SGML Nets: Integrating Document and Workflow Modeling[1]

Wolfgang Weitz
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe (TH), D-76128 Karlsruhe, Germany
wolfgang.weitz@aifb.uni-karlsruhe.de

Abstract

In this paper, we introduce so-called SGML nets as a new formalism for an integrated modeling of document structures as well as document manipulation processes. SGML nets are a variant of high-level Petri nets where each place (passive element, "document store") is typed using an SGML document type definition (DTD). Each place may be marked with a set of DTD-conforming document instances. Each transition (active element) specifies a class of operations on these document stores. Edges in SGML nets are inscribed with document templates. The incoming arcs of a transition select a set of instances to be read from the input places, while outgoing arcs define insertions into output places. The definition of the occurrence rule ensures DTD-conformance of the document instances in all places of the net at every moment.

1. Introduction

With the rising interest in making organizations more flexible and responsive to customer and market demands, business process re-engineering projects are widely expected to strengthen competitiveness. Modeling and analysis of a company's business processes are the key elements to uncover inefficiencies and facilitate the introduction of effective software support.

In this context, two different views have emerged which can be characterized as process-centered vs. document-centered. While both of them deal with the creation, manipulation, transfer and general management of business documents, they differ in what they choose as their key point of interest, documents or activities, respectively.

The process-centered approach tends to focus on activities such as the flow of documents between different workflow participants, while the documents themselves are represented as more or less unstructured "tokens". On the other hand, the document-centered approach tries to capture and take advantage of the often complex internal structure of documents to provide for example efficient means of document retrieval, version management and access control, while the modeling of activities like the manipulation and transfer of documents is usually not equally considered. These different perspectives are also mirrored in the respective modeling formalisms and the architecture of the associated software products, workflow management systems (WFMS) and document management systems (DMS).

A well-established language for modeling the behavior of dynamic systems (which includes business processes) are Petri nets. They allow for an integrated modeling of data objects and the specification of operations on these objects. Expressing concurrency is another strong point of Petri nets. Furthermore, their formal semantics permit to analyze the modeled system and to prove certain properties. This is a great advantage over many methods based on non-formal diagrams which are often used for describing business processes. The graphical notation of Petri nets makes them better accessible for non-experts. [22] reports on the advantages of Petri nets in a real-world project.

In the Petri net variants usually used for modeling business processes, individual documents are represented by unstructured "tokens", which ignores the complex internal structure documents often have. With SGML nets, we make an attempt to tightly integrate the modeling of document structures as well as document processing (workflow). The international standard SGML [13] is used to make the logical structure of documents explicit and thus accessible in the Petri net model of the workflow. As a consequence, parallel access to different parts of the same documents, decisions based on the actual content and the structure of a document as well as complex document creation and manipulation operations can be expressed within this model.

This paper is organized as follows: the next section deals with structured documents and SGML as an international standard in this area. Section 3 briefly introduces Petri nets. In section 4, SGML nets are introduced as a language to

[1]This work was supported by the Deutsche Forschungsgemeinschaft (German National Science Foundation) as a part of the research project INCOME/WF (Stu98/9).

model document manipulation processes based on structured documents. Possible extensions of this concept follow in section 5. After an overview of related work in section 6, section 7 concludes the paper with an outlook.

2. Structured Documents

The term *document* can be defined as "a structured amount of information intended for human perception, that can be interchanged as a unit between users and/or systems"[12]. The logical structure of a (text) document is often only given implicitly: a new section is introduced by a section heading, which may be typeset as a single line of text in a bold font of a certain size. While human readers are used to extract the logical structure of a text on the basis of these visual hints, efficient automated document retrieval and processing requires such information in an explicit form. Two international standards for *structured documents* have emerged: ODA (Open Document Architecture, [12]) and SGML (Standard Generalized Markup Language, [13]). We chose the latter as the basis for our work. The interested reader is referred to [7] and [5] for a discussion of these two standards.

2.1. SGML

SGML is a generic markup language. An SGML document consists of two parts: a *document type definition* (DTD) providing a grammar which defines a class of conforming documents, and an *instance* of this definition which carries the document content. The constituting logical parts of an instance, its *elements*, are explicitly marked by a pair of begin- and end-*tags*, which usually take the form `<TAG>...</TAG>`. We denote the portion of the document which is enclosed in an element's pair of tags as the *content* of this element.

```
<!DOCTYPE paper [
<!ELEMENT paper    - -
       (title,author,abstract+,text?) >
<!ELEMENT title    - - (#PCDATA) >
<!ELEMENT author   - - (#PCDATA) >
<!ELEMENT abstract - - (#PCDATA) >
<!ELEMENT text     - - (#PCDATA) >
<!ATTLIST abstract language
   ( en | fr | de | other ) "fr" >
]>
```

Figure 1. document type definition

Fig. 1 shows an example of a DTD for papers. It requires that every `paper` consists of (exactly) one `title`, the name of its `author`, one ore more `abstract`s, and an optional `text` element. The `abstract` element carries an additional *attribute* `language` which allows to specify in what language the abstract is written. Four permissible attribute values are defined, with `fr` as the default.

