Analytical Version Control Management in a Hypertext System

Antonina Dattolo
Dipartimento di Informatica
University of Salerno
84081 Baronissi, Italy
email antos@udsab.dia.unisa.it

Antonio Gisolfi
Dipartimento di Informatica
University of Salerno
84081 Baronissi, Italy
email gisolfi@udsab.dia.unisa.it

Abstract

In this paper it is shown how structural and cognitive versioning issues can be efficiently managed in a Petri nets based hypertextual model. The advantages of this formalism [st:89] are enhanced by a modular and structured modeling; modularity allows to focus the attention only on some modules, while giving the abstraction of the others. Each module owns metaknowledge that is useful in defining new layers and contexts.

The central point of the data model is the formulation and resolution of three recurrence equations, effective in describing both the versioning and the derivation history; these equations permit to express in precise terms both the structural evolution (changes operated on specific nodes of the net) and the behavioral one (changes concerning browsing).

1 Introduction

Abstraction mechanisms in an hypertextual environment first have been investigated by Garg [ga:88] and their properties are described in several papers, most of them concerning information retrieval techniques [ag:93]. Abstraction mechanisms characterize the dynamic point of view of an hypertext and its most relevant features; one can recognize two types of dynamicity: the first having behavioral nature and thus linked to browsing and search (queries, dynamic linking, filtering, and so on), and the other of structural type, related to the opportunity that the user plays the role of author intervening personally on the system.

This paper is mainly concerned with this second type of dynamicity, and structural and cognitive issues related to the abstraction mechanism of versioning are scrutinized.

The versioning is considered as one of the relevant issues in modeling and building of hypertextual systems, and its importance has been emphasized by Halasz [ha:88, ha:91] as regards its relationships with software engineering [go:87] and design of databases [ka:90]. Versioning is introduced to preserve former states and save them from destruction; former states may be saved for later reuse of material, but also to preserve the historical perspective of work done [ha:92].

In particular, in distributed and collaborative hypertext systems, version control mechanisms are no longer an optional and become absolutely necessary to maintain the consistency of the database [ma:93] and provide effective solutions to multiuser concurrency control [wi:93].

Of course, version support will only be accepted by the users if the effort spent on version management is out-weighed by the benefits. Conklin [co:87] emphasizes that, when changes are carried out, assigning names to the new nodes or labeling links requires too additional effort to the user. In order to maximize benefits, the versioning must be carried out automatically, according to a session-based schema, instead of the user-decided approach that is required to let the user free to adopt the choices he prefers [ha:92, ha:93, ma:93]. Of course, if session-based versioning is adopted, the user, who can focus his attention on his normal activity while changes take place, must know the versioning strategies used by the system in order to collect the information he needs.

Only few hypertextual systems provide versioning of the structure [ak:88, go:87, ha:92, ha:93, os:92, ut:89] and a little more allow node versioning. For example, Trellis [st:89] does not permit modifying nodes or links and only versioning of node is allowed, managed by means of a logical function that associates structure and contents; the map defined by this function can remain unmodified for the common parts of the versions and be different as regards the modified contents. Thus Trellis does not have a specific structure to store the versions, and it is supposed that the latter do not modify the structuring net. However, Petri nets offer powerful tools for the analysis and representation and permit their description in precise terms. This paper aims at showing how Petri nets can be used in dynamic terms so that one can automatically reconstruct, by means of layers, any intermediate state of the net. Section 2 outlines some of the basic concepts concerning Petri nets such as the incidence matrix. Section 3 illustrates the data model, showing how the characteristics of modularity allow to manage every operation concerning the hypertext as an operation on a single module, and in such way the incidence matrices can explicate their full expressive power. The next section represents the core of the paper: first, the elementary operations that can be carried out are described, then it is shown how these operations can be managed in terms of variations of the incidence matrix and of the state equation. Moreover, the recurrence equations involved by the versioning are formulated and solved. This section includes the definitions of layer and context. Section 5 is devoted to node versioning: interface issues are discussed and it is shown how a browser can be visualized. In particular, the interpretation of the node versioning in terms of net versioning, i.e. recurrence equations, is investigated. A suitable example is discussed. Finally, the results of this paper are compared with other ones and some open problems are briefly illustrated.

2 Petri nets, incidence matrix and state equation

This section presents the basic concepts concerning Petri nets, incidence matrix and state equation. These topics are discussed in depth in [mu:89, pe:81].

2.1 Petri Net Definition

A Petri net is a particular kind of directed graph, together with an initial state called the initial marking, M. The underlying graph N of a Petri net is a directed, weighted, bipartite graph. A marked Petri net is a 5-tuple (N, M) = {P, T, L, W, M} where:

$P = \{p_1, p_2, ..., p_m\}$ is a finite set of *places*,
$T = \{t_1, t_2, ..., t_n\}$ is a finite set of *transitions*,
$L \subseteq (P * T)U(T * P)$ is a set of *arcs*,
$W : L --- > \{1, 2, 3, ...\}$ is a *weight* function,
$M : P --- > \{0, 1, 2, ...\}$ is the *initial marking*.

M is an m-vector, if m is the total number of places. The *p-th* component of M, denoted by *M(p)*, is the number of tokens in place p. A Petri net is said to be *ordinary* if all of its arc weights are 1's.

