修士学位論文
電子文書の長期的保管のための枠組

東京大学大学院総合文化研究科
広域科学専攻 広域システム科学系
川合研究室 学生証番号86802
伊戸川 暁

平成12年1月21日

要旨

電子計算機 (特に計算速度や記憶容量) の長足の進歩やネットワークの急激な発達の結果、従来なら紙の上に記されてきたであろう文字情報が電子的媒体に記されることがますます多くなり、今や、質・量ともに電子情報の重要性は増す一方である。

そのような状況において現出した膨大な電子情報の中には、後世から見て歴史的に重要な価値を持つことになるであろう文書が少なからず含まれていることになり、従ってそれらの文書を適切に管理する手法を確立することは、電子文書の時代の黎明期に生きる我々にとって急務であるはずである。

にもかかわらず、電子情報の管理のための技術に関する研究は、即時的な情報の共有については比較的進んでいるが、長期的な保存に関するそれは、文書自体を支える媒体の寿命が不十分であることもさることながら、文書に付随する管理状態、文書の恒久的な指示方法、管理者情報の組織化など、技術的なものだけを取っても、次に掲げるような多くの問題を抱えたままになっている。

- 媒体の物理的寿命が必要な水準 (数百年程度) に足りない。
- 技術革新のテンポが激しいので、文書を収めたハードウェアや文書が必要とするソフトウェアがすぐに忘れ去られてしまう。
- 恒久的な使用に耐える命名体系が実装されていない。
- 現行の多くの情報システムでは、管理者と文書の間の連係が十分ではない。
- 文書の改竄への対応など、セキュリティの問題が十分解決されていない。
- ライフサイクル管理に応用できる価値判断の基準が、電子文書については未整備である。
- 以上の問題の解決に資するようなメタデータの表現形式が定められていない。

むろんこれらの他に政治的・法律的問題も多々あるが、それらは本論文の範囲を越えた問題である。

既に図書館情報学方面を中心とした方面から、この課題を解決するべく様々な提案がなされている。しかしそれらはあくまで利用者の立場に立った提案に留まり、詳細なモデルや実装を伴ったものはない。

一方、計算機科学の方面でも、電子文書の長期保存に関した問題群の一部に解を与えるようなシステムは散発的に存在していたが、問題群を包括的に解決するための努力は、計算機科学の歴史を通じて、殆んどなされてこなかった。

そこで、本論文では、まず両者のギャップを埋めるための解を与える際の基本方針を次のように定めた。

- 文書の寿命が媒体の寿命に拘束されないようにすること。
- 任意の情報の形式に関する情報を保持すること。
- 文書に永続的な名称が与えられること。
- ライフサイクル管理を支援すること。
- Worldwide な管理組織を確保すること。
- 各文書の管理責任が明らかにされること。
- 文書の引き継ぎ (管理者の変更) が確実に行われること。
- 文書の改竄に対策を施すこと。
- 文書の流出に対策を施すこと。
- 第三者が任意の文書の管理の適切さを検証できること。
- ネットワークを通じた文書情報の交換を容易にすること。

続いて、それらの方針に則った包括的なソフトウェア技術的枠組として、**POT(Persistent Online Text)** モデルを提案する。POT モデルは、長期的な (百年単位の) 使用に耐え、かつ、後世の検証にも耐えることを目的とした、長期的電子文書保管システムのためのモデルである。

POT モデルは、大別して、命名規則 ·文書モデル ·階層的アーキテクチャの 3 部より成る。

1. 命名規則は、全ての利用者や文書、さらには時刻の表現など、システム内の文字列で表現されうるあらゆる要素への命名のしかたを与える規則である。
   これによって、様々なリソースに筋道の通った命名を与えることができるようになると同時に、文書モデルの (b) に定める文書の種別や管理者組織を一意的な文書識別子に反映させることが可能になる。
2. 文書モデルは、記録すべき情報自体とそれを管理するための情報とを統一的に扱いつつ、他の文書への参照の集まりと保存の主たる対象である本文とを 1 つの単位として扱うモデルである。
   文書モデルは、以下の 2 つに細分される。
   (a) 文書構造モデル。本システムでの情報の基本的な単位となる **POT** 文書の基本的な構造について定めるモデル。POT 文書は、識別子 ·本文 ·リンクの集合 ·ログの集合の 4 部より成り、リンクは他の POT 文書を参照する。これによってシステムが扱う情報の構造化のための基礎が形作られる。
   (b) 文書関係モデル。文書の種別とその役割分担の定義をするモデル。文書の種別には通常文書 ·仕様書 ·認証文書 ·制御文書の 4 つが存在する。これらを互いに適切にリンクさせることによって、文書の公開 ·廃棄 ·制御 ·履歴 ·認証 ·解釈情報が組織化される。
3. 階層的アーキテクチャは、命名規則及び文書モデルのほか、システムの物理的要件をも秩序立てるための階層構造であり、解釈層 ·文書層 ·分散層 ·物理層の 4 層より成る。
   物理層は媒体を永続化させ、分散層は POT 文書をネットワーク透過にし、文書層は文書構造モデルを実現し、分散層は文書関係モデルを実現する。

以上のモデルは、認証文書などの管理情報もまた一般の文書と同様に扱うことによって管理情報の寿命を保証し、また、ネットワークを通じて情報を共有することによってスタンドアロンのシステムよりも高い信頼性をシステムに与える。

以上のモデルに加えて、些細ではあるが重要な以下の2点をも定めた。

文書はXMLで表現すべきこと POTシステムにおいて用いられる文書は、媒体にバイト列として記される他、データ列としてネットワークをも流通する。本システムでは、メタデータを効率良く、かつ可読性の高い形式で記述できるXMLを、POT文書の記述形式として利用することにした。

初期状態において存在すべき文書 1つの文書を登録するにあたっては、その文書に先立って、幾つかの管理文書が必要になる。そこで、システムの運用を始めた時点で既に存在しているべき文書と、それらの文書が持つべきリンク関係について定めた。

本論文の後半では、階層アーキテクチャの下位の2層(物理層・分散層)の実装可能性について、既存の技術を紹介しつつ検討を加えた後、階層アーキテクチャの上位の2層に関わるモデル、すなわち文書モデルに対してより重点的な検討を加える。

文書関係モデルには、例えば、

- 変更/削除が可能な要素の変更/削除
- 管理文書のリンクの連鎖に関する妥当性

など、問題を引き起こす恐れのある操作が多い。

そこで、

- 所有者自身による文書要素の書き換え
- 所有者以外によって文書が書き換えられる場合
- 制御文書が指定する文書の公開範囲の整合性
- リンクが循環してしまう可能性

について踏み入った検討を行い、単純な文書関係モデルの上にさらに施すべき制約やモデル自体の限界を導出した。

具体的には、

- 各種文書の書き換えをして良いところと悪いところの決定
- 制御文書同士のリンクには或る制限が加えられる

などである。

続いて、POTモデルに従ったプロトタイプシステムをJDK1.1.7 + XML for Java 1.1.14によって(不完全ながらも)実装し、

- 文書の出納

- 利用者の登録/抹消
- 所有権の移転

などの、想定される幾つかのユースケースによりつつ、モデルとその理論的検証の結果を実地に追試してみた。その結果、POT モデルは十分に実装可能なものであることが証明された。

結論として、POT モデルが冒頭の目標を達成していることを確かめ、更にモデルの長所と短所について論じる。続いて、官公庁の文書の出納や篤志家による電子文書の蒐集など、POT モデルに基づくシステムの応用の可能性を挙げ、最後に、暗号の使用や分散層の実現方式の検討など、本研究では果たせなかったモデルへの拡張を今後の課題として述べた。

目 次

第1章　序論

1.1　文書を保存する意義

自らの活動について記録を遺すことは、人類の文化の発展において極めて重要な要件である。

文字 (あるいは図形など) の形で自分の考えたことを書き留めることによって、ものを考えた人が死んでも、その人が考えた事柄は未来に残り、将来の人文・学術・人倫の発展を促す。

近代社会が到来して生産・消費活動が活発になるにつれて、社会生活のさまざまな局面において、生産される文書の数量は増大の一途を辿った。情報の増大傾向は20世紀末の今日に至るまで着実に続き、ついには電子計算機の長足の進歩とあいまって、今まで紙の上に記されてきた電子情報は電子媒体の上に移されたり、新たに紙に記される段階を経ること無く、はじめから電子媒体に記される文書も近年急激に増えてきた。こうして、我々が有している情報の多くが電子的媒体に記録されるようになってきている。

既に電子的な人類の記憶が出現する以前に、文書館など、重要な文書を確実に伝えるための制度は確立しており、人文の発達に図り知れない恩恵を与え続けてきた。このことを鑑みたとき、電子媒体に記録されている情報が、その改竄・消去されやすいという特徴により完全に消滅してしまうのだとすれば、人類の記憶にとってまことに大きな損失であると言えるだろう。