An example for an instance of the `paper` DTD is shown in Fig. 2.

```
<PAPER>
  <TITLE>Factoring Algorithm</TITLE>
  <AUTHOR>John Smith</AUTHOR>
  <ABSTRACT LANGUAGE="EN">
    This algorithm factors a prime
    number in constant time.
  </ABSTRACT>
  <ABSTRACT LANGUAGE="DE">
    Dieser Algorithmus faktorisiert
    eine Primzahl in konstanter Zeit.
  </ABSTRACT>
  <TEXT>
     ...
  </TEXT>
</PAPER>
```

Figure 2. document instance

Software systems like SGML-aware text editors and parsers can take advantage of the rigorous syntax definition in the DTD and thus support the writer of a document effectively in creating a complete and conforming instance of the DTD.

The elements of every SGML document are tree-structured (which is in essence the parse tree of the document instance wrt. the underlying grammar, as defined in the document's DTD). This conveys also a natural graphical representation of the document's hierarchical structure. Figure 3 shows the structure of the example document instance of Fig. 2 as a tree.

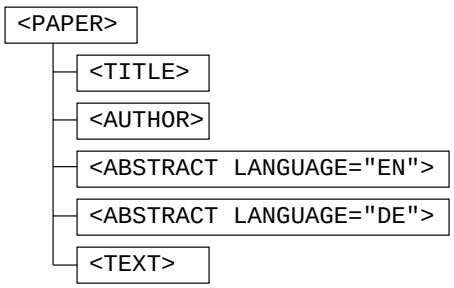


Figure 3. tree representation

3. Petri nets

Petri nets [6] provide a well-founded formal notation to model a dynamic system's behavior. Due to their graphical representation (see Fig. 4 for an example) and the possibility to introduce hierarchies of nets, even complex and highly parallel models are still manageable.

A *Petri net* is a triple $N = (P, T, F)$, where P is the set of *places* (passive elements) and T the set of *transitions* (active elements), $P \cap T = \emptyset$. Generally, a place may be

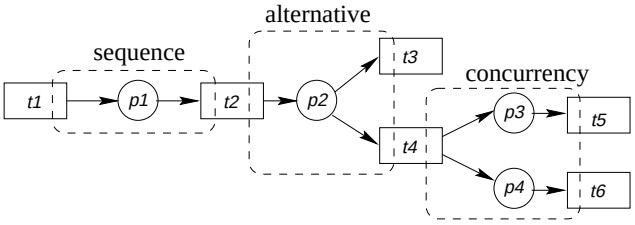


Figure 4. Petri net

interpreted as an object store, while a transitions specifies a class of operations on its adjacent places. In the graphical representation, places are depicted by circles and transitions by rectangles. The set of arcs $F \subseteq (P \times T) \cup (T \times P)$ is called the *flow relation* of N.

A place $p \in P$ is called an *input place* of a transition t if $(p,t) \in F$, and it is called an *output place* of t if $(t,p) \in F$. The set of all input places of t is denoted by $\bullet t$, while $t\bullet$ is the set of all output places of t. In Fig. 4, $p2$ is an input place of both $t3$ and $t4$, and it is an output place of $t2$.

A *marking* of a net N assigns a set of *tokens* to every place in N. It can be interpreted as a global system state. When a transition *occurs*, tokens are removed from its input places and inserted into its output places. The *occurrence rule* defines when a transition is *enabled* (i.e. it *can* occur) and (if so) how tokens in its input and output places are manipulated. As a result, the state of a transition t (enabled or not) is only determined by its direct vicinity ($vic(t) = \bullet t \cup t\bullet$), and in turn t will only manipulate places in its vicinity when it occurs.

Figure 4 also demonstrates some basic synchronization primitives in Petri net notation:

- sequence: *t2* consumes tokens from *p1* which may have been previously produced by *t1*.
- alternative: a token in *p2* can be consumed *either* by *t3* *or* by *t4*.
- concurrency: *t4* inserts a token into both *p3* and *p4*, which *simultaneously* enables *t5* and *t6*.

Petri nets where every token has an individual identity (and possibly structure) are called *higher Petri nets*.

4. SGML nets

SGML nets are a form of higher Petri nets. They specialize on the modeling of the processing of structured (SGML-) documents. The following sections describe the rationale behind this and introduce some central concepts.

4.1. Places

Places in SGML nets can be interpreted as stores for structured documents. Each place has an SGML style document type definition (DTD) associated with it[1]. A marking of a place is a set of valid document instances with regard to the grammar defined in the corresponding DTD.

For an unambiguous identification of document instances in a place, we introduce the notion of *keys*. A key is a set of SGML elements, the *key elements*. The DTD of a place is required to include at least one key declaration. For each key, every two instances in the marking of a place must disagree in the content of at least one key element. The DTD makes sure that every valid document instance contains at least one key exactly once (which implies that key elements must not be made optional in the DTD). Keys are specified in the DTD with a declaration of the form `<!KEY ele1, ele2, ...>`.

```
<!DOCTYPE paper [
<!ELEMENT paper    - -
       (title,author,abstract+,text?) >
<!ELEMENT title    - - (#PCDATA) >
<!ELEMENT author   - - (#PCDATA) >
<!ELEMENT abstract - - (#PCDATA) >
<!ELEMENT text     - - (#PCDATA) >
<!ATTLIST abstract language
   ( en | fr | de | other ) "fr" >
<!KEY title,author>
]>
```

Figure 5. DTD with KEY declaration

In continuation of our example `paper` DTD of Fig. 1, we have defined a key consisting of the elements `title` and `author` in Fig. 5. It requires that there are no two papers with the same author *and* the same title in a place typed with this DTD.

