLAL. However, this analogy cannot be complete, since FDs depend entirely on the natural language used, on different conventions adopted in particular countries, firms and organizations. Two aspects of this dependency will be mentioned below:

– it is impossible to construct a set of FDs which does not depend on a certain natural language.

– it is possible for an algorithm for one and the same FD to vary in different natural languages.

The problem of defining the constituents of the FD library is in fact a problem of tuning the synthesizing system to the specific requirements of a certain workplace.

In the process of our study a set of 22 FDs often met in Bulgarian documents was derived [BaIS89]. Some of these FDs are typical of Bulgarian documents, e.g. date of birth and sex (male, female) derived from the personal identification number. Probably some of the selected FDs might be also found in documents written in other languages. For example: naming some simple arithmetic operations (sum, difference, etc.) in numeric parameters; expressing some numeric accounts as strings of words;

separating family names from first names, etc.

Our experiments have shown that the use of a mechanism of FD parameter calculation increases considerably the intelligence of the synthesizing system and reduces the amount of information to be entered (cf. 4.1.).

```
<<Info about substitute>>

    <titname> was born in .. . In .. he received his <deg>
degree in the .. .  He has been working with us since .. .
The main interests of =ADDRESSING(titname, deg) are .. .
                          (a)

      Prof. V. Slavchev was born in 1949. In 1976 he
   received his Doctor's degree in the University of
   Sofia.  He has been working with us since 1977.  The
   main interests of Prof. Slavchev are multimedia
   documents, information retrieval and fuzzy sets.
                          (b)

        titname          deg    ADDRESSING(deg, titname)

   Prof. P. Black        Dr.          Prof. Black
   P. Brown              Dr.          Dr. Brown
   Robert White           -           Mr. White
                          (c)
```

A sample FD named ADDRESSING with two arguments is used. The first argument is a person's scientific degree. The second argument is a person's name which may be preceded by a title. A suitable addressing is formed according to the argument values.

Figure 7. Examples of Functional-Dependent Parameters
(a) a fragment containing a FD
(b) a sample text generated from the fragment
(c) sample argument values and results of a FD.

3.4. Parameter types. Usually the value of a given parameter is not an arbitrary string. Some of the constraints imposed on parameter values related to its context can be described formally. Each parameter value has its specific range of definition. Some of the values are composite and have an internal structure. Thus we come to the concept of *parameter type*. The following data can be treated as examples of parameter types: date, personal name, mailing address, phone number, etc. The concept of parameter types is similar to that of data types in HLAL, especially

languages with advanced data types (e.g. Pascal, Modula-2). The concept of parameter types in the sense we use it is similar to that of object classes in the so called object-oriented systems [Wegn87].

```
<<Info about substitute>>
      <name>{PN} was born in {Y}. In {Y} he received
his {E:Degr} degree in the .. . <name> has worked in
.. . He has been working with us since {Y}. The main
fields of his interests are {S:Fields}.

The following parameter types are used:
{PN}        - Personal name
{Y}         - Calendar year
{E:Degr}    - Element of standard enumerated type consisting of
              scientific degrees
{S:Fields} - Set of elements of enumerated type  consisting of
              computer science fields
Type checking is not accomplished for the parameters marked by dots.
```

Figure 8. An Example of Parameter Types

The main benefit of introducing parameter types is in the more detailed definition of permissible values or ranges of values obtained through parameters. Also, the opportunities for parameter values checking and document consistency maintenance increase.

The set of parameter types should be flexible and adaptable to the needs of each office. It depends on the natural language used, on the inter-firm conventions, etc. A set of 23 parameter types, found in Bulgarian documents very frequently, has been derived [BaIS89]. The *enumerated parameter types* are of special interest for their frequence of usage. They take values consisting of one or more elements of a base type defined in advance. In the process of synthesis the parameter value of such a type is selected from the set of values instead of being entered.

The elements of the base type might be whole phrases as well. For example if a user wants to vary the contents of an instance s/he may select a phrase of identical meaning from the set of phrases. Note that the longer the base type elements are, the greater the linguistic problems with respect to inserting them into the text of an instance are. It is not a common practice to use long phrases in t-documents. Probably document synthesis based on standard phrases is a way to deal with some of the difficult-to-standardize documents. This problem is studied by our colleagues [KeNN90].