2.2 Incidence Matrix

The incidence matrix is a tool for the analysis of Petri nets that can be used only for restricted classes [mu:89] when one is facing problems concerning the possibility of solving the state equation. In the specific case, it is used, along with the state equation, as a tool more descriptive than analytical and thus it suits most cases well. For a marked Petri net (N, M) with n transitions and m places, the incidence matrix $A = [a_{ij}]$ is an $n * m$ matrix of integers and its typical entry is given by

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

where $a_{ij}^{+} = w(i, j)$ is the weight of the arc from transition i to its output place j and $a_{ij}^{-} = w(j, i)$ is the weigth of the arc to transition i from its input place j. It is easy to see that a_{ij}^{-}, a_{ij}^{+}, and a_{ij} represent the number of tokens removed, added, and changed, respectively, in place j when transition i fires once. Transition i is enabled at a marking M iff

$$a_{ij}^{-} \leq M(j) j = 1, 2 ...., m.$$

As we assume that the net does not include self-loops, one has that either $a_{ij} = a_{ij}^{+}$ or $a_{ij} = a_{ij}^{-}$ namely a_{ij}^{+} and a_{ij}^{-} cannot be both different from zero.

2.3 State equation

In writing matrix equations, we write a marking M_k as an $m * 1$ column vector. The *j-th* entry of M_k denotes the number of tokens in the place j immediately after the *k-th* firing in some firing sequence. The *k-th* firing or *control vector* u_k is an $n * 1$ column vector of *n-1* 0's and one nonzero entry, a 1 in the *i-th* position indicating that transition i fires at the *k-th* firing. Beacause the *i-th* row of the incidence matrix A denotes the change of the marking as the result of firing transition i, we can write the following state equation for a Petri net, for $k = 1, 2, ...$:

$$M_k = M_{k-1} + A^T \cdot u_k \qquad (1)$$

Suppose that a destination marking emM_d is reachable from M_0 through a firing sequence $u_1, u_2, ..., u_d$. Writing the state equation 1 for $k = 1, 2, ..., d$ and summing them, we obtain

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k \qquad (2)$$

3 The data model

This section illustrates the hypertextual data model, based on Petri nets. This choice has taken in account the behavioral properties of these nets which, as is shown in [st:89, st:90], are more general than directed graphs, since the latter can be described as subclass of Petri nets ([pe:81], p.41). The design principle that has driven our work is widely accepted, and basically states that a system should be composed of a set of independent modules, where each module in the system hides the internal details of its processing activities and modules communicate through well-defined interfaces [fa:85]. The single module, a small Petri net, is defined *fortress* and is a finite structured set of information, which can be viewed as the cognitive unit, the smallest autonomous piece of information, like the Garzotto's entity [ga:93], that can be analyzed independently; for each unit suitable standards are defined and consequently it can be considered as a black box capable of producing and receiving information. The flow of information between the units takes place by means of a simple structure called *draw-bridge* (db, for short) that can be activated if a "toll" is paid, namely a fixed number of tokens that represents the user's know-how. In order that the browsing be effective, it is crucial that each fortress is capable of generating tokens so that dbs are kept ables to firing. A fortress is a structure that allows the user's browsing. It is a Petri net, where it is vital that the logical roles are clearly specified, as expressed by its basic units (i.e. place, transitions and tokens) that are inherited, with slight changes, from the Trellis model [st:89]. Such aspects are enclosed in the definition of a logical hypertext $I_L = \{R_k, R_m\}$, where R_k is a set of associative rules whose goal is to manage the knowledge and the way it should be visualized (see below the *map, locus* functions, and so on) and R_m is a set of rules for managing the marking (see, for example, the *history* function).

- The place is an elementary unit of information useful for both explicit (visible) on and implicit (control) information. The content of a place is visualized when it contains at least one token, by means of the *map function* (similar to the logical function $C_l(p)$ described in [st:89]) used in the following to define layers. A *locus function* is also associated with each place to denote the logical position on the screen of the contents of the place (similar to the function $W_l(p)$ described in [st:89]).
- The transition gets associated with an anchor on the screen (similar to the button $B_l(t)$ discussed in [st:89]) and thus with the possible path.
- In turn, the history of the navigation in the fortress, via the set of control vectors, gets associated with the

token by means of the *history function* (the effectiveness of this choice is discussed also in [de:93]).

From the global point of view, an hypertext is a couple $I = \{I_L, I_S\}$, where I_L represents the logical hypertext and I_S the structural one, respectively. The distinction of I in the couple I_L and I_S allows for the separation of structure from content emphasised in [st:89, ga:93], favouring the reusing of the same structure (or part of it) into different fields, offering the possibility of several presentations (for example, written in several languages). Thus, from the structural point of view an hypertext can be considered as a couple $I_S = \{F, DB\}$ where:

$F = \{F_1, F_2, ..., F_f, \textit{F-In}, \textit{F-Out}\}$ is a finite set of fortresses ($f \geq 0$), where a specific role is played by *F-In* and *F-Out*.

$DB = \{dB_1, dB_2, ..., dB_d\}$ is a finite set of db_s and $d \geq f$ (this condition stems from the requirement that no deadends are allowed). The following schema outlines the structure of the single fortress (Fig.1).