4.2. Edge Inscriptions

To define document processing operations, we need a way to select document instances (or parts of them). Since every document instance has a logical tree structure given by its accompanying DTD, we define a graphical form of regular tree expressions, so-called *document templates*, for this purpose. Following the idea of "Query-by-Example" as described in [23], document templates resemble the graphical tree representation of an SGML instance, enriched with a few additional operators. Thus, an example document may be used as a starting point which can be transformed into a document template by deleting some irrelevant parts or adding some operators.

Document templates are used to inscribe arcs in SGML nets. Viewed from a transition's perspective, we need to differentiate between two types of arcs, as shown in Fig. 6.

[1] The key point of interest here is SGML's ability to formally define a document grammar. For clarity, additional features of SGML like minimization, marked sections and external references are not regarded.

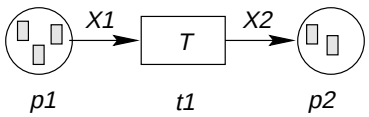


Figure 6. SGML net elements

The inscription $X1$ of the *input* arc defines which document instances (or parts of them) in the marking of place $p1$ may be *extracted* (*read*) when transition $t1$ occurs. Likewise, the inscription $X2$ of the *output* arc specifies an *insert* (*write*) operation, i.e. the creation of a new document instance or the modification of an existing one in $p2$.

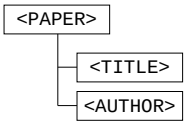
The following subsections give an informal introduction of the syntax and semantics of document templates.

4.2.1 Document templates

This section introduces the basic elements of document templates as a graphical query language for structured documents. Later on, these concepts will be picked up and specialized for use as arc inscriptions in SGML nets. The following examples are based on the `paper` DTD (see Fig. 1), if not stated otherwise.

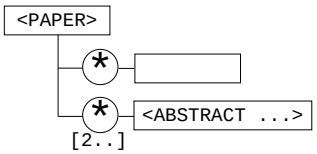
The *element match* operator, a box, matches an SGML element. It may be inscribed with the element's name. If left blank, it matches any element. To match a document instance beginning with a `paper` element, the following template suffices:

In our example DTD, a `paper` element's content consists of further elements. In document templates, a subordinate expression stems from the middle of the containing element, while elements on the same hierarchic level are connected by a vertical line. The following template matches any document starting with a `paper` element whose content begins with a `title`, directly followed (on the same hierarchy level) by an `author`.

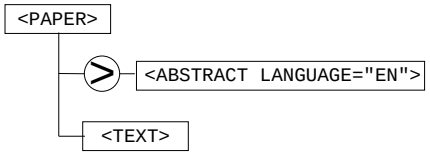


The *sequence operator* "`*`" matches a maximal sequence of elements on the current level of nesting hierarchy. If the DTD defines attributes for an element, these may also be used in a document template to restrict the set of eligible instances. The dots "..." in the `abstract` box act as a wild card to match an arbitrary number of additional attributes (which, of course, still have to conform with the DTD).

An optional lower bound a and/or an upper bound b for the length of the sequence may be added in square brackets $[a..b]$. If left out, the default values for a and b are 0 and ∞, respectively. Consequently, this expression matches any `paper` that contains at least two `abstract`s after an arbitrary number of other elements:

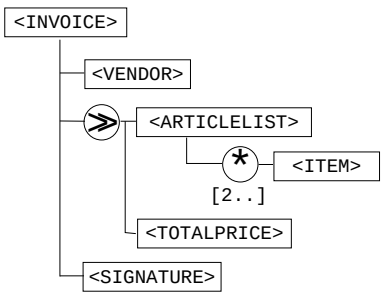


The *flat search* operator "$>$" searches the following elements of the *same nesting level* for a specified pattern. The expression shown in the following figure tries to find a `paper` containing an English `abstract` which is directly followed by a `text`.

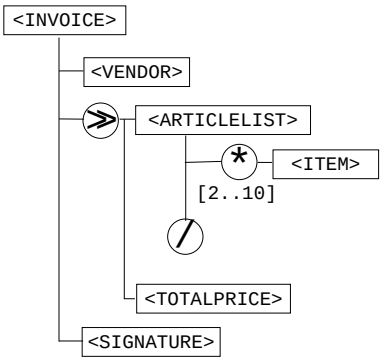


The *deep search* operator "$\gg$" allows to widen the search range of a subordinate expression to deeper nesting levels. The document tree is traversed in "depth first" order. Again, an interval of the form $[a..b]$ may optionally be added to consider only subtrees which are at a nesting level of at least a and at most b deeper, counted relative to the level of the last element matched before in the expression. This example finds all documents (i.e. not necessarily `paper`s) which *somewhere* contain a French `abstract`:

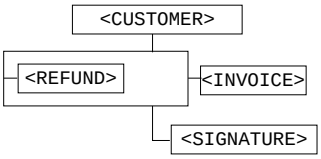
Another, more complex example query is not based on the `paper` DTD. It looks for documents starting with an `invoice` element which itself is structured as follows: the first (sub-) element must be a `vendor`, followed by a `signature`. Somewhere between `vendor` and `signature` there must occur a substructure with an `articlelist` having at least 2 subordinate `items` and, directly after the `articlelist` and on the same hierarchy level, a `totalprice`.