Our observations have shown that sometimes untrained users have some diffi-

```
<<Info about substitute>>
    <name>[Name of the substitute?] was born in [Birth year?].
In [The year he received his last degree in ?] he receives his
[Last degree?] degree in the [University?]. <name> has worked
in [The places he worked for]. He works with us since [Year?].
The main fields of his interests are [Select!].
```

The prompt texts for parameters are put in brackets. They supply users with information about the semantics of the relevant parameters

Figure 9. An Example of Parameter Prompts

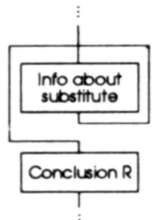
culties in satisfying parameter type requirements when entering parameter values. For this purpose a parameter value an opportunity may be provided for taking down the requirements of parameter type after a chosen number of failures.

The introduction of parameter types can be considered as an extension of the logical structure of a document. The ODA standard does not provide such a level of logical structuring but some authors think that such an extension is natural and does not contradict the notion of the standard generic logical structure of documents [Schu88].

3.5. Prompts for Parameters. The prompt for a parameter is a text message which appears when a parameter value is to be entered. It differs from the prompts in EMO (cf. 2.3.) which help users in the navigating processes. The prompts for parameters supply users with information on the semantics of the parameters of a certain document. The prompt text is defined by the class designer in advance.

3.6. Iterative Parameters. Each sequence of fragments used more than once is called an *iterative group (IG)*. Some problems related to the use of named parameters arise with IG.

If a parameter is cited within an IG and if its value is obtained out of that IG no special problems arise. The parameter value will be one and the same in each iteration. Problems arise with the use of a parameter quoted in an IG at which it acquires a value. Such parameters will be called *iterative* ones. It is not clear what the value of an iterative parameter (IP) within each iteration and after leaving an IG will be. The solution at which the value of an IP should be entered once in each iteration is the most natural one in such cases. Then within that IG, the IP may be quoted freely and its value will be the current one for the iteration. In the course of preparation for the next entering in the IG the current value of IP is stored as a consecutive element of an internal enumerated type (IET). This type and the relevant IP are given one and the



(a)

```
<<Info about substitute>>
    <*name> was born in .. . In .. received
his .. degree in the .. .  <*name> has been
working with us since .. .  The main fields
of his interests are .. .


<<Conclusion R>>
    My colleagues  <name>  have not any en-
gagement during the time in question.  Each
of them has agreed to come at your <event>.
```

(b)

```
    Prof. V. Slavchev was born in 1949.  In 1976 he received
his Doctor's degree at the University of Sofia.  Prof. Slav-
chev has been working with us since 1977. The main fields of
his interests are multimedia documents,  information retrie-
val and fuzzy sets.
    Dr. S. Alexandrov was born in 1957.  In 1987 he received
his Doctor's degree at the University of Sofia.  Dr. Alexan-
drov  has been working with us since  1984.  The main fields
of his  interests  are office automation,  document analysis
and synthesis and knowledge representation.
    Dr. I. Dimanov was born in 1960. In 1990 he received his
Doctor's degree at the University of Sofia.  Dr. Dimanov has
been working with us since 1987.  The main fields of his in-
terests are cognitive science, knowledge representation  and
knowledge based systems.
    My colleagues Prof. Slavchev, Dr. Alexandrov and Dr. Di-
manov have not any engagement  during the time  in question.
Each of them has agreed to come at your conference.
```

(c)

Figure 10. Examples of Iterative Parameters
(a) an iterative group
(b) the texts of the corresponding fragments
(c) a part of the sample instance

same name. In the next iteration a new value is entered for the IP and this value becomes a current one. Thus the process goes on until that IG is left. The quotation

of the IP after that moment will result in inserting a "generalized" value of IP in the instance. The "generalized" value is obtained from the elements of the corresponding IET as follows:

```
                    algebraic sum      when elements are numeric
"Generalized"       of elements
                =
    value           concatenation      when elements are symbolic
                    of separated
                    elements
```

The IET created in an IG may be used in the following course of synthesis as each other enumerated predefined parameter type (e.g. for loop organizing where for each element of the IET a single iteration is carried out). In Fig. 10 an example of a "generalized" value is given.

Our observations have shown that more than 2-3 IP in an IG can hardly be found in practice.

4. Implementation of the Proposed Approach

4.1. Project History. The problem of t-document synthesis has been studied by the Laboratory of Applied Mathematics since 1985. At first we aimed at creating and implementing an experimental system. The first version was written in Pascal and was completed in 1986 [BaNg86]. Some of the ideas described in the present paper originated from the implementation of that version. In 1987 the system was tested through pilot experiments in several organizations and we got some useful feedback information about its capabilities. The initial version and its pilot usage revealed some drawbacks of the implementation and led us to some new ideas of improving the method used.