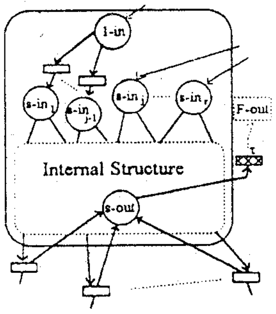

Fig.1 - The structure of a generic fortress

Each $F_i = \{(N, M), SI, \textit{s-out}\}$ where (N, M) is a colored marked Petri net; $SI = \{\ \textit{1-in}, s - in_1, ..., \textit{s-in}_r\}$ is a subset of P and is a set of control places, called *sentries-in* ($s - in_s$, for short), each containing either an index or a browser or a map of all reachable nodes starting from it. In particular, the *1-in* sentry gives the abstraction for the fortress (similarly to the place GSP in [de:93]). Only this sentry, when marked, allows reaching any place of the fortress; however, each of the other *s-in$_s$* has the structural property of permitting to get out from the fortress and thus the control place *s-out* is certainly reachable. It is worth noting that the modular architecture allows to manage the navigation just by maintaining the control of the hypertext via the management of a limited number of nodes inside each fortress. The *s-out* place ($\textit{s-out} \in P$) is a special control place devoted to managing the exit from the fortress: in fact every exit from the fortress, which can take place only via a db, induces, as is shown in fig.1, that a token, containing the history of the browsing in the fortress, reaches the *s-out* place and updates, via the transition t, the global browsing schema contained in the *F-Out* place. Two different roles are played by the sentries/fortresses IN and by the OUT ones: in fact, the first have to store detailed information about the contents of the net and in particular of the fortress, whereas the latter manage the markings and keep track of the user's browsing. For example, let's suppose we want to organize this paper (indicated by dg94) according to the fortress schema. One possible structure is shown in fig 2. As you can see, the *1-in* represents the abstraction of the fortress content; in a similar way, each *s-in* represents the abstration of its subtree. Certain sentries (Petri Net Definition, Incidence Matrix, State equation) if their contents can be used in other contexts. Some places (see in fig.2, Related Work) are linked to information relating to other fortresses (in our example, the node, Related Work, is connected to some papers (ga93, ma93, etc.) contained in the section references.

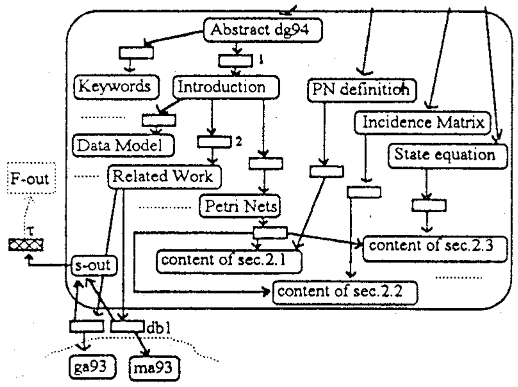

Fig.2 - An example of fortress organization

As regards browsing, if the user in the example were to follow the sequence of transitions 1-2-dbl, then the token, sent towards the $s - out$ when it crosses *db1*, would contain the (time-sorted) set of control vectors; such information would reach the *F-Out* with the aim of controlling the derivation history of user browsing. It follows that the link to ga93, in fig.2, leads to the crossing of db and, therefore, the access of the other cognitive unit. We can also note that this is not the only way to organize a fortress: it change according to the structural philosophy pursued by the author.

It can be easily shown that the structural hypertext is fully described by the set $A_1, A_2, ..., A_n, A_I, A_O$ of incidence matrices (related to the fortresses $F_1, F_2, ..., F_n$,*F-In*, *F-Out*, respectively), and by the associated marking vectors. On the other hand, the user's browsing requires the logical *map* function, and the equation (2); this equation, that includes the matrix A and the vectors M and u, fully describes the static structure of the net and its behavioral dynamics as expressed by the variation of the markings. We aim at investigating the properties of structural dynamics, thus formulate a more general equation of recurrence that expresses the structural dynamics of the net, showing in the meantime the correspondence that links each change carried out on a specific fortress to a set of changes on A, M and u. It is worth emphasizing that the fortresses are inherently modular; thus every operation will be carried out working on a specific fortress, as described by its incidence matrix, denoted by A and the marking vector, denoted by M.

4 Versioning of structure

In this section it is shown how the versioning of the structure can be managed by the system in active terms by applying the layer definition, considered as net operators devoted to its transformation both in logical and physical terms. After introducing the basic operations and the specific notation for vectors and matrices in the paragraphs 4.1 and 4.2, the paragraphs 4.3 and 4.4 discuss the formulation and the resolution of three recurrence equations corresponding to A, M and u. Then, in Section 4.5, the concepts of *layer* and *context* are illustrated.

4.1 Basic operations

The changes undergone by a net can be viewed as the result of some basic operations, such as add or delete a place, a transition, arc, or token. In order to get a precise mathematical definition, it is important to examine how each elementary operation modifies the net by changing the values of the matrix A and the vectors M and u. It is shown below how the single elements can be modified, in correspondence with each elementary operation. Let us briefly illustrate the elementary cases which give rise to other cases.

1. Adding/Deleting a place.
1.1 Adding/Deleting a column in the matrix A corresponding to the new/old place.
1.2 Adding/Delcting an element to the vector M corresponding to the new/old place.
1.3 Invariance on u.

2. Adding/Deleting a transition.
2.1 Adding/Deleting a row in the matrix A corresponding to the new/old transition.
2.2 Invariance on M.
2.3 Adding/Deleting an element to the vector u corresponding to the new/old transition.