The *anchor* operator "/" matches the end of a hierarchy level. If we want to restrict the number of `item`s in the previous example additionally to at most 10, the resulting expression might look like this:

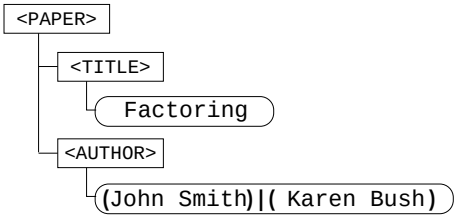


The following example shows how to specify (mutual exclusive) alternatives. Here, a document instance beginning with a `customer` element is matched if it contains *either* a `refund` *or* an `invoice` element, followed by a `signature`:



So far, we are only able to construct queries for document instances with a certain *structure*. For example, we are able to find documents with both a `title` and an English `abstract`, but we cannot restrict the "free text" *content* of an element[2]. For this purpose the *text match* operator (symbolized by an oval box) may be used to match the content of an element. This content is considered an unstructured character string. The box is inscribed with a regular (text) expression, with the familiar metacharacters (e.g. [1]) "**|**" (alternative), "*****" (iteration), "**( )**" (grouping) and "**?**" (match any character). For clarity, we use a bold font for metacharacters to distinguish them from their literal use in text.

This allows us for example to search for `paper`s with an occurrence of the word "Factoring" in the title, written by "John Smith" or "Karen Bush":

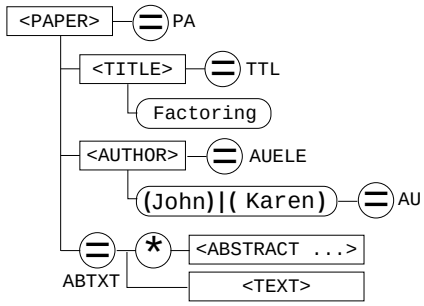


[2]In our example, the parts of text of the document instance corresponding to #PCDATA in the element's definition.

4.2.2 Read operations

In SGML nets, the inscription of an arc leading from a place to a transition ("input arc") specifies the subset of document instances in the place which might contribute to enable the transition and consequently is read when the transition occurs. For this use, some modifications and extensions to the concept of document templates as introduced in the previous section are necessary.

Obviously, we need a possibility to "assign" an element to a *variable*. Such variables can be used in the inscription of the adjacent transition (to specify further conditions under which the transition can be enabled) and its outgoing arcs (to create new documents). To achieve this, there is a *variable unification operator* "=", annotated with a variable name, which either has a dependent subexpression or (for ease of notation) may be connected to an "element match" or "text match" operator:



In this example, the entire `paper` can be later accessed through the variable PA, the `title` element through TTL, the `author` (which is required to *contain* the substring "`John`" or "`Karen`") via AUELE and the `author` element's *content* using the variable AU. Finally, the variable ABTXT holds all abstracts *and* the following `text`.

The contents of a variable is always considered plain *text* (possibly containing SGML markup). This explains the difference in contents of AUELE and AU: both hold copies of the content of the `author` element (#PCDATA, see the `paper` DTD in Fig. 1), but AUELE additionally includes the SGML begin and end tags of the `author` element. Likewise, the variable PA holds a copy of the *entire* matched `paper` document instance.

The scope of a variable is always restricted to the inscription of a transition and its incoming and outgoing arcs. The example in Fig. 7 illustrates this: The use of the variable TTL in both arc inscriptions expresses that transition notify_pc_chair can only occur if there is a pair of `paper`s in the two input places with the same title[3].

As already generally stated in the introductory section on Petri nets, selected tokens are removed from the input places

[3]This process of binding variables by *unification* is known e.g. from the area of logic programming and will be explained in more detail later on.

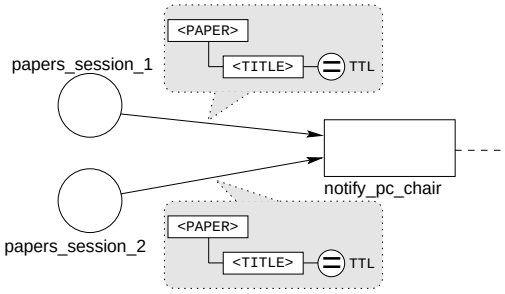


Figure 7. variable scope

when a transition occurs. Since we are able to access the internal structure of the "tokens" (i.e. document instances) in SGML nets, we do not necessarily want to remove a *whole* document instance from an input place. In order to support parallel access to different parts of an instance, it should be possible to remove only selected elements instead, allowing concurrent access to disjoint sections of a document instance by different transitions. Consequently, there needs to be a way to specify in the arc inscription which of the matched elements in the selected document instance should be removed (consumed) when the transition occurs and which elements only help to identify a document instance and should generally be left untouched.