電子情報の世界においても文書館の相当物を設立することは、電子情報の時代の黎明期に生きる我々の急務ではないだろうか。

1.2　文書電子化への政治的対応

日々重要度を増しつつある電子情報に対応するため、内外の関係諸機関は、様々な対応を迫られている。ここでは主に公的セクターにおける対応を概観する。(なお、ここでは「政治的」という言葉を、「技術的ではない」くらいの意味で用いている)。

日本の紙文書の管理については一応の基準が既に官公庁に存在しているが、電子文書のそれについては、特に規則を定めていない地方自治体も多いなど、未だ不備が多い。

しかし、1998(平成10)年3月30日に「電子計算機を使用して作成する国税関係帳簿書類の保存方法等の特例に関する法律」が可決され[25]、企業などの帳簿に関する文書については、いちいち紙に出力しなくても良いこととなるなど、電子的形態を持つ文書への配慮がなされるようになり、既に膨大となった電子文書の存在について、はっきりと意識せざるを得ない状況が既に現出している。

また、1999(平成11)年には情報公開法が国会で可決され[26]、確固たる基準に基づいた公文書の公開が官公庁に義務づけられるようになってもいる。

また、特定の国家の施策ではないが、ISO (International Organization for Standardization)[5] が制定した品質保証の国際規格である ISO9000 シリーズは、電子文書管理の国際規格をその一部として含んでいる。

1.3　電子文書における適切な技術的枠組の不備

政策的方針 (ポリシー) の策定と技術的基盤作りとは、文書管理 (に限らず、何事でもそうだが) を実行する上での両輪である。ポリシーがなければ、それに見合った技術を開発することはできず、また、技術的基盤が整備していなければ、ポリシーは実行できない。

しかし、電子文書保存に関する議論は、前節に掲げたような、図書館情報学などの関係者による「こんなのができたらいいね」という仕様の提案、つまりポリシーの策定のみが先行し、その仕様に対して技術的にモデルを建てて検証するという、技術的基盤を作る作業の方はまるで追い付いていないというのが実情である。

こうなってしまった理由について、ARMA International 元会長の D. O. Stephens は、電子情報の扱い方に関する情報科学での研究の主流は、情報の即時的な伝達の方面に置かれ、長期的視野に立った保管の研究はおろそかにされがちであったと指摘している [17]。

また、アメリカの The Research Libraries Group も、[15] において、電子文書の長期的保存のための然るべき枠組が研究されるべきことなどを訴えているが、これに応えた研究の出足は鈍い。

以上述べたような、質・量ともにますます重要度を帯びつつある電子情報の増大と裏腹な管理手法の未整備という現状に鑑み、本論文では、電子的に蓄積された情報を数百年のスパンで保持するための枠組を提案・検証する。

第2章　問題と解決への方針

本章では、紙媒体と同じ、あるいはそれ以上の確度で電子文書を適切に管理するに当たっての問題は詳しくは何か、そしてそれらに与えることのできる解決策は何かについて検証する。

2.1　問題とその詳細

本節では、電子文書を長期保管しようとしたときに直面する問題群を列挙し、その各々について詳述する。

2.1.1　媒体の物理的寿命

現行普通に存在する媒体の寿命は非常に限られている。最も普通に使われている媒体であるハードディスクは、磁気的に記憶を行っているため、寿命はよくて数十年とされている。これでは本研究の目的である数百年のオーダーには全く届かない。しかもこの値は使用中に1度もクラッシュを起こさなかった場合の値であり、クラッシュ発生時にはこれよりも寿命は短くなる。また、CD-ROMやCD-Rは、磁気的な記録をしていないのでより長く寿命が取れると期待されているが、媒体の出現から十数年しか経っていないこともあり、公称されている寿命の確証が得られているわけではない。

要するに、現行の技術では、電子情報を永続的に記録するための単一の媒体(紙の上に墨で書くのが1000年以上の寿命を既に証明しているような)は未だに存在しないし、よしんば存在しても検証もされていないのである。

2.1.2　媒体の技術革新による寿命

しかし、時によっては、媒体そのものの寿命というよりはむしろ、媒体を取り巻く環境によってデータが読めなくなることもしばしばある。電子計算機の発展は未だにソフト・ハードともに激しく、そのため、媒体の物理的寿命が来る前に使用不能になるケースも多い。

一例を挙げると、1960年のアメリカ合衆国の国勢調査の調査結果は磁気テープに「永久保存」され、テープ自体は未だ残っているが、そのテープを読み出すのに必要なハードウェアは、今やスミソニアン博物館に1台、日本のどこかに1台の計2台しか存在しておらず、研究者はそのテープの内容のプリントアウトを必要とあれば手で再入力しなければならないのだという[15][17]。

2.1.3　文書の形式が失われる危険

前項に挙げたハードウェアの問題が解決できればよいかといえば決してそうではなく、文書そのもの以外の要因でも、容易に電子文書は利用できなくなる。最初に挙げる問題は、前節のソフトウェア版である。

例えば、現在は普通に用いられていないワードプロセッサで作成した文書は、文書それ自身が残っていても、文書を解読する方法が失われてしまった場合、表示したり印刷したりすることができなくなってしまう。

従って、文書を有効に保管するためには、文書そのもののみならず、文書のファイル形式などへの情報も必要となる。

2.1.4　命名体系の弱さ

従来の紙でできた文書の場合、公開時の名前がそのまま永久に文書を指し示す名前として用いられる。しごく当然のことを言っているようだが、例えば、電子文書の配布手段として最も広く用いられている HTTP[4] の場合、文書の参照はファイルの物理的所在を指示することで行っているので、物理的所在が変化すると参照が無効になってしまうという問題がある [22]。

これは、紙の世界で例えて言えば、書物を書名ではなく、図書館の書架の位置のみで指定しているようなものであり、極めて脆弱であり、危険である。

2.1.5　価値判断に関する問題

この世に生成される全ての電子情報を永久に保存することはむろんコスト的に不可能であるし、有益な情報を凡庸な情報の山の中に埋もれさせることにもなる。電子情報の量は特に膨大なので、そのようなことが起こると、有益な情報を掘り当てるにために大変な労力が必要となってしまう。

このような事態を承けて、既に紙ベースの文書管理の世界でも、20 世紀に入ったあたりからライフサイクル管理が一般的になされるようになっていた。ライフサイクル管理とは、大まかにいえば個々の文書の運命を作成時から決定しようというもので、近代社会の発展によってますます増大した文書の事後的な選別が人間の手に負えなくなってきたところから考案されたアイデアである。

2.1.6　管理組織の管理の不備

数百年という、人間の寿命を越えたオーダーで文書を保管することを目指すためには、文書の管理者権限を、人間を入れ換えつつシステマチックに保ち続けることが可能でなければならないが、現行の情報システムでは、文書の管理と利用者の管理とが全く独立しているのが常態である。

また、文書とその管理者との対応は、個々のサイトを越えて知られるべき出あるが、この点でも現行ユーザ管理は、利用者の管理は個々の計算機 (やネットワーク) の内側の問題として処理され、worldwide な問題としては考えられてこなかった。

2.1.7　セキュリティの問題

電子文書は複製・改竄が容易であるので、電子文書を故意の改竄（著者によるものを含む）などから守るための手段を講じなければならない。

文献 [1] によると、電子文書の保護のためには、以下の 3 つの性質が満たされなければならない。

機密性 (Confidentiality) 認証を与えていない者から、オブジェクト及びそれへのアクセスが保護されること。完全に公開されていない文書に対しては、それを流出から保護する手段が必要であるし、「時間に関する情報」も重要となる。

一貫性 (Integrity) オブジェクトが改竄されないようにすること。

原本性 (Authenticity) オブジェクトの原本が作者/所有者に関連づけられること。原本性は確実な一貫性に基づくとする。

2.1.8　メタデータの標準形式の欠如

永続的に電子情報を管理するためには、文書本体の他にも様々な補助情報が必要になってくる。このような補助情報を、元の文書のメタデータという。例えば、書物そのものがデータならば、その書物の題名、著者、出版者などの書誌情報はメタデータである。

一般的なメタデータの標準形式については、以下に挙げるような先行研究がいくつか存在する。

Dublin Core[23] 書誌管理用に設計されたメタデータのための簡潔な枠組。メタデータを著者名、書名などの 15 項目にわたって定義し、その書式を定めている。

Warwick Framework[13] Dublin Core に基づき、もう少し複雑な書誌情報を記述できるようにするもの。

PICS[10] 主に WWW 文書のアクセス制限に用いるために考えられたメタデータの枠組。

RDF[20] XML[11] でメタデータを記述するための汎用的な枠組。

しかし、本論文が解決すべき課題である電子文書の長期保存のために特化したメタデータの形式については、先行研究を未だ見ない。

2.2　本研究の目的

以上に挙げた問題点を踏まえつつ、電子文書長期保管問題の包括的な解決のための解を与えるための方針を以下に記す。項目名の後ろには、第 2.1 節中の関係する節番号を付した。