3. Adding/Deleting a set of arcs related to a place.
3.1 Updating a column in the matrix A corresponding to the place.
3.2 Invariance on M.
3.3 Invariance on u.

4. Adding/Deleting a set of arcs related to a transition.
4.1 Updating a row in the matrix A corresponding to the transition.
4.2 Invariance on M.
4.3 Invariance on u.

5. Adding/Deleting tokens in a set of places.
5.1 Invariance on A.
5.2 Updating the element in the vector M corresponding to the set of places.
5.3 Invariance on u. We are looking for a general relationship that describes the changes of the triple *(A, M, u)* that summarizes, together with the *map function*, univocally the hypertext. This means that "snapshots" of the net are to be taken allowing to choose the particular version to be operated upon [ha:88]. Of course, in order to carry out the changes while keeping track of them it is necessary that a set of relevant data is specified. For the sake of simplicity, such data can be represented by a vector, $a = (a_1, a_2, a_3, a_4, a_5)$, where each component is referred to the preceding five operations. We assume that all changes operate on a net represented by the $n * m$ matrix A.

a_1 is an index j whose meaning is: if $j=m+1$ a place is added in the net, if $1 \leq j \leq m$ the *j-th* place is deleted, if $j=0$ no operation is carried out.

a_2 is an index i whose meaning is: if $i=n+1$ a transition is added in the net, if $1 \leq i \leq n$ the *i-th* transition is deleted, if $i=0$ no operation is carried out.

a_3 is a bi-dimensional vector where the first component, $a_{3,1}$, is an index j, $1 \leq j \leq (m+1)$, and the second , $a_{3\,2}$, is a column vector containing the changes of the arcs corresponding to the *j-th* place.

a_4 is a bi-dimensional vector where the first component, $a_{4\,1}$, is an index i, $1 \leq i \leq (n+1)$, and the second, $a_{4\,2}$, is a row vector containing the changes of the arcs corresponding to the *i-th* transition.

a_5 is a row vector of length m and contains the changes of tokens corresponding to the places.

4.2 Notation

In order to formulate adequately a general equation describing the hypertextual net, we have to define operators capable of expanding/compressing/modifying the triple *(A, M, u)*. In particular, the matrix ϵ defined below is an operator used either to expand or compress rows and columns of the incidence matrix, while the matrix σ modifies the contents by means of direct addition. The above-defined vector a is the input vector for both operators.

Definition of ϵ (Expansion matrix).
The matrix $\epsilon_{n,i}$ is an operator capable of expanding or compressing the matrix A, i.e. adding and deleting places and transitions in the net. This operator is defined as follows:

$$\epsilon_{n,i} = \begin{cases} \begin{pmatrix} I_n & 0 \end{pmatrix} & \text{if } i=n+1 \\ i\begin{pmatrix} I_{i-1} & 0 \\ 0 & 0 \\ 0 & I_{n-i} \end{pmatrix} & \text{if } 1 \leq i \leq n \\ I_n & \text{if } i=0 \end{cases}$$

Fig.3 - The expansion matrix

where I_j denotes the identity matrix of dimensions $j * j$. The operator ϵ, if applied to the right hand side of the incidence matrix, enables, in the case of $1 \leq i \leq n$, the elimination of the *i-th* column of the matrix (that is to say the *i-th* place in the net) or the addition of a new place if $i = n + 1$. Dually, the argumentation is valid for the transitions; in this second case, the transpose operator is applied to the left hand side of the equation in a similar way to the above.

Definition of σ (Sum matrix).
The matrix $\sigma_{n\ m,j,m}$ is a matrix whose dimensions are $n * m$ and whose elements are zero except on the *j-th* $1 \leq j \leq m$ column, represented by the n-vector μ (Fig.4).

$$\begin{array}{cccccccc} & 1 & \cdots & j-1 & j & j+1 & \cdots & m \\ \sigma_{n \cdot m, j, \mu} = & (0 & \cdots & 0 & \mu & 0 & \cdots & 0) \end{array}$$

Fig.4 - The sum matrix

4.3 Formalizing the recurrence equation.

Let $(A_{k-1}, M_{k-1}, u_{k-1})$ be the triple that describes a generic fortress which has undergone *k-1* structural changes; suppose now that the *k-th* change should carried out and that the dimensions of the matrix, before the change, are $n * m$. Thus the vectors M_{k-1} and u_{k-1} have dimensions $1 * m$ and $n * 1$, respectively. Then let us consider how the primitive operations can lead to the new triple (A_k, M_k, u_k). We show how the changes on the triple $(A_{k-1}, M_{k-1}, u_{k-1})$ depend upon the values of a. For the sake of simplicity each row or column will be added at the rear of the others. Thus, the general equations can be formalized as follows:

1. Adding/Deleting a place.
1.1 $A_k = A_{k-1} \cdot \epsilon_{m,a_1}$
1.2 $M_k = M_{k-1} \cdot \epsilon_{m,a_1}$
1.3 $u_k = u_{k-1}$

2. Adding/Deleting a transition.
2.1 $A_k = \epsilon^T_{n,a_2} \cdot A_{k-1}$
2.2 $M_k = M_{k-1}$
2.3 $u_k = \epsilon^T_{n,a_2} \cdot u_{k-1}$