To deal with these problems, we first introduce the following notion: The set of elements in a document instance explicitly matched by one of the two box operators in the arc inscription is said to be *covered* by that inscription. As a consequence, matching an element of a higher hierarchy level implicitly also covers *all* subordinate elements.

When using document templates to inscribe input arcs of transitions, we obviously need a way to differentiate between elements that are just used to help to filter out relevant document instances and those which additionally mark elements to be removed when the transition occurs. To make this distinction clear in the graphical notation, we define that both the element match operator and the text match operator are to be drawn as before with a closed box if the matched element is to be removed when the transition occurs and with a dashed box if no removal of the matched element is desired. Additionally, we define that a document instance is *always removed* completely from an input place iff *all* of its elements are covered by the respective arc inscription and the inscription contains at least one closed box.

The following scenario demonstrates that the problems explained above can now be solved quite elegantly: We might want to make two lists from a set of papers, consisting of title, author, and English or French abstract, respectively. The SGML net fragment in Fig. 8 helps to extract the necessary elements from a set of `paper` document instances by language, as requested.

To become enabled, transition collect_english_abstracts requires that at least one paper in its input place all_papers has an English abstract. When the transition occurs, it removes only this `abstract` from the document instance as long as there is another abstract left. If the paper has only an English `abstract`, the entire document instance is discarded in this operation. Transition collect_french_abstracts acts analogously with French abstracts. To give an example, if all_papers is marked just with the document instance of Fig. 2, transition collect_english_abstracts can occur exactly once, extracting the English abstract and leaving the document instance with just the German one behind. After that, neither of the two transitions is enabled under the resulting marking of all_papers.

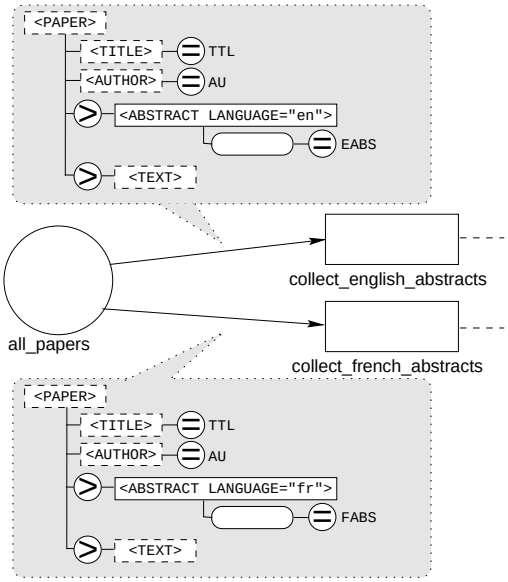


Figure 8. extracting abstracts by language

Note that the multiple occurrences of the variable names TTL and AU indeed denote different variables here since each is local to its respective transition.

4.2.3 Write operations

The inscription of an arc leading from a transition to a place ("output arc") specifies how the marking of the place is to be modified when the transition occurs. Here we need to differentiate between two complementary operations,

- the *creation* of a new document instance on the one hand and
- the *insertion* of new elements into an existing document instance on the other.

Again, the concept of document templates is used to specify where an insertion should take place or how a new document instance should be structured. An output arc is inscribed with *two* document templates, one defining the

create operation and another for the *insert* operation. In the graphical representation, the templates are marked by the keywords "CREATE:" or "INSERT:", respectively.

In order to define the operations, we must be able to identify the document instance in which to insert unambiguously. Therefore, each template in an arc inscription must contain at least one of the keys defined for the DTD of the adjacent place (see section 4.1).

Creation of new document instances. The creation of a new document instance is defined by a document template specifying the new instance's structure and contents. The following components may be used: As before, a closed box represents an SGML element while an oval box symbolizes an element's textual content. For obvious reasons, boxes must not be left blank in this context. Analogously to the variable unification operator in input arcs, the "=" operator is used here to interpolate the value of a variable (which implies that – in contrast to its previous use – it cannot have an additional subexpression on its right hand side). A variable has to be bound before in the inscription of the adjacent transition or one of its input arcs. The resulting document instance (after variable interpolation) must conform to the DTD of the adjacent place.

For an example, let us come back to the "English" branch of Fig. 8. We have bound variables to a paper's title (TTL), author (AU) and abstract (EABS) and now want to add these data to a list of English abstracts. We introduce a place abstract_lists which holds `list`s of `item`s containing `title`, `author`, and a `summary` of a paper, one list per `language`. Clearly, `language` identifies a document instance and is therefore declared as a key in the place's type definition (Fig. 9). Fig. 10 shows a valid document instance of `abstractlist`.

```
<!DOCTYPE abstractlist [
<!ELEMENT abstractlist    - -
       (language,list) >
<!ELEMENT language - - (#PCDATA) >
<!ELEMENT list     - - (item)+ >
<!ELEMENT item     - -
         (title,author,summary) >
<!ELEMENT title    - - (#PCDATA) >
<!ELEMENT author   - - (#PCDATA) >
<!ELEMENT summary  - - (#PCDATA) >
<!KEY language>
]>
```

Figure 9. abstractlist DTD

The arc inscription in the Fig. 11 states that if there is no document instance with matching key value(s) in place abstract_lists, a new instance with this structure will be created.