In 1988-89 a new improved version named **Syd2** was designed and implemented [BaIS89]. The new variant was written in C. It comprised all the useful ideas of the old version and some new ones as well. Special attention was paid to the user interface. The Syd2 system is a software product intended for IBM/PC and compatibles. Some experiments with Syd2 have already been carried out so as to evaluate the efficiency of the parametrization apparatus introduced. The initial results obtained from 300 instances of three classes (5-20% of the parameter values are duplicated; 0-30% parameters are calculated by using FDs) show that the time benefit is about 15% and 30% compared to the first version.

4.2. Implementation Structure. The general structure of the Syd2 system is described in Fig. 11.

The monitor provides the user with unified environment for monitoring the other system components.

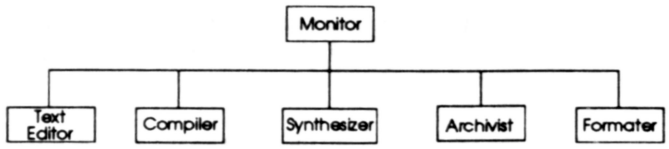


Figure 11. General Structure of Syd2 system

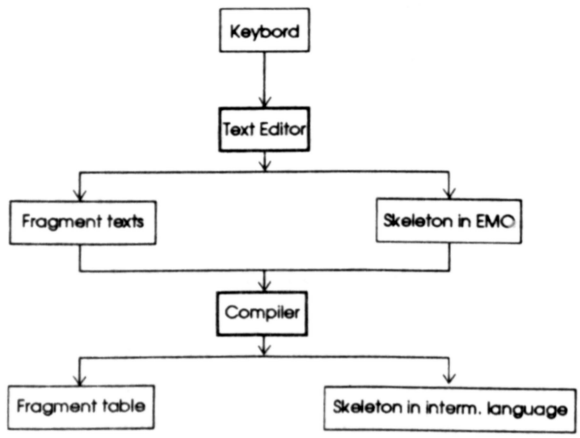


Figure 12. The Preparatory Stage

The text editor serves to create and update the initial skeleton descriptions and fragment texts for the document classes.

The compiler serves to transform the skeleton description of document class instances from an EMO language into an intermediate language.

The synthesizer is the main component of the system. It interprets the skeleton presented in the intermediate language and in interaction with the user creates an instance of the corresponding class. The synthesizer stores the names of the included fragments, organizes the entry of their parameter values and performs type checking or calculating. It provides the short-term storing of the synthesized instances.

The archivist serves to organize the long-term storage of the instances from different document classes and the access to them as well.

The formatter serves for outputting the instances as required by users (in a form suitable for displaying, printing, or storing them on the disk as text files).

The compiler is created on the basis of traditional techniques for such a kind of software products. A number of original elements have been built in the synthesizer

and archivist. Special attention has been paid to the synthesizer with respect to its efficiency and flexibility of interaction in the process of generating ready-made document instances. The archivist is not completely implemented yet.

4.3. Processing Model. Documents are created through the Syd2 system at three stages namely: preparatory, working and layout.

A document class is defined and formally described by an expert in EMO during the preparatory stage. That stage consists of:

– analysing paper documents which represent the class;

– defining the class by using the skeleton description and entering fragment texts;

– compiling the class definition.

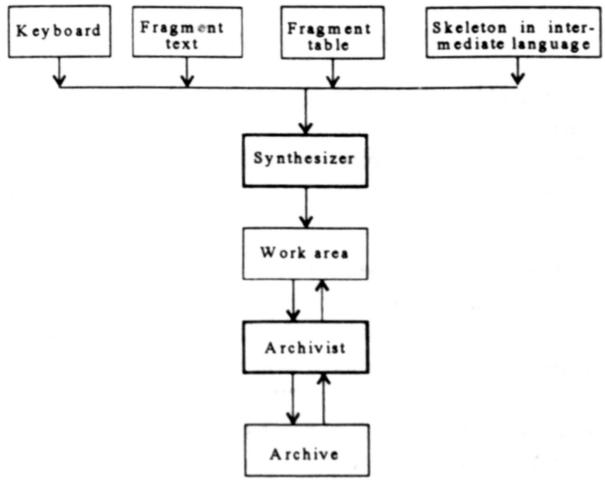


Figure 13. The Working Stage

The working stage (Fig. 13) consists of the synthesis of particular instances of a given class, as well as of instance archiving and retrieval. In the course of the synthesis of a particular instance by means of the synthesizer the user carries out the following activities:

– selects a document class;

– indicates the necessary fragments in accordance with the skeleton requirements;

– enters the parameter values for the traversed fragments.