3. Adding/Deleting a set of arcs related to a place.
3.1 $A_k = A_{k-1} + \sigma_{n\ a_{3\ 1}, a_{3\ 1}, a_{3\ 2}}$
3.2 $M_k = M_{k-1}$
3.3 $u_k = u_{k-1}$

4. Adding/Deleting a set of arcs related to a transition.
4.1 $A_k = A_{k-1} + \sigma^T_{n \cdot a_{4.1}, a_{4\ 1}, a_{4.2}}$
4.2 $M_k = M_{k-1}$
4.3 $u_k = u_{k-1}$

5. Adding/Deleting tokens in a place.
5.1 $A_k = A_{k-1}$
5.2 $M_k = Mk - 1 + a_5$
5.3 $u_k = u_{k-1}$

These 15 equations can be reduced to just three, one for each element of the triple (A_k, M_k, u_k). The notation can be suitably modified in order to render the equations more readable. In fact, one can define:

$\epsilon_{m,a_1} = \pi$ to denote changes related to the places,
$\epsilon^T_{n,a_2} = \tau^T$ to denote changes related to the transitions,
$a_5 = \mu$ to denote changes related to the markings,
$\sigma_{n\ a_{3.1}, a_{3.1}, a_{3\ 2}} + \sigma^T_{m\ a_{4\ 1}, a_{4.1}, a_{4.2}} = \sigma$ to denote the information concerning the arcs given by places and transitions. As a consequence of the way σ has beed defined, both operands cannot be simultaneously different from zero.

Of course, each variable has an index related to the specific intermediate state. The matrix A and the vectors M and u represent the known quantities, i.e. the initial conditions for the resolution of the equations. Thus:

$$A_k = \tau_k^T \cdot A_{k-1} \cdot \pi_k + \sigma_k \qquad (3)$$
$$A_0 = A$$

$$M_k = M_{k-1} \cdot \pi_k + \sigma_k \qquad (4)$$
$$M_0 = M$$

$$u_k = \tau_k^T \cdot u_{k-1} \qquad (5)$$
$$u_0 = u$$

It is worth emphasizing that these equations include all the basic operations, as shown in Section 4.1. We note that, in case the operation regards a place, the operator τ_k^T that describes the variations on transition is the identity operator and thus does not give any contribution to the matrix A_k. The operator π_k behaves in a similar way.

4.4 Solving the recurrence equations

The equations 3 describe the generic step concerning the evolution of A. The initial stop condition coincides with the original matrix. This equation can be solved by carrying out the appropriate changes:

$$A_k = \tau_k^T \cdot A_{k-1} \cdot \pi_k + \sigma_k =$$
$$= \dots\dots\dots =$$
$$= \tau_k^T(\tau_{k-1}^T \dots (\tau_1^T \cdot A_= \cdot \pi_1)\pi_2 + \sigma_2)\dots)\pi_{k-1} + +\sigma_{k-1}\pi_k + \sigma_k$$

In such way one gets the generic matrix A_k as expressed in terms of the original matrix. This facts is similar to what happens in ANCESTOR [ga:88], where the relationships between versions concerning the same fragment are investigated. As the matrix product is associative, one gets:

$$\tau_k^T \cdot \tau_{k-1}^T \dots \tau_2^T \cdot \tau_1^T \cdot A \cdot \pi_1 \cdot \pi_2 \dots \pi_{k-1} \cdot \pi_k +$$
$$\tau_k^T \cdot \tau_{k-1}^T \dots \tau_2^T \cdot \sigma_1 \cdot \pi_2 \dots \pi_{k-1} \cdot \pi_k +$$
$$\tau_k^T \cdot \tau_{k-1}^T \dots \tau_3^T \cdot \sigma_2 \cdot \pi_3 \dots \pi_{k-1} \cdot \pi_k + \dots\dots + $$
$$\tau_k^T \cdot \sigma_{k-1} \cdot \pi_k + \sigma_k.$$

and, if $s_0 = A$, because the product of transpose matrices equals the transpose product in reverse order of these matrices, the final equation is:

$$A_k = \sum_{i=0}^{k-1} (\prod_{j=i+1}^{k} \tau_j)^T \cdot \sigma_i \cdot (\prod_{j=i+1}^{k} \pi_j) + \sigma_k \qquad (6)$$

This equation expresses in a compact way a set of possible changes in the initial net: more precisely, it consists of the sum of σ-type matrices (except that representing the fortress in its initial state), that give their contribution to the changes on the arcs. The latter take suitably in account the changes concerning the transitions and the places by means of the left and right product, respectively. The recursion step is given by the so called *k-th change*. More precisely, it is worth noting the in the equation 6 the matrices of type τ, π and σ do not contribute in a mutually exclusive way; thus the *k-th* step consists either of a single operation selected among the elementary ones (paragraphs 4.1, operations labelled with 1, 2, 3, 4, 5 with the only provision that within the same recursive step only one of the operations 3 and 4 can take place) or of the union of two, three or four different operations. Moreover, we note that reconstructing a configuration is connected to multiplying ϵ-type matrices with σ-type matrices; such product involves sparse matrices and consequently can be easily carried out. Finally, the equations 4 and 5 can be solved in a similar way, and the result is as follows, if $\mu_0 = M$ and $u_0 = u$:

$$M_k = \sum_{i=0}^{k-1} \mu_i \cdot (\prod_{j=i+1}^{k} \pi_j) + \mu_k \qquad (7)$$

$$u_k = (\prod_{j=0}^{k-1} \tau_j)^T \cdot u_0 \qquad (8)$$

4.5 Layers and Contexts.

The changes that can be carried out for the single fortress can be expressed by the equations 6, 7, 8 except the changes of contents of the specific nodes. In fact, it is important that for each node the value of the *map* function is taken in account, which is to be considered as one of the basic changes. Thus it is worth introducing the concept of *layer* as the set of elementary data that are required to descrive the evolution of the net through its intermediate states. In particular, the generic *k-th* layer expresses the changes related to a generic recursion step and is specified by:

- the dimensions of the matrix $A_{k-1} : n * m$;
- the vector a;
- the possible change undergone by the *map* function on the involved nodes;

- the set of possible layers to which the *k-th* one can be applied.

As a consequence of the definition, the layers get hierarchically organized: the root is at layer 0 and contains the fortress in its initial state. Building the hierarchy is necessary to determine priorities. Every path from the root leads to new states of the fortress, advancing by difference. Consequently, for each fortress, a *derivation tree* is obtained. It is worth emphasizing that, although a layer (obeying the equations 6, 7, 8 could include, in general, changes involving several elementary operations (and this can happen, as a consequence of an user's query, in user-decided), the layer originated by an automatic versioning, i.e. in session-based, contains just one change on the net: this is necessary for reconstructing the derivation step-by-step. The sequence of two or more layers represents a particular structure called *context*. A context can be visualized, starting from a certain state that is represented as an edge in the derivation tree, as a sub- edge of the latter. Our interpretations of layer and context originate from the definitions given by PIE [go:87]. Such definitions are considered a good starting point [ha:88], but now they are completely specified in a formal way. In fact, the rule of composition of layers into contexts is analytic. In the following section we shall show how a context can be considered, under particular conditions, as an object that applies, consecutively and without intermediate steps, all the layers it consists of: this basically happens when the set of changes are logically difficult to separate.

5 Versioning of node

The aim of this section is to give an example which shows how versioning works in a situation of node versioning. On this, we will focus our attention and we will stress how it can be considered as a particular case of versioning of the structure. From the structural point of view, the changes concerning the specific node are speficied by the equations 6, 7, 8 and by the derivation tree. The latter can be used to visualize a browser that allows the user to examine the sequence of changes undergone by the net. On the other hand, the user sometimes needs to examine the local evolution of the net: to this aim the versions related to a node are linked to the node itself and this goal is achieved by means of a structure allowing only local navigation thanks to a small colored Petri net. More precisely, in the example of fig.2, we can imagine modifying the contents of the place, Data model, from now on shown as p. Let v be p's version, then a small net can be added, as shown in Fig. 5, where the bold line implies that traversal is possible only for a gray token (fig. 5).

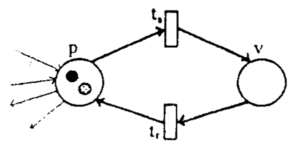

Fig.5 - Node with only one version

The situation becomes more complex if the modifications on the node are numerous and if we want to maintain their history. In this case, there would exist several versions of the node p $v_I, v_2, ..., v_v$ which would be able to be structured as shown in fig. 6, where p_I is an index place whose contents is a browser on the versions. Thus every generic version is "freezed" and considering another one as current adds the previous version to the set of versions as the r+1- *th* and the new version becomes current. We note that the net shown in Fig.6 gives rise to an incidence matrix for each node with version of the general form: in fact, if the number of versions is v, then the matrix corresponding to the node p has size $(3v + 1) * (v + 2)$ where the addenda +1 and +2 are given by the transition index t_I and index places p and current p_I, respectively.

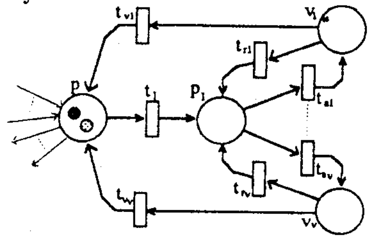

Fig.6 - Node with multiple versions

Thus, by adding a new version a new column and three rows will be added to the matrix and the positions of the elements +1, -1 and 0 will be standard, as shown in Fig.7.

$$\begin{array}{c|cccccc}
 & p & p_I & v_I & \cdots & v_v \\
\hline
t_I & -1 & 1 & 0 & \cdots & 0 \\
t_{aI} & 0 & -1 & 1 & \cdots & 0 \\
t_{rI} & 0 & 1 & -1 & \cdots & 0 \\
t_{vI} & 1 & 0 & -1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & & \vdots \\
t_{av} & 0 & -1 & 0 & \cdots & 1 \\
t_{rv} & 0 & 1 & 0 & \cdots & -1 \\
t_{vv} & 1 & 0 & 0 & \cdots & -1
\end{array}$$

Fig.7 - Incidence matrix corresponding to the net depicted in fig.6

For example, we show that adding a new version *(v+1-th)* can be managed by a derivation tree and then expressed by the equation 6 - the equations 7 and 8 are deducible from the 7. Then suppose that the initial matrix is that depicted in fig.7 and that a new version for the node p is gotten in order to modify its contents. If the value of the *map* function corresponding to p is $f_m(p)$ and the variable *new-content* contains the change carried out, then the derivation tree can be represented as follows:

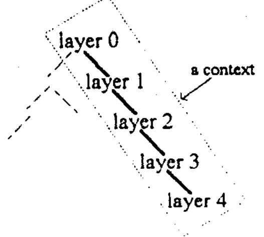

Fig.8 - The derivation tree

where:

$layer1 = [(3v+1)*(v+2)|a^1 = (v+3, 0, (0, \underline{0}), (0, \underline{0}), \underline{0})|$
$f_m(v_{v+1}) = fm(p); f_m(p) = new - content|layer0]$

$layer2 = [(3v+1)*(v+3)|$
$a^2 = (0, 3v{+}2, (0, \underline{0}), (3v{+}2, (0, -1, 0, ..., 0, 1)), \underline{0})||layer1]$

$layer3 = [(3v+2)*(v+3)|$
$a^3 = (0, 3v{+}3, (0, \underline{0}), 3v{+}3, (0, 1, 0, ..., 0, -1)), \underline{0})||layer2]$

$layer4 = [(3v+3) * (v+3)|$
$a^4 = (0, 3v+4, (0, \underline{0}), (3v+4, (1, 0, ..., 0, -1)), \underline{0}]||layer3]$

The equation 6 becomes simple because $\tau_1, \pi_2, \pi_3, \pi_4$ are identity matrices and σ_1 is a zero matrix:

$$A_4 = (\tau_2 \cdot \tau_3 \cdot \tau_4)^T \cdot A \cdot \pi_1 + (\tau_3 \cdot \tau_4)^T \cdot \sigma_2 + \tau_4^T \cdot \sigma_3 + \sigma_4$$

In particular, if we apply the substitutions specified in subsection 4.3, the following values are obtained,

for *ı=2, 3, 4*:
$\tau_1 = \epsilon_{v+2,v+3}$
$\tau = \epsilon_{3v+\imath-1,3v+\imath}$
$\sigma_\imath = \sigma_{(v+3)\ a^\imath_{4.1}, a^\imath_{4\ 1}, a^\imath_{4\ 2}}$

The previous example shows how it is possible to construct layers, modify the contents of the nodes in the new versioning, apply the recurrence equations, maintaining in the derivation tree the derivation history.

6 Related Work

First of all, it is worth sketching the data model. As said before, it originates from [st:89] but, although the basic structure are identical (Petri nets) it differs in several aspects. First, the knowledge is organized in structured and modular units, the fortresses (the relevance of this aspect stressed also in [ga:93, sh:93]). Second, our approach is characterized by the peculiar management of the meta-knowledge contained, for example, in the control place *s-out* and the *history* function [de:93] describing the behavioral evolution of the net. Finally, the paper emphasizes the role of dynamic structures and the version control mechanisms, such as the recurrence equations and the derivation tree. In particular, the concepts of layer and context, similar to those present in [go:87] and in [ha:91], are different as regards the rules for organizinig layers and obtaining contexts. In fact, the contexts can be viewed as a sequence of layers, but the word "sequence" implies that the net be transformed by the equations 6, 7, 8. A context arises by the simple composition of layers: thus it shows its double nature, i.e. the aspect present in PIE [go:87] where it is considered as a composition of layers and that illustrated in [os:92] as a sum of layers. In our approach both proposal coexist but with some differences: in fact, it differs from the first proposal because the derivation tree allows to avoid possible inconsistent (as noted also in [os:92]) combinations of layers. Our approach is different also as regards the second proposal: in fact, a context can be explicited as a sum of its parts (the layers) and thus it allows to reconstruct any intermediate state obtaining the whole derivation history. This last feature is important: few systems [de:87] permit to build step-by-step the system evolution by means of the derivation tree. It is similar, to a certain extent, to the "revision tree" [ti:85] because both operate by difference. However, while the *revision* tree is simply explicited only for revisions involving one hypertextual object, the *derivation* tree contains the history of the whole hypertext by means of the fortresses the tree consists of. Moreover, differently by [os:92] and in a way similar to [ha:91], the versioning can be applied to each part of the net (places, transitions and arcs). Also, it is worth emphasizing that our approach stresses, as in [ha:92] and [ma:93], the importance of the automatic management of the versioning Finally, we want to emphasise that the management philosophy of versioning mechanisms is not necessarily tied to Petri nets, but the latter prove an effective instrument of organization and control; they represent a wider class in comparison with directed graphs, generally used in hypertext models.

7 Concluding remarks and outlook

Basically this paper has dealt with an analytical approach to the version control management. In fact, the single layer, viewed as a set of changes that can be carried out on the generic fortress, leads to apply these changes considered as algebraic operators. The structure described in this paper allows dynamic behavior, i.e. the management of the user's derivation history, within a context, via the set of *control* vectors deriving from the sentries-out and thus the state equation 2. Moreover, dynamic management is allowed thanks to the equations 6, 7, 8 and the *derivation* tree that is capable of managing partial changes both concerning the contents (by means of the *map* function) and the structure of the net.