Note the difference in the way the TTL and AU on the one

```
<ABSTRACTLIST>
 <LANGUAGE>English</LANGUAGE>
 <LIST>
  <ITEM>
   <TITLE>Factoring Algorithm</TITLE>
   <AUTHOR>John Smith</AUTHOR>
   <SUMMARY>
     This algorithm factors a prime
     number in constant time.
   </SUMMARY>
  </ITEM>
  <ITEM>
   <TITLE>P = NP</TITLE>
   <AUTHOR>Karen Bush</AUTHOR>
   <SUMMARY>
     In this paper we prove for a
     special case (N=1) that P = NP.
   </SUMMARY>
  </ITEM>
 </LIST>
</ABSTRACTLIST>
```

Figure 10. a document instance

hand and EABS on the other are interpolated. In the inscription of the input arc of collect_english_abstracts, the TTL and AU were directly bound to *element* match operators. Consequently, they contain the necessary begin and end tags of their respective elements. Since the same element names were declared in the DTD of abstract_lists, their values can be inserted directly without type violation. In contrast to that, the `abstract` element in the `paper` DTD unfortunately corresponds to the one named `summary` in `abstractlist`. For this reason, the EABS variable was bound to the *content* of an `abstract` element, so we can insert a "constant" `summary` element and simply include EABS's contents there, as shown in Fig. 11.

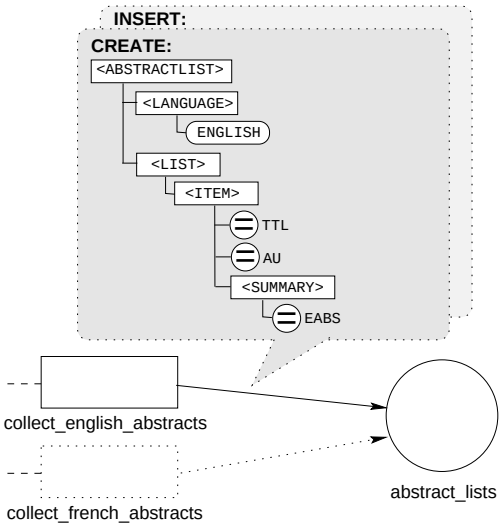


Figure 11. creation of a new abstractlist

Insertion into existing document instances. In analogy to the inscription of input arcs (see section 4.2.2), dashed boxes in the template help to "navigate" to the desired insertion points by matching corresponding elements in a document instance, while closed boxes symbolize insertion of new elements and therefore must not be left blank. It is possible to insert constant text as well as the contents of a previously bound variable.

Let us consider the following continuation of the "English" branch of our abstract sorting example. Assuming that there already exists a document instance with key value "ENGLISH" in place abstract_lists, the following arc inscription (Fig. 12) specifies the insertion of a new `item` at the top of the `list` element in such an `abstractlist` and thus completes the example of Fig. 8. Note that in contrast to the CREATE template of Fig. 11 we now have dashed boxes to *locate* the insertion point and closed boxes to actually *insert* the new `item`.

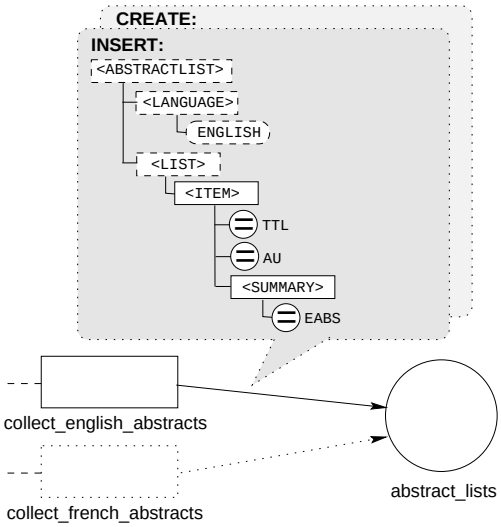


Figure 12. insertion at the top of list

A slight modification allows to append new entries at the end of the `list` (Fig. 13).

4.3. Transition Inscription and Occurrence Rule

Transitions can be inscribed with logical expressions to further specify under which conditions the transition becomes enabled. The variables used in the inscription of an incoming or outgoing arc of the transition may also be referenced in such an expression. Function symbols can be used for calculations and string manipulation as well as to model external influences, e.g. decisions of a human workflow participant. Figure 14 outlines another modification of the running example. Here, inscribed transitions are used to separate papers having abstracts with less than 2000 characters from longer ones.