Synthesis of a group of instances when they differ in a parameter value only (e.g. circular letter) is also allowed.

The working stage is the main one in everyday exploitation of the system and it is carried out by employees who are not experts.

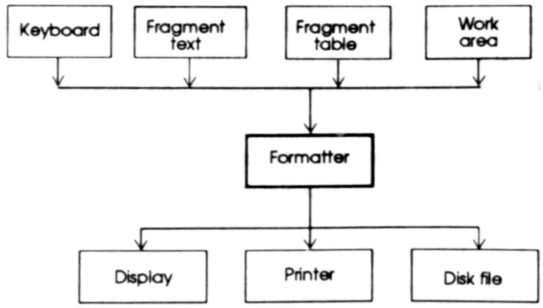


Figure 14. The Layout Stage

The layout stage (Fig. 14) is necessary since the instances are stored in a compact way. As mentioned above the instances are stored in the archive as tracks. The full text of the instance is rebuilt from its track at the layout stage. Then the rebuilt instance is output on the selected output device as specified by users (length of line, pages, etc.)

5. Conclusion. An approach and a completely developed method to text document synthesis were given in the present paper. The approach is based on the popular idea of assembling a unit from "ready-made" parts. It enables users to combine text fragments in accordance with a fixed scheme. This scheme is similar to the well-known flowcharts.

Our approach for automated document synthesis is only one of the possible ones. In the Laboratory of Applied Mathematics some other methods have been experimented too. They are based on the use of abbreviations, standard phrases, document patterns, etc.

The approach presented in this paper may be improved. We intend to focus our future work on:

- increasing the flexibility of skeletons;
- simultaneous synthesis of instances belonging to several interrelated classes;
- organizing a library which consists of fragments common for several classes, etc.

Acknowledgements. We are very grateful to Dr. Stephan Kerpedjiev who looked through two versions of the draft and made a number of useful remarks. We are thankful to Mrs. Nadka Gouneva-Ivanova who helped us edit the English version of the paper.

REFERENCES

[BaIS89] BARNEV P., V. IGRACHEV, VL. SHKOURTOV. Syntax-directed document synthesis – on the role of parameters. Proc. of the 18 Annual Spring Conference of the Union of Bulgarian Mathematicians., Albena 6-9 April 1989, (In Bulgarian).

[BaNg86] BARNEV P., DO VIET NGA. A system for automatized synthesis of free texts. KNVVT Conf. on Automatization of Inf. Proc. on PC, Budapest 5-9 May 1986, 15-26, (In Russian).

[Bour78] BOURNE S. R. An Introduction to the Unix Shell. *The Bell System Technical Journal*, **57** (1978), 2797-2822.

[ISO86] International Standard ISO/IS 8613/1..8, Information processing – Text and office systems – Office Document Architecture (ODA) and interchange format.

[Kell85] KELLOG R. T. Computers aids that writers need. *Behavior Research Methods Instruments & Computers*, **17(2)**, 253-258.

[KeNN90] KERPEDJIEV S., S. NEDKOVA, R. NIKOLOVA. On a Method of document synthesis using a phraseological dictionary. Proc. of the 19 Annual Spring Conference of the Union of Bulgarian Mathematicians., Sl. bryag 6-9 April 1990.

[Rabi85] RABITI F. A Model for Multimedia Documents. Office Automation (ed. D. Tsichritzis), Springer Verlag 1985, 227-250.

[Schu88] SCHULZE G. Office document architecture and its use in an office environment. Conference on Information Technology for Organizational Systems. Athens, 16-20 May 1988, 1025-1030.

[Tsic82] TSICHRITZIS D. Form management. *Comm. ACM*, **25 (7)** (July 1982), 453-478.

[Tsic85] TSICHRITZIS D. (ed.) Office Automation – Concepts and Tools. Springer Verlag, Berlin, Heidelberg, Tokyo 1985.

[Wegn87] WEGNER P. Dimensions of Object-Based Language Design. Brown University, Tech. Rep. No. CS-87-14, July 1987.

Laboratory of Applied Mathematics
15 N. Vaptzarov St.
4000 Plovdiv
Bulgaria

Received 13.04.1993
Revised 17.08.1994