文書の寿命が媒体の寿命に拘束されないようにすること (第 2.1.1 節、第 2.1.2 節)

使用期限の異なる複数の媒体を用意して、媒体の寿命が近付いてきたら、あらかじめ用意された寿命のより長い別の媒体に移動させるような機構が必要になるであろう。

この技術を媒体変換といい、文書保管の上で、重要な基礎技術となることが目されている [15]。

任意の情報の形式に関する情報を保持すること (第2.1.3節)

情報自体のみならず、情報を読むための情報もまた保管されなければならない。

文書に永続的な名称が与えられること (第2.1.4節)

要するに、サイトの移動に伴って変更になってしまうURLのようなもので文書を指示してはいけないということである。

ライフサイクル管理を支援すること (第2.1.8節)

永久的保存に値する文書が最初に収集した文書の中から適切に選ばれるために、あるいは、保存年限が決まっていて最終的には捨てられる運命にある文書でも適切に管理されるために、あらかじめ文書ごとに存続期間・公開範囲などが指定できるようにすべきである。加えて、管理計画に関する情報も保管の対象とし、文書自体と同じ程度の慎重さで扱うべきである。

Worldwideな管理組織を確保すること (第2.1.6節)

管理組織の一貫性をworldwideに保つためには、永続的な識別名(前項参照)が文書の管理者に対しても与えられるべきであるし、また、複数の管理者にわたって、文書の公開・管理記録が引き継がれていく仕組みが必要である。

また、保存されるであろう電子文書の分量は広範囲かつ大量になるため、少数のサイトが文書群を集権的に管理することは困難であることが予想される。従って、各所に多数の管理者権限を置いて分散的に管理する体制を整える必要があろう。

各文書の管理責任が明らかにされること (第2.1.6節)

各々の文書について、誰が責任をもっているかがシステム的に明示できなければならない。

文書の引き継ぎ(管理者の変更)が確実に行われること (第2.1.6節)

引き継ぎの方法が明確になっていないために文書が失われる可能性がある。そこで管理者の確実な引き継ぎをシステムとしてサポートし、後顧の憂いを断つことが重要である。

文書の改竄に対策を施すこと (第2.1.7節)

電子情報は改竄・複製が非常に容易である。従って、管理者以外の人物によって、文書が改竄されないような手段をそなえるべきである。

文書の流出に対策を施すこと (第2.1.7節)

外部に流出すべきでない文書を保護するため、流出を防ぐためにアクセス制限をかける必要が生ずる。また、万一流出が生じても、追跡できるような仕掛けがあることが望ましい。

第三者が任意の文書の管理の適切さを検証できること (第2.1.7節)

これは、セキュリティを向上するために第三者の審級を持ち込もうという発想である。

文書の来歴・管理情報の適切さ等々について、第三者が充分な情報を得ることができるようにすると、たとえ改竄・流出が起こってしまっても、管理者の管理の仕方を第三者が検証できるならば、どこで不適切な管理を行ったかが公正に判断されるようになるであろうし、

信用できる管理者とできない管理者が客観的に評価され、管理者の質を結果として高めることにもなる筈である。

ネットワークを通じた文書情報の交換を容易にすること

必ずしも必要な要件ではなかったので第2.1節内に節を設けなかったが、遠隔地に文書を運ぶことによるリスクの分散が容易になるなど、大目的を達成するためには、あると非常に便利な機能である。

以下の事項については、本研究の対象外とする。

管理ポリシーの自動生成

入力した文書について、自動的に価値判断するような機構は、あれば確かに便利だが、このような価値判断は、管理スケジュールという形で、システムの外部から与えられるもの仮定する。

書誌情報の管理・検索

また、近年の電子文書関係の研究の流行である、書誌情報の管理や検索、さらには異版管理についても、本論文で研究するシステムの上にアプリケーションとして作られるべきものと捉え、研究の範囲外とした。

第3章　モデルの概観

提案するモデル本体の詳述に先立ち、本章では、POTの基本的構造の解説を行うほか、次章以降で断りもなく用いるであろう諸概念についての定義・解説をも与える。

3.1　モデルの機能

POT(Persistent On-line Text)モデルは、電子的情報を長期的に保存・管理するための、多層式クライアント・サーバモデルである。

本システムの大きな目的は以下の通りである。

- 全世界から同様に見える巨大な文書貯蔵庫を提供する
- 利用者が登録した文書を永続的に保存することを保証する
- 登録した文書と永久に1対1対応する識別子を提供する

3.2　設計上の特徴

利用者から見て、POTシステムは以下のような特徴を持つ。

全てのリソースは文書である：POTが保存するデータには、主な対象であるデータ一次的データを本文に持つものと、一時的データの管理のために必要な二次的データを本文に持つものの2つがある。どちらの種類のリソースをも「文書」という単位で表現するのがPOTの特徴である。

文書同士の関係はリンクによって与えられる：POT文書は、そのプリミティブとして、他の文書へのリンクを有する。一次的データを収めた文書から二次的データを収めた文書にリンクを張ることによって、メタデータの表現を実現する。

管理のための情報は別の文書に記される：文書自体とその管理権限や管理スケジュールを切り離すことによって、複数の文書に同一のメタデータを関連づけることを可能とする。

第4章　文書モデル

本章では、POT の根幹をなす、文書構造モデル及び文書関係モデル (まとめて文書モデル) について解説する。

4.1　文書構造モデル

POT システムに収められる情報は、全て **POT 文書** (あるいは以下しばしば単に「文書」) という単位より成る。

POT 文書 ($\langle Doc\rangle$) は、文書識別子 ($\langle DocID\rangle$, 第 5.2 節参照)、本文 ($\langle Text\rangle$)、他の文書との関係を記述するリンク部 ($\langle Links\rangle$)、そして本文とリンク部の変更履歴を収めるログ ($\langle Logs\rangle$) の四部より成る。

文書識別子 全ての文書を一意に定義する識別子。詳細は第 5.2 節を見よ。

本文 ($\langle Text\rangle$ 保存すべき情報を収める部分。

リンク集合 ($\langle Links\rangle$) 複数のリンクを 1 つにまとめたもの。

リンク ($\langle Link\rangle$) 管理上必要な他の文書を指し示すもの。形式的には参照先の文書の文書識別子に等しい。

ログ ($\langle Logs\rangle$) 文書の作成・更新・削除に関する記録。常に公開され、文書管理の妥当性が公に検証されるようにする。

ログ要素 ($\langle LogItem\rangle$) ログの構成要素。ログ見出し・絶対時刻 (ログ要素の生成時刻を表す)・ログ詳細の 3 要素より成る。システムが自動的に生成し、利用者が直接操作することはできない。

ログ見出し ($\langle LogType\rangle$) 文書に起こった事象を大別して記したもの。文書識別子に等しいときは、関連するリンクに変更があったことを表し、それ以外の場合は他の箇所に変更があったことを表す。

ログ詳細 ($\langle LogDetail\rangle$) 文書に起こった事象の詳細な情報を与える。

以上を形式的に記すと、以下のようになる。

- $\langle Doc\rangle ::= \langle DocID\rangle, \langle Logs\rangle, \langle Links\rangle, \langle Text\rangle$
- $\langle Text\rangle ::= \langle Character\rangle+$

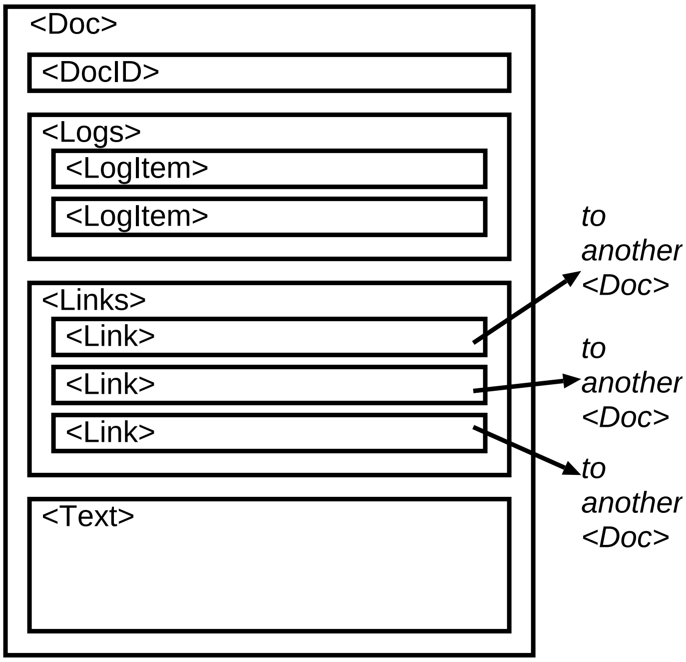


図 4.1: POT 文書の構造

- $\langle Links \rangle ::= \langle Link \rangle +$
- $\langle Link \rangle ::= \langle DocID \rangle +$
- $\langle Logs \rangle ::= \langle LogItem \rangle *$
- $\langle LogItem \rangle ::= \langle LogType \rangle, \langle AbsoluteDate \rangle, \langle LogDetail \rangle$
- $\langle LogType \rangle ::= \texttt{create} | \texttt{delete} | \langle DocID \rangle$