The occurrence rule specifies the preconditions that have

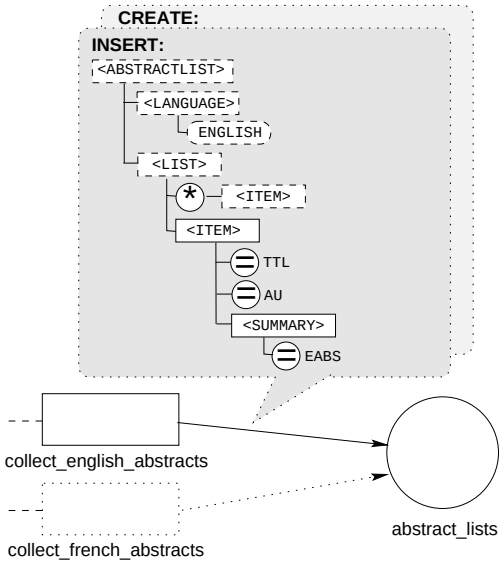


Figure 13. insertion at the end of list

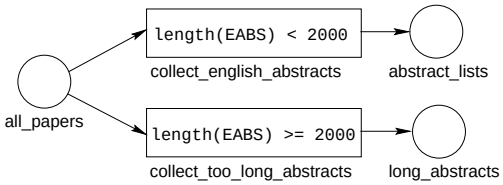


Figure 14. inscribed transitions

to be met for a transition to be enabled (so it *can* occur) and what actions are performed when the transition occurs. It defines the dynamic aspects of the Petri net model.

A transition t in an SGML net is *enabled* for a given marking and an instantiation of the variables in t's and its adjacent arcs' inscriptions if:

- every input place $p \in \bullet t$ contains a document instance which can be matched with the document template in the inscription of the arc leading from p to t *and* if extracting elements as specified by the arc's inscription will not cause a violation of p's DTD and if
- the transition inscription evaluates to "true" and if
- for each outgoing arc, the document instance creation/insertion operation as defined by the arc inscription will not lead to a violation of the adjacent place's DTD.

Only if all these conditions are met, the transition can occur and the manipulation of the document instances in the places in its vicinity is performed as specified by the transition and arc inscriptions. Note that starting from a valid initial marking of the net, the definition of the occurrence rule *guarantees* that no invalid document instance can possibly be generated in any following state. In every reachable system state, all document instances always conform to their respective places's DTD.

5. Extensions

This section outlines some extensions to the basic concept of SGML nets and introduces briefly some ideas how the integration of structured documents and a formal workflow model may act as a starting point for including other interesting concepts in business process modeling.

5.1. Access control

Since the logical structure of documents is made explicit in SGML nets, we can not only model the flow of complete document instances but also accesses to the document's constituent parts concisely. This opens the way for formally modeling and analyzing the flow of these individual elements with very high granularity. For example, it is easy to ensure that a certain workflow participant (modeled by a transition) will only see "non-confidential" elements of a document, simply by inscribing the input edges of the transition appropriately. This might be extended to a comprehensive model for formally specifying and validating security constraints in business processes based on structured documents.

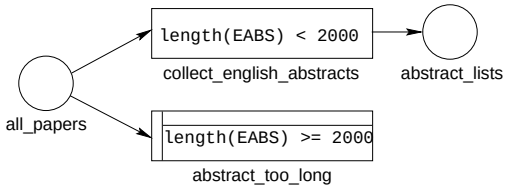
5.2. Active Documents

Traditionally, there is a separation between (passive) documents and (active) application programs for document manipulation. An interesting alternative approach is to view documents as user interfaces [4] or (conversely) viewing user interfaces as documents [9]. So-called active documents as presented in [20, 8] associate executable code with documents, for example allowing an specific rendering of certain parts of the document when it is displayed. This idea has been picked up recently with great interest by the Internet community, as the growing popularity of the Java programming language and its use for embedding "applets" into World-Wide-Web documents show.

SGML nets seem to be well prepared to take advantage of such an integration of document data and supplementary document-specific code, for example by declaring `<code>` elements containing executable program code in some platform-independent programming language. This code could then be referred to in the inscription of a transition.

5.3. Modeling Integrity Constraints

Fact transitions [10] are special transitions which by definition must *never* be enabled. This is an elegant and powerful concept to introduce a *declarative* specification of integrity constraints into Petri nets. Fact transitions are usually depicted by a transition symbol, inscribed with a stylized letter "F". As an example, the following modification of Fig. 14 uses a fact transition abstract_too_long to declare that the marking of all_papers must never contain a `paper` with an English abstract of 2000 characters or more:



Fact transitions are especially useful for formally validating a modeled system: an analysis may show whether there is a reachable marking of the net where a fact transition is enabled. If this is the case, the system may run into a state where an integrity constraint is violated, and the model thus should be reconsidered.

6. Related Work

There have been some efforts in the area of query languages for structured documents. Surveys of retrieval languages which consider content as well as structure are given in [3] and [15]. SCRIMSHAW [2] is implemented as a tool similar to the UNIX `grep` command. It uses regular expressions to filter out and re-write structured documents. [14] presents a retrieval language based on the Prolog logic programming language which takes advantage of the tree structure of hierarchical texts. [16] describes a query language for SGML documents which is influenced by the relational query language SQL. In contrast to the graphical approach described in this paper, these languages expect the query specification in a textual form.

Many current workflow systems use a texual programming language for defining business processes, but offer little support for high-level process modeling. Some use data flow diagrams to provide an intuitive but non-formal notation. An overview over workflow management methods and a comparison of some software products can be found in [11]. [21] discusses the use of high-level Petri nets for workflow modeling. Nested relation/transition nets [17] are a Petri net variant capable to express operations on complex structured objects on the basis of non-normalized (NF^2) relations.