The modularity allows to decentralize the controls via the single fortresses, and moreover the exponential complexity involved in connected graphs decreases. In particular, studying the equation 6, 7, 8 leads to formulate two remarks: first, we note that the equations 7 and 8 can be gotten by solving the equation 6; secondly the 6 is formulate in a simple way and requires, thanks to the definitions given for ϵ-type and σ-type matrices, limited space for its storage and short times for its solution, because we are dealing with sparse matrices. As a consequence, the derivation history can be entirely described. Our approach is just a starting point, we think that guidelines for further work are two-fold: the first is related to the data model and authoring problems, the second leads to investigate versioning issues in a collaborative and distributed environment. As regards the first point, the net underlying the hypertext should be fully scrutinized. In fact, the author and the user are greatly interested in compacting the information present in the net in order to improve the facilities for changes and navigation, respectively. We think that using suitable metrics [bo:92] that adequately express the relevant features of the graph underlying the hypertext, and allow to tackle problems such as the reachability of a node or its centrality. The second point aims at defining a collaborative and distributed environment [da:94], where the version control mechanisms become necessary and are part of the set of basic requirements to be specified [ha:93, wi:93].

References

[ag:93] M. Agosti. Hypertext and Information Retrieval. *Information Processing & Management, Special Issue: Hypertext and Information Retrieval*, 29(3):283-285, May-June, 1993.

[ak:88] R. M. Akscyn, D. L. McCracken, E. A. Yolder. KMS: A Distribuited Hypermedia System for Managing Knowledge in Organitation. *Communications of the ACM*, 31(7):820-835, July 1988.

[bo:92] R. A. Botafogo, E. Rivlin, B. Shneiderman. Structural Analysis of Hypertext· Identifying Hierarchies and Useful Metrics. *ACM Transactions on Information Systems*, .10(2):142-180, April 1992.

[co:87] J. Conklin. Hypertext: An introdution andsurvey. *IEEE Computer*, 20(9):17-41, September 1987.

[da:94] A. Dattolo, V. Loia. Hypermedia Design Issues in an Actor-based Framework. *Technical Report USDIA-94-2*, Marzo 1994.

[de:87] N. Delisle, M. Schwartz. Contexts - A Partioning Concept for Hypertext. *ACM Transactions on Office Information Systems*, 5(2):168-186, April 1987.

[de:93] Y. Deng, S. K. Chang, C. A. de Figueired, A. Perkusich. Integrating Software Engineering Methods and Petri Nets for the Specification and Prototyping of Complex Information Systems. *Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science (691), Proc. 14th International Conference* Chicago - Illinois, USA, pp. 206-223, Marco Ajmone Marsan (Ed.), Springer- Verlag, June 1993.

[fa:85] R. Fairley. *Software Engineering Concepts.* MacGraw-Hill, New York, NJ, 1985.

[ga:88] P. K. Garg. Abstraction mechanisms in Hypertext. *Communications of the ACM*, 31(7):862-870, July 1988.

[ga:93] F. Garzotto, P. Paolini, B. Schawbe. HDM - A Model Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1): 1-26, January 1993.

[gi:94] A. Gisolfi, V. Loia. Designing Complex Systems within Distributed Architectures. to appear in *International Journal od Applied Artificial Intelligence* 8(3), 1994.

[go:87] I. Goldstein, D. Bobrow. A layered approach to software design. *Interactive Programming Environments*, Barstow D., Shrobe H., Sandewall E Eds., McGraw-Hill, NewYork, pp 387-413, 1987.

[ha:92] A. Haake. CoVer: A Contextual Version Server for Hypertext Applications. *ECHT92, Proceedings of the ACM Conference on hypertext*, Milano (Italy), November, pp. 43-52, 1992.

[ha:93] A. Haake, J. Haake. Take Cover: Exploiting Version Support in Coopertative Systems. *Conference Proceedings INTERCHI93*, April 24-29, pp. 406-413, 1993.

[ha:88] F. G. Halasz. Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836-852, July 1988.

[ha:91] F. G. Halasz "Seven issues": revisited. Slides from keynote talk at *Hypertext91*, December 1991.

[ka:90] R. H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases *ACM Computing Surveys*, 22(4):375-408, December 1990.

[ma:93] C. Maioli, S. Sola, F. Vitali. Versioning issues in a Collaborative Distributed Hypertext System. *Technical Report UBLCS-93-6.* Aprile 1993.

[mu:89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4):541-580, April 1989.

[os:92] K. Osterbye. Structural and Cognitive Problems in Providing Version Control for Hypertext. *ECHT92, Proceedings of the ACM Conference on hypertext*, Milano (Italy), November, pp. 33-42, 1992.

[pe:81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, N. J., 1981.

[sh:93] D. E. Shackelford, J. B. Smith, F. D. Smith. The architecture and Implementatation of a Distributed Hypermedia Storage System. *ACM Hypertext93 Proceedings*, Seattle, Washington USA, November 1-13, pp. 14-24, 1993.

[st:89] P. D. Stotts, R. Furuta. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, 7(1):3-29, January 1989.

[st:90] P. D. Stotts, R. Furuta. Hierarchy, composition, scripting languages, and translators for a structured hypertext. *ECHT90, Proceedings of the ACM Conference on Hypertext 90*, Cambridge, pp. 180-193, 1990.

[ti:85] W. F. Tichy. Rcs - a system for version control. *Software-Experience and Practice*, 1985.

[ut 89] K Utting, N Yankeolovich. Context and Orientation in Hypermedia Networks. *ACM Transactions on Information Systems*, 7(1): 58-84, January 1989.

[wi·93] K U. Wiil, J J. Leggett. Concurrency Control in Collaborative Hypertext Systems *ACM Hypertext93 Proceedings*, Seattle, Washington USA, November 14-18, pp. 14-24, 1993.