大略を図示すると図 4.1 のようになる。

また、以下に、文書の重要な部位の略記法をまとめる。

- $id(d) : d \in \langle Doc \rangle$ の文書識別子 ($\in \langle DocID \rangle$)。
- $text(d) : d \in \langle Doc \rangle$ の本文 ($\in \langle Text \rangle$)。
- $logs(d) : d \in \langle Doc \rangle$ のログ ($\in \langle Logs \rangle$)。
- $creationDate(d) : d \in \langle Doc \rangle$ の登録時刻 ($\in \langle AbsoluteDate \rangle$)。

4.2　文書関係モデル

前節の「文書構造モデル」のみでは、どれがデータでどれがメタデータか、メタデータのうちどれが利用者に関する情報か、などの情報を表現することができない。従って、本節に解説する文書

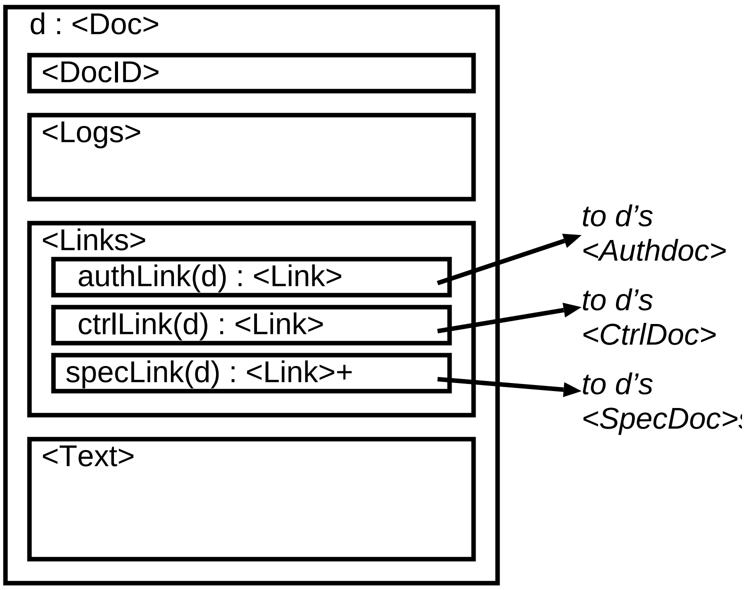


図 4.2: 通常文書の構造

関係モデルによって、データに対するメタデータの種類、及び文書構造モデルを用いたデータとメタデータの表現法を規定する。

文書関係モデルにおいては、全ての POT 文書は、以下のいずれかに属する。

通常文書 ($\langle OrdnDoc \rangle$) 「普通の」文書。

仕様書 ($\langle SpecDoc \rangle$) リンク元の通常文書の解釈の仕方を記した文書。

認証文書 ($\langle AuthDoc \rangle$) 利用者に関する情報を記した文書。

制御文書 ($\langle CtrlDoc \rangle$) リンク元の文書の管理スケジュールを記した文書。

通常文書以外の種別の文書を、総称して管理文書と呼ぶ。

POT 文書は通常文書を派生し、通常文書は更に管理文書を派生するという関係にある。

4.2.1　通常文書

通常文書は、POT 文書に、「種別ごとのリンク」の概念を追加したものである (図 4.2 を参照)。以下に、通常文書の重要な部位の略記法をまとめる。

- $docType(d)$: 文書の種別。$type(id(d))$ に等しい。
- $authLink(d)$: ある文書の認証文書へのリンクである文書識別子。すなわち、$Links(d)$ に含まれる文書識別子 i のうち、$type(i) = \texttt{auth}$ であるもの。$ctrlLink(d)$ などの定義も同じ。

- $ctrlLink(d)$: ある文書の制御文書へのリンクである文書識別子。
- $specLink(d)$: ある文書の仕様書へのリンクである文書識別子の集合。

d に対する $[authLink(d)]$ を、d の所有者と呼ぶ。

4.2.2　仕様書

仕様書は、その本文に、リンク元の文書の解釈の仕方を定める仕様を収める。種別と本文の持つ意味が違う他は、形式の上では通常文書と変わるところはない。

仕様書の本文としては、リンク元文書本文の形式のデータをそのまま扱うことのできるソフトウェアの実行形式を入れることを要求することも検討したが、この方式は取らなかった。なぜなら、特定のハードウェアに対応した実行形式を仕様書の本文として収めても、現在のハードウェアの発展速度が当分の間続くと仮定した場合、実行形式そのものが有効である期間は非常に限られることが明白だからである。

しかし、仕様書を理解する人間さえあれば、将来の任意のハードウェア/ソフトウェア環境の上に読解のための任意の実装を作ることが (原理上) でき、結果として、より文書の寿命を伸ばすことができると期待できる。これが、仕様書の中身をバイナリにしなかった理由である。

4.2.3　認証文書

認証文書は、ログイン時に利用者を認証する時に用いられるほか、任意の文書の所有権を明示するためにリンクされる文書である。

認証文書は以下のような構造を持つ (図 4.3)。

- $\langle AuthText\rangle ::= \langle Passwd\rangle, \langle Announce\rangle$: 認証文書の本文。$\langle AuthText\rangle \subset \langle Text\rangle$。
- $\langle Announce\rangle ::= \langle Message\rangle, \langle Offer\rangle *$: メッセージと Offer を入れる要素。
- $\langle Passwd\rangle ::= \langle Character\rangle +$: パスワード。可能ならば何らかの方式で暗号化されていることが望ましいが、「注意」を参照。
- $\langle Offer\rangle ::= \langle OfferType\rangle, \langle OfferDocID\rangle, \langle Limit\rangle$: 所有者以外の人物が行い、所有者によって承認される必要のある操作を記録する要素。
- $\langle OfferType\rangle ::= \texttt{chown}|\texttt{rmauth}$: 所有者以外の人物が行ったが、所有者によって承認される必要のある操作を記録する要素。`chown` は $\langle OfferDocID\rangle$ 要素に示される文書を所有するべく求められていることを、`rmauth` は制御文書自体が消されようとしていることを示している。
- $\langle OfferDocID\rangle ::= \langle DocID\rangle|\mathsf{null}$: 承認すべき動作関連する文書識別子。$\langle OfferType\rangle$ が `chown` のときは文書識別子、`rmauth` のときは空要素 (null)。
- $\langle Limit\rangle ::= \langle AbsoluteDate\rangle$: この要素が記された認証文書をもつ利用者が $\langle Offer\rangle$ 要素に対して返答しなければならない期限。

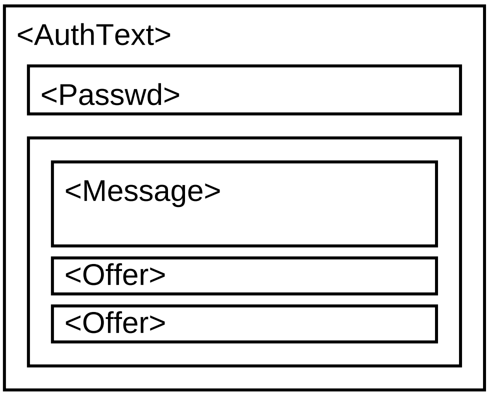


図 4.3: 認証文書の本文の構造

文書への署名のための暗号鍵は、特にこれを規定しなかった。理由は、仕様書にバイナリを収めないようにしたのとほぼ同じである。

distributed.net[3] で行われている暗号解読プロジェクトを引くまでもなく、現在の計算能力の発展速度を考えると、新しい暗号は必ず数年のうちに破られる (無効になる) 運命にあり、将来の相当長い期間にわたっても、本システムが目指す文書の寿命に対抗できるような暗号方式は出現しない可能性が高い。

また、そうでなくとも、暗号の鍵というものは、その性質上永続的に用いるものではない。従って、文書への署名は「消耗品」と考えるべきものであり、少なくとも、文書自体に識別子を付加するような形で署名を付するわけにはいかないということになる。

4.2.4　制御文書

制御文書 ($\langle CtrlDoc \rangle$) は、リンク元の文書の管理スケジュール (どういうタイミングでどれだけの範囲に文書を公開するか、また、削除するか) を記述する文書である。

制御文書本文の書式

制御文書の本文は、時間帯ごとのルール ($\langle TimeRule \rangle$) の集合より成る。更に、$\langle TimeRule \rangle$ は、ドメインごとのルール ($\langle DomainRule \rangle$) の集合より成り、$\langle DomainRule \rangle$ は、指定されたドメインに対するパーミッション $\langle Permission \rangle$ の集合より成る。

これを図示すると図 4.4 のようになり、形式的に記述すると以下のようになる。

- $\langle CtrlText \rangle ::= \langle TimeRule \rangle +$: 制御文書の本文。$\langle CtrlText \rangle \subset \langle Text \rangle$。
- $\langle TimeRule \rangle ::= (\langle Date \rangle, \langle DomainRule \rangle)+$: 時間帯ごとのルール。
- $\langle DomainRule \rangle ::= \langle ExtendDomainName \rangle +, \langle Permission \rangle$: ドメイン (第 5.3 節参照) ごとのルール。

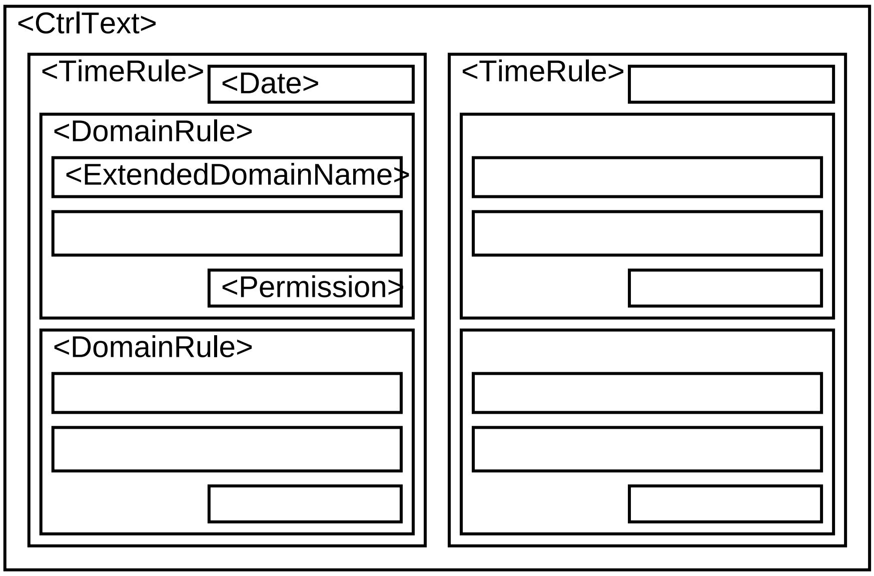


図 4.4: 制御文書の本文の構造

- $\langle ExtendDomainName\rangle$ ::= $\langle DomainName\rangle$|`owner` : `owner` は文書の所有者を意味する。
- $\langle Permission\rangle$ ::= {`all`|`meta`|`delete`} : パーミッション。`all` ならば対象となった文書の全体を見せることを許可し、`meta` ならば本文以外を見せる。`delete` ならばリンク元の文書を「削除」することをエージェント (第 6.5 節参照) に許す。

  公開に制限をかけることのできる文書の要素を本文のみとしたのは、ログ及びリンク要素については、管理計画を外部に対して透明にし、何人でも管理者に文書の責任を問えるようにすることができるという重要なメリットを一般の利用者に与えるためである。

これだけでは分かりづらいと思うので、XML で表現した制御文書の本文の一例を下に掲げる。

```
<text>
  <time value="A-0">
    <range value="root">meta</range>
    <range value="root:foo">all</range>
  </time>
  <time value="A-0-3">
    <range value="root">all</range>
  </time>
</text>
```

この制御情報は、リンクされた任意の文書 d について、d の作成から 3ヶ月以内はドメイン `root:foo` 内の者のみに本文を見ることを許し、3ヶ月経過ののちは永久かつ全ての者に本文を見ることを許している。

第5章　命名規則

5.1　利用者とその識別子

POT システムを利用する主体を利用者 ($\langle User\rangle$) と呼ぶ。(以下、いずれかの文書に管理権を持つ利用者を特に管理者と呼ぶ場合があるが、要は利用者と同じことである。)

全ての利用者は、システムに利用者識別子 ($\langle UserID\rangle$) という記号列を伴って登録される。システムは利用者を利用者識別子によって識別する。ここで、、利用者識別子は、登録時から抹消時まで変化することがないものである。

全ての利用者には、1 つの認証文書が 1 対 1 で対応する。利用者の認証文書の識別子については、第 5.2.2 節で後述する。

以下、便宜のため、「利用者識別子が x である利用者」を $[x]$ と略記する。

システムの初期状態に存在する利用者は、[`root`] のみである。

利用者 $[X]$ は、任意のトークン Y を用いて、新しい利用者をシステムに登録することができる。このとき、新しい利用者の識別子は、$X{:}Y$ となる。また、ここで、$[X]$ は $[X{:}Y]$ に対して上位であり、$[X{:}Y]$ は $[X]$ に対して下位であると定める。

5.2　文書識別子

文書識別子 $\langle DocID\rangle$ は、定められた文字の集合 ($\langle Characters\rangle$) から成るいくつかのトークンを、文字 ‘:’ でつなげた文字列である。

文書識別子は、一度登録されたら取り消すことはできない。文書を「削除」しても、識別子を使用した記録はシステムに残り、再利用はできない。

以下に、文書識別子の例をいくつか示す。

```
auth:root:foo
ctrl:root:permanent
ordn:root:ktyy:itogawa:thesis-draft
spec:root:ietf:rfc:822
```

5.2.1　種別名

第 4.2 節で定義した文書の種別に対し、表 5.1 のように、種別名 $\langle DocType\rangle$ を定義する。

種別	種別名
通常文書	`ordn`
認証文書	`auth`
制御文書	`ctrl`
仕様書	`spec`

表 5.1: 文書の種別と種別名の対応関係

5.2.2 認証文書の識別子

認証文書の識別子は、

種別名 (`auth`) : 利用者識別子

という形式を持つ。言い換えれば、利用者 $[X]$ を特定する認証文書の識別子は、`auth:X` である。

各々の認証文書 D が持つ (別の) 認証文書へのリンク $authLink(D)$ は、登録時の値から変更できないものとする。

5.2.3 認証文書以外の文書識別子

認証文書以外の文書は、

種別名 (`auth` を除く) : 作成した利用者の識別子 : 利用者が定める名前

という形式の識別子を持つ。

そうすると、例えば $[u]$ と $[u:v]$ という 2 人の利用者が存在した場合、$[u:v:w]$ という文書[1]を作成する権利は $[u]$ と $[u:v]$ のどちらもが有することになるが、この場合には、先に登録した方の権利が優先される (図 5.1 参照。図中の三角形が利用者の命名権の及ぶ範囲である)。

しかしこの曖昧さはむしろ意図的に持ち込んだものであり、こうすることによって、例えば $[\texttt{auth}:\texttt{foo}:\texttt{bar}]$ の下に後から利用者 $[\texttt{auth}:\texttt{foo}:\texttt{bar}]$ を出現させることによって、$[type\colon \texttt{foo}:\texttt{bar}]$ 以下の命名権を $[\texttt{auth}:\texttt{foo}:\texttt{bar}]$ に委譲することができるようになる。なお、ここで、$type \in \{\texttt{ordn}, \texttt{ctrl}, \texttt{spec}\}$ である。

つまるところ、認証文書以外の文書識別子は、あくまで全世界的に一意な名前を付するための手段に過ぎず、文書の所属 (誰のものか) については、作成時の所属をしか示唆しない。文書 D のその時々の管理者は、あくまで $authLink(D)$ が示すものであって、これは解釈層の $chown$ に始まる一連のメソッド (後述) によって変更することができる。

文書識別子の定義を BNF を用いて表すと、以下のようになる。

- `<DocID> ::= <NonAuthDocID> | <AuthDocID>`
- `<AuthDocID> ::= 'auth:'  <UserID>`
- `<NonAuthDocID> ::= <NonAuthDocType> ':'  <UserID> ':'  <DocNameID>`

---

[1] 以下、便宜のため、利用者と同様に、「文書識別子が x である文書」を [x] と略記する。

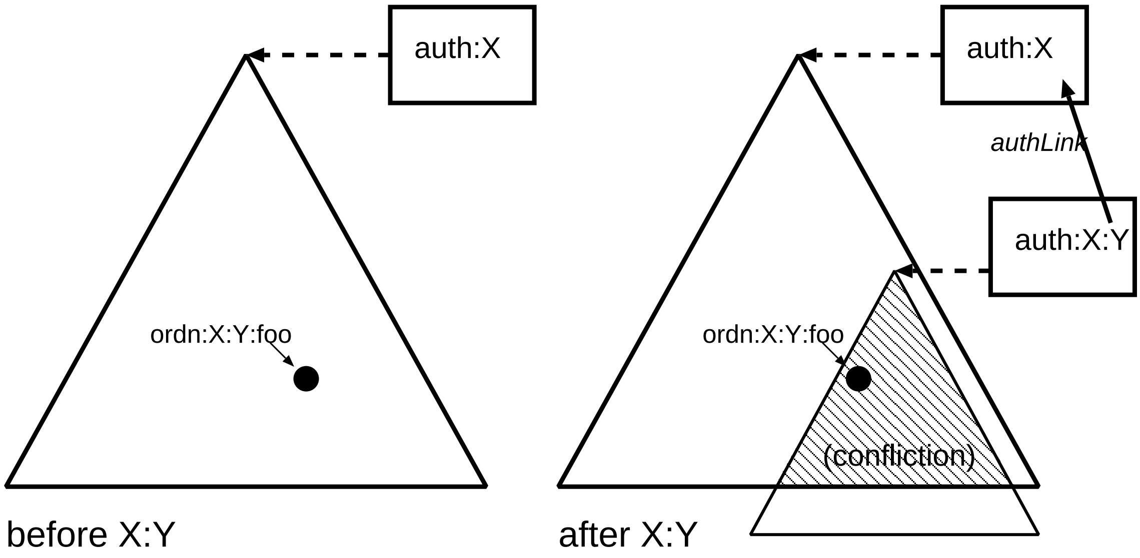


図 5.1: 利用者とその命名権の範囲

- `<UserID> ::= <Token> (':'  <Token>)*`
- `<DocfNameID> ::= <Token> (':'  <Token>)*`
- `<DocType> ::= 'auth' | <NonAuthDocType>`
- `<NonAuthDocType> ::= 'ordn' | 'ctrl' | 'spec'`
- `<Token> ::= <Character>+`

5.3　ドメイン

ドメインは利用者の集合を表す概念で、文書のテキストを個々の利用者に見せるか否かを制御文書に記述するために用いられる。

ドメインの文字列表現は利用者識別子のそれと同じである。

利用者識別子 u がドメイン dom と任意の文字列 r を用いて `u ::= dom ':'  r` のように表記できることを、$u \in dom$ であると定義する。

利用者識別子及びドメインの定義により、ドメイン `root` は全ての利用者を含む。

5.4　時刻識別子

制御文書やログ要素で用いる時刻表示は、以下の 2 種類とする。

1. 絶対時刻: 特定の時刻を表す。ISO8601[6] の書式に従うものとする。

2. 相対時刻: 何らかの絶対時刻(或る文書の作成時刻など。コンテキストによって定まる)からの時間差を表す。

時刻指示子の定義をBNFを用いて表すと、以下のようになる。

- `<Date> ::= <AbsoluteDate> | <RelativeDate>`
- `<AbsoluteDate> ::= (ISO8061に定められた文字列)`
- `<RelativeDate> ::= 'A' ('-' [0-9]+){1, 6}`

以下に時刻識別子の例を示す。

- 絶対時刻の例: `2000-01-07T07:49:19`
- 相対時刻の例(1): `A-1-3`
- 相対時刻の例(2): `A-0-0-0-0-0-15`

上の例は、順に「西暦2000年1月7日午前7時49分19秒(UTC)」、「ある基準となる時刻(文脈によって定まる)から1年3ヶ月後」、「ある基準となる時刻から15分後」を表している。

第6章　階層アーキテクチャとそのメソッド

本章では、POT システムの階層アーキテクチャについて解説する。階層アーキテクチャもまた POT モデルの一部である。

POT アーキテクチャを構成する階層は、以下の 4 つである。

解釈層　文書関係モデルを実装する。

文書層　文書構造モデルを実装する。

分散層　文書をネットワーク透過にする。

物理層　文書の媒体を管理し、電子情報自体が媒体の寿命を越えて存続できるようにする。

以上の層は、各々独立に機能し、隣り合った階層同士のみが互いに通信をする。そのようにして 4 つの階層が上記の順に積み重なったものが、1 つのシステムの全体として機能することになる (図 6.1)。

上記のようにシステムを分割する利点は、前章で述べた多岐にわたる問題の側面を適当な大きさの個別の問題として扱うことができる点である。

また、モジュールではなく階層としたのは、問題の要素の間に依存関係が存在するためである。例えば、本システムの特徴をなす「本文から分離したメタデータ (の一部) の管理」などは、情報一般を永続化することができないと意味をなさない、などである。

6.0.1　新たに登場する型

この章の本題に入る前に、前章までには登場しなかったが、ここから先の解説に登場するオブジェクトの型についてまとめておく。

パス (`Path`)：文書識別子の文字表現の右端から 0 個以上の「‘:’ $\langle Term \rangle$」を切り落としたもの。また、$\langle Path \rangle ::= \langle DocID \rangle | \langle DirID \rangle$。

ディレクトリ識別子 (`DirID`)：パスのうち、いかなる文書識別子とも一致しないもの。

一覧 ($\langle LsDoc \rangle$)：指定したディレクトリがもつ文書の一覧を収めるためのオブジェクト。$\langle LsDoc \rangle ::= \langle LsItem \rangle +$。

文書識別子要素 ($\langle LsItem \rangle$)：$\langle LsDoc \rangle ::= \langle DocID \rangle | \langle DirID \rangle$。

ホスト名 ($\langle HostName \rangle$)：個々の分散層サーバの所在を識別するもの。

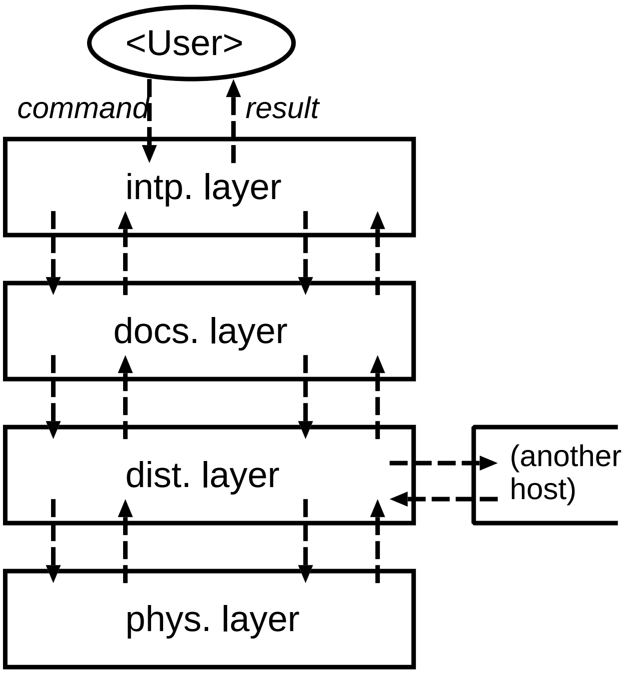


図 6.1: 階層アーキテクチャ

6.0.2　層を越えて用いられるメソッド

$now() \rightarrow \langle AbsoluteDate \rangle$: 現在時刻を表す時刻識別子を返す。

以下、下位の階層から順に、階層毎の持つメソッドについて解説する。

6.1　物理層

物理層は、媒体の寿命を越えて永続的に上位の層から与えられた情報を保管することをその機能とする。

物理層に保存される全ての文書は、制御文書に指定されている期限とは全く関係なく、物理層の実装によって存続のための処置を受ける。

保管のために内部的には複数の媒体を用いていても、外部からは「永続的な 1 つの媒体」に見えなければならない。

内部の実装に関する議論は第 8 章において行うこととし、ここでは物理層が外部に提供する機能についてのみ記述する。

6.1.1 定義するメソッドと各々の機能

$write(id : \langle DocID \rangle, d : \langle Doc \rangle) \rightarrow \langle Status \rangle$: id で指示された位置に d を書き込み、ステータスを返す。

$read(id : \langle DocID \rangle) \rightarrow \langle Doc \rangle$: 文書 $[id]$ を読み出す。

$exists(id : \langle DocID \rangle) \rightarrow \langle boolean \rangle$: id の存否を返す。$read(id)$ の結果が空ならば false、空でなければ true。

$ls(id : \langle DocID \rangle) \rightarrow \langle LsDoc \rangle$: id で指示されたディレクトリ以下の情報を返す。

6.2 分散層

分散層は、POT 文書の構成要素である POT ファイル (以下単に「ファイル」) を複数のサイトに分散して配布し、同一の (永続的な) 文書識別子をグローバルに利用できるようにする。

文書を複数のホストに分散する意義は、以下の通りである。

- 複数のホストに文書を置くことにより、文書が失われる可能性がより少なくなる。
- 遠隔のホストにおくことによって、災害などによって文書が失われる可能性が減ずる。なお、これを期待する場合、ホスト同士はなるべく互いに離れていた方が望ましい。
- 公開されている文書に遠隔から遠隔からアクセスする利便が増す。

この層も物理層と同様、ここでは外部に提供する機能を主に記述し、実装の詳細に関する議論は第8章において行うこととする。

6.2.1 外部に接続する位置

あるホスト A の分散層が他のホスト B に接続するときは、直接 B の物理層につなぐのではなく、B の分散層に接続する。このようにする理由は、「他のホスト」という概念を物理層に介在させなくてもよいようにするためである。以下のメソッド中の $\langle Host \rangle$ 要素は接続の際に容易に得ることができるので、これは大きなオーバーヘッドにはならないと考えられる。

6.2.2 定義するメソッドと各々の機能

ここでは、文書層及び分散層 (に属する他のホスト) に対して定義するメソッドについて解説する。

$put(remote : \langle Host \rangle, d : \langle DocID \rangle) \rightarrow \langle Status \rangle$:

1. 識別子が id である文書 d を自ホストの物理層に登録させる。
2. $serverList$ にある各々のホストの分散層に対して、$put(localHost, d)$ を実行する。
3. ステータスを文書層に返す。返り値は、put が全て失敗したときに限り fail、一つでも成功すれば OK。

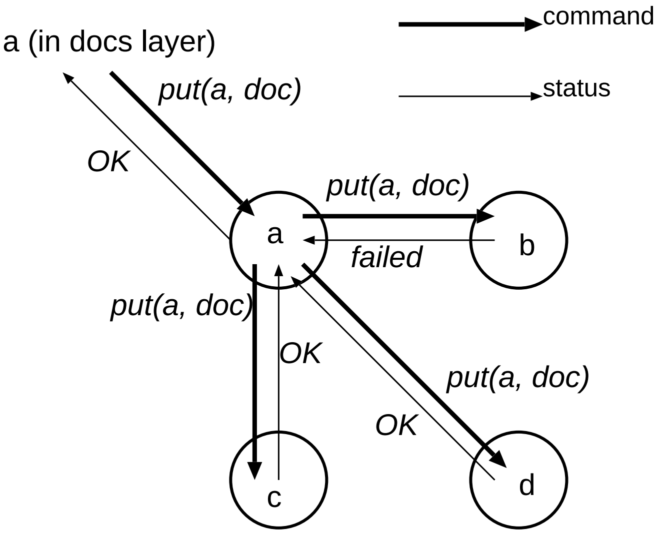


図 6.2: $put()$ コマンドにおけるホスト間の処理の流れ

この流れを図示すると図 6.2 のようになる。

$get(remote : \langle Host\rangle, d : \langle DocID\rangle) \to \langle Status\rangle$ ：識別子が id である文書を文書層に返す。

1. 自ホストの物理層から $[id]$ を探し、あれば文書層に返して終了。
2. なければ、$serverList$ にある各々のホストの分散層に対して順に $get(localHost, d)$ を実行し、見つかったところで文書層に文書を渡して終了。

$exists(remote : \langle Host\rangle, d : \langle DocID\rangle) \to \langle boolean\rangle$ ：

1. まず自ホストの物理層に対して $exists(d)$ を行い、true ならば true を返し、false ならば次に進む。
2. $serverList$ にある各々のホストの分散層に対して順に $exists(d)$ を実行し、見つかったところで文書層に true を返して終了。
3. それでも見つからなければ false を返して終了。

$ls(remote : \langle Host\rangle, d : \langle DocID\rangle) \to \langle LSDoc\rangle$ ：

1. $remote = localHost$ ならば、自ホストの (というか自ホストのしか実行できないが) 物理層の $ls(d)$ を実行し、結果を得る。
2. そうでなければ、自分を除く全ての $serverList$ の各々に対して $ls(localHost, d)$ を発行し、その結果をもらう。
3. 1 及び 2 の結果をマージして文書層に渡す。即ち、1 及び 2 で得た全ての結果を L とおくと、文書層に返す値は、$\bigcup_{l\in L}(l)$ となる。

6.2.3　サーバ内部用の変数/メソッド

$localHost : \langle HostName\rangle$ ： サーバ自身のホスト名。

$serverList : \langle HostName\rangle +$ ： 分散層サーバが接続することができる、他のサーバのホスト名の集合。

6.3　文書層

文書層は、文書構造モデルを実装し、解釈層以上に提供する層である。

6.3.1　解釈層に提供するメソッド

$store(id : \langle DocID\rangle, l : \langle Links\rangle, t : \langle Text\rangle) \rightarrow \langle Status\rangle$ ： 文書を登録・改訂する。

1. l の各々の要素として指定されている $lid := \langle DocID\rangle$ が 1 つでも存在しない場合 ($exists(lid)$ で調べる)、fail が返る。
2. そうでない場合、T(文書層が持つ文書の集合。以下同じ) は以下のように変化する。
   - T に $[id]$ がない場合、T には $\langle id, l, t, \langle \texttt{created}, now(), null\rangle\rangle$ が追加される。具体的には、分散層に対して、$store(\langle id, l, t, \langle \texttt{created}, now(), null\rangle\rangle)$ を発行する。
   - 既に識別子が id である文書 $d' \in T$ がある場合、d' は $\langle id, l, t, newLog(l, t, d')\rangle$ に置き換わる。

$exists(id : \langle DocId\rangle) \rightarrow \langle boolean\rangle$ ： 文書 $[id]$ が存在するか否かを返す。

$retrieve(id : \langle DocId\rangle) \rightarrow \langle Doc\rangle$ ： 文書 $[id]$ を取得する。$[id]$ が存在しない場合、null を返す。

$delete(id : \langle DocId\rangle) \rightarrow \langle Status\rangle$ ： 文書 $[id]$ を「削除」する。すなわち、T のうちで識別子が id である要素を、$\langle id, l, t, h\rangle$ から $\langle id, \textsf{null}, \textsf{null}, \langle h \cup \langle \textsf{deleted}, now(), \textsf{null}\rangle\rangle$ に置き換える。言い替えれば、分散層に対して、$store(id, \textsf{null}, \textsf{null}, \langle h \cup \langle \textsf{deleted}, now(), \textsf{null}\rangle)$ を発行する。文書自体をあとかたもなく消すのではないことに注意すること。

$ls(path : \langle DocID\rangle) \rightarrow \langle LSDoc\rangle$ ： 指定したパス $path$ に対する文書名の集合を返す。

6.3.2　内部で使用するメソッド

$newLog(l : \langle Links\rangle, t : \langle Text\rangle, d' : \langle Doc\rangle) \rightarrow \langle Logs\rangle$ ： 新旧の文書の差分から新しいログ要素を作る。主に、文書を上書きする際に用いられる。

1. $h'' := \textsf{null}$ とおく。ただしここで、$h'' \in \langle Logs\rangle$ である。
2. $t \neq text(d')$ ならば、h'' の要素に $\langle \textsf{text}, now(), t\rangle$ を加える。
3. l のうちで `auth` を含むリンク要素を a としたとき、$a \neq authLink(d')$ ならば、h'' の要素に $\langle \textsf{auth}, now(), a\rangle$ を加える。

4. l のうちで `ctrl` を含むリンク要素を c としたとき、$c \neq \mathit{ctrlLink}(d')$ ならば、h'' の要素に $\langle \mathsf{ctrl}, \mathit{now}(), c\rangle$ を加える。
5. h'' を返り値とする。

6.3.3　内部変数

$\mathit{localHost} : \langle \mathit{HostName}\rangle$: 意味は分散層のそれに同じ。

6.4　解釈層

解釈層は、文書構造モデルを定義する文書層の上に、第 4.2 節で定義した文書関係モデルに関する様々なメソッドを定義する層である。

6.4.1　外部に提供するメソッド

$\mathit{login}(l : \langle \mathit{UserID}\rangle, p : \langle \mathit{Passwd}\rangle) \rightarrow \langle\langle \mathit{UserID}\rangle, \langle \mathit{Announce}\rangle\rangle$: 接続を求めてきた利用者を認証する。実行手順の詳細は以下の通り。

1. $\mathit{verifyLogin}(u, p)$ を実行する。
2. 前項の結果が false ならば $\langle \mathsf{null}, \mathsf{null}\rangle$ を返して終了する。true ならば、$\mathit{retrieve}(u, \mathit{ttauth} : u)$ によって取得した $\langle \mathit{AuthDoc}\rangle$ から $\langle \mathit{Announce}\rangle$ 要素を取り出して a とし、$\langle u, a\rangle$ を返す。

$\mathit{store}(\mathit{type} : \langle \mathit{DocType}\rangle, u : \langle \mathit{userID}\rangle, \mathit{id} : \langle \mathit{DocID}\rangle, \mathit{ctrl} : \langle \mathit{DocID}\rangle, \mathit{spec} : \langle \mathit{DocID}\rangle+, t : \langle \mathit{Text}\rangle) \rightarrow \langle \mathit{Status}\rangle$: u の権限で、識別子が $\mathit{type}, u, \mathit{id}$、 所有者が $[u]$ (i.e. 認証文書へのリンク先が $\texttt{auth} : u$)、制御文書へのリンク先が ctrl、仕様書へのリンク先が spec、本文が t となる文書を、id という名前で登録する (認証文書には、この階層では別のメソッドを用いる)。

1. $\mathit{exists}(\mathit{type}, u, \mathit{id})$ が真ならば forbidden を返して終了。
2. $\mathit{type} = \texttt{auth}$ ならば、forbidden を返して終了。
3. $\mathit{type} = \texttt{ctrl}$ ならば、t の整合性を検査し、不適切ならば defunct を返して終了。
4. $\mathit{ctrl} \Rightarrow \langle \mathit{DocType}\rangle = \mathsf{ctrl}$ かどうかと、$\forall x \in \mathit{spec}(x \rightarrow \langle \mathit{DocType}\rangle = \mathsf{Ctrl}$ かどうかを確認し、どちらかが偽ならば fail を返して終了。
5. 文書層の $\mathit{store}(\langle \mathit{type}, u, \mathit{id}\rangle, \langle \texttt{auth} : u, \mathit{ctrl}, \mathit{spec}\rangle, t)$ を実行し、その返り値をそのまま返す。

下位の層と併せた動作イメージ図は図 6.3 の通り。

$\mathit{storeAuth}(u : \langle \mathit{UserID}\rangle, \mathit{new} : \langle \mathit{UserID}\rangle, \mathit{ctrlID} : \langle \mathit{DocID}\rangle, \mathit{pw} : \langle \mathit{Passwd}\rangle) \rightarrow \langle \mathit{Status}\rangle$: 新しい利用者 $[\mathit{new}]$ を表す認証文書を登録する。

1. $t \in \langle \mathit{Text}\rangle$ を、$t := \langle \mathit{pw}, \langle \mathsf{null}, \mathsf{null}\rangle\rangle$ とおく。

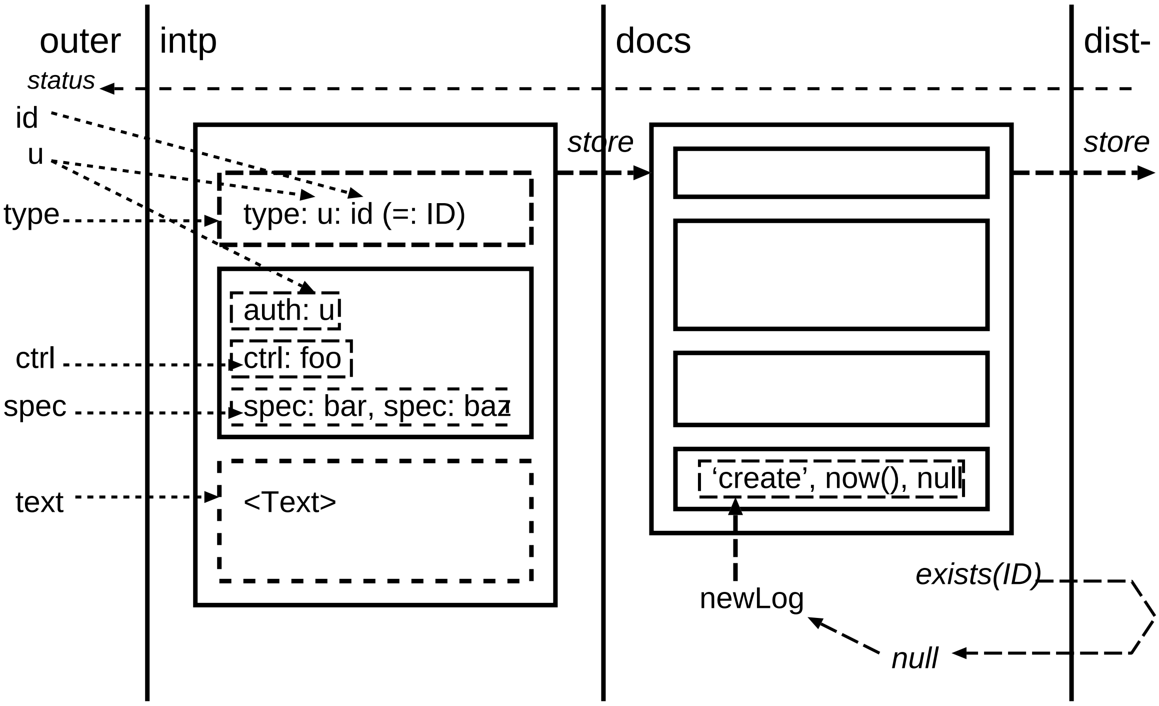


図 6.3: メソッド $store()$ の動作イメージ図

2. 文書層の $store(\texttt{auth}:u{:}new, \texttt{auth}:u, ctrlID, \langle \texttt{spec}:\texttt{root}:\texttt{authspec}, \texttt{spec}:\texttt{root}:\texttt{xml}\rangle, t)$ を実行し、その返り値を返す。

$ls(path:\langle DocID\rangle) \rightarrow \langle LsDoc\rangle$: $path$ の下にある文書識別子の一覧を返す。文書層の $ls(path)$ の結果をそのまま返す。

$exists(id:\langle DocID\rangle) \rightarrow \langle boolean\rangle$: id が存在するかどうかを返す。文書層の $exists(id)$ の結果をそのまま返す。

$retrieve(u:\langle UserID\rangle, id:\langle DocID\rangle) \rightarrow \langle Doc\rangle$: u の権限で、ID が id である文書の全体を読み出す (図 6.4 にイメージ図を示した)。

1. 文書層に対して $retrieve(id)$ を実行し、$d = \langle id, l, t, h\rangle \in \langle Doc\rangle$ を得る。
2. h に deleted を含む要素が存在していれば、利用者にログを見せるため、d をそのまま返す。
3. 前項で得られた d から $ctrl = ctrlLink(d)$ を得、さらにこれによって $c = retrieve(ctrl)$ を得る。
4. $filter(d, c)$ を実行し、これの結果を返り値とする。

制御文書について、再帰的に動作が行われる点に注意すること。ループの可能性については、第 9.4 節で検証する。

$chtext(u:\langle UserID\rangle, id:\langle DocID\rangle, nt:\langle Text\rangle) \rightarrow \langle Status\rangle$: $[u]$ の権限で、$[id]$ の本文を変更する。

1. $type = id \Rightarrow \langle DocType\rangle$ とおき、$type$ が ordn か spec に等しければ、forbidden を返して終了する。
2. 文書層に対して $retrieve(id)$ を実行し、$d = \langle docid, l, t, h\rangle \in \langle Doc\rangle$ を得る。
3. $authLink(d) \neq \langle \mathsf{auth}:u\rangle$ かつ $id(d) \neq \langle \mathsf{auth}, u\rangle$ ならば forbidden を返して終了。
4. そうでなければ、$d' = \langle docid, l, nt, h\rangle \in \langle Doc\rangle$ と置き、文書層に対して $store(d')$ を実行する。

認証文書 a の本文に限り、$authLink(a)$ によって示される管理者のほか、$[a]$ 本人が変更することができる。これは、いかなる場合でも、$[a]$ をしてパスワードの変更を可能ならしめるためである。

$chctrl(u:\langle UserID\rangle, id:\langle DocID\rangle, newCtrlID:\langle DocID\rangle) \rightarrow \langle Status\rangle$: $[u]$ の権限で、$[id]$ の制御文書へのリンクを $newCtrlID$ に変更する。

1. 文書層に対して $retrieve(id)$ を実行し、$d = \langle docid, l, t, h\rangle \in \langle Doc\rangle$ を得る。
2. $authLink(d) \neq u$ ならば forbidden を返して終了。
3. そうでなければ、d 中の $ctrlLink(d)$ を l から $newCtrlID$ に置き換えた文書 d' を作り、文書層に対して $store(d')$ を実行する。

$chown(u:\langle UserID\rangle, new:\langle UserID\rangle, d:\langle DocID\rangle,)limit:((\langle Date\rangle|\mathsf{null}) \rightarrow \langle Status\rangle$: $[d]$ の管理者のみ実行可能。$[u]$ から $[new]$ に $[d]$ を譲渡しようとする。これを実行すると、$[new]$ に $[d]$ に関する $\langle Offer\rangle$ 要素が出現し、サーバは new の login 時に $[d]$ の管理権の受諾を迫るようになる。