7. Conclusion and outlook

SGML nets allow to exploit a document's structure in the specification of a business process in an elegant way. Parallel exclusive access of multiple workflow participants to different sections of the same document is an example for a phenomenon which is common in reality yet hard to represent in most workflow modeling formalisms which do not capture the document's structure. SGML nets can express

this quite naturally. Due to their formal semantics, SGML nets can be directly executed by a net interpreter and thus may serve as an executable specification. Extensions which are currently investigated cover areas such as active documents (by allowing SGML elements to contain executable code in an interpreted programming language, which can be accessed by a transition when it occurs) and taking advantage of the fact that the integrated modeling of process and document structure allows for a much more detailed specification and analysis of security and access regulations.

SGML nets have been developed as a part of the INCOME/WF project [19]. In the course of this project and its predecessor, INCOME/STAR, methods and tools for process modeling with Petri nets have been developed, among them a Petri net simulation system [18] and a distributed Petri net editor, based on distributed object technology. Currently, work is underway to add support for SGML nets to these systems.

References

[1] A. V. Aho. *Handbook of Theoretical Computer Science*, volume A, chapter Algorithms for Finding Patterns in Strings, pages 257–300. Elsevier, 2 edition, 1992.

[2] D. S. Arnon. Scrimshaw: A language for document queries and transformations. *Electronic Publishing*, 6(4):385–396, December 1993.

[3] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.

[4] E. A. Bier and A. Goodisman. Documents as user interfaces. In R. Furuta, editor, *EP90 – Proc. of the International Conference on Electronic Publishing, Document Manipulation & Typography, Gaithersburg, Maryland*, The Cambridge Series on Electronic Publishing, pages 249–262. Cambridge University Press, September 1990.

[5] U. Bormann and C. Bormann. Standards for open document processing: current state and future developments. *Computer Networks and ISDN Systems*, 21:149–163, 1991.

[6] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri nets, central models and their properties: advances in Petri nets 1986, part I: Proc. of an advanced course Bad Honnef, 8.-19. September 1986*, volume 254 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987.

[7] H. Brown. Standards for structured documents. *The Computer Journal*, 32(6):505–514, 1989.

[8] P. M. English, E. S. Jacobson, R. A. Morris, K. B. Mundy, S. D. Pelletier, T. A. Polucci, and H. D. Scarbro. An extensible, object-oriented system for active documents. In R. Furuta, editor, *EP90 – Proc. of the International Conference on Electronic Publishing, Document Manipulation & Typography, Gaithersburg, Maryland*, The Cambridge Series on Electronic Publishing, pages 263–276. Cambridge University Press, September 1990.

[9] M. Fuchs. The user interface as document: SGML and distributed applications. *Computer Standards & Interfaces*, 18:79–92, 1996.

[10] H. J. Genrich and K. Lautenbach. Facts in place/transition-nets. In J. Winkowski, editor, *Proc. of the 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *LNCS*, pages 213–231, Zakopane, Poland, Sept. 1978. Springer-Verlag.

[11] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, Apr. 1995.

[12] ISO 8613. Information processing – text and office systems – office document architecture (ODA) and interchange format, 1989. International Organization for Standardization.

[13] ISO 8879. Information processing – text and office systems – standard generalized markup language (SGML), October 1986. International Organization for Standardization.

[14] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. ACM SIGIR '93, Pittsburgh, PA, USA*, pages 214–222, 1993.

[15] A. Loeffen. Text databases: A survey of text models and systems. *SIGMOD RECORD*, 23(1):97–106, 1994.

[16] I. A. Macleod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[17] A. Oberweis, P. Sander, and W. Stucky. Petri net based modelling of procedures in complex object database applications. In *Proc. IEEE 17th Annual International Computer Software & Applications Conference (COMPSAC), Phoenix/Arizona, Nov 1–5, 1993*, pages 138–144. IEEE Computer Society, 1993.

[18] A. Oberweis, V. Sänger, and W. Weitz. GAPS – a multiuser tools for graphical simulation of Petri nets. In J. Halin, W. Karplus, and R. Rimane, editors, *CISS – First Joint Conference of International Simulation Societies Proc., Zurich, Switzerland, August 22–25, 1994*, pages 377–381. The Society for Computer Simulation, San Diego, CA, USA, August 1994.

[19] A. Oberweis, R. Schätzle, W. Stucky, W. Weitz, and G. Zimmermann. INCOME/WF – a Petri net based approach to workflow management. In H. Krallmann, editor, *Wirtschaftsinformatik '97, Berlin, Germany, February 26–28, 1997*, pages 557–580, Heidelberg, Germany, 1997. Physica-Verlag.

[20] V. Quint and I. Vatton. Making structured documents active. *Electronic Publishing*, 7(2):55–74, June 1994.

[21] W. van der Aalst, K. van Hee, and G. Houben. Modelling workflow management systems with high-level petri nets. In G. D. Michelis, C. Ellis, and G. Memmi, editors, *Proc. of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages pages 31–50, 1994.

[22] W. M. P. van der Aalst. Three good reasons for using a petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proc. of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, November 1996.

[23] M. Zloof. Query by example. In *Proc. of the National Computer Conference*, volume 44. AFIPS, 1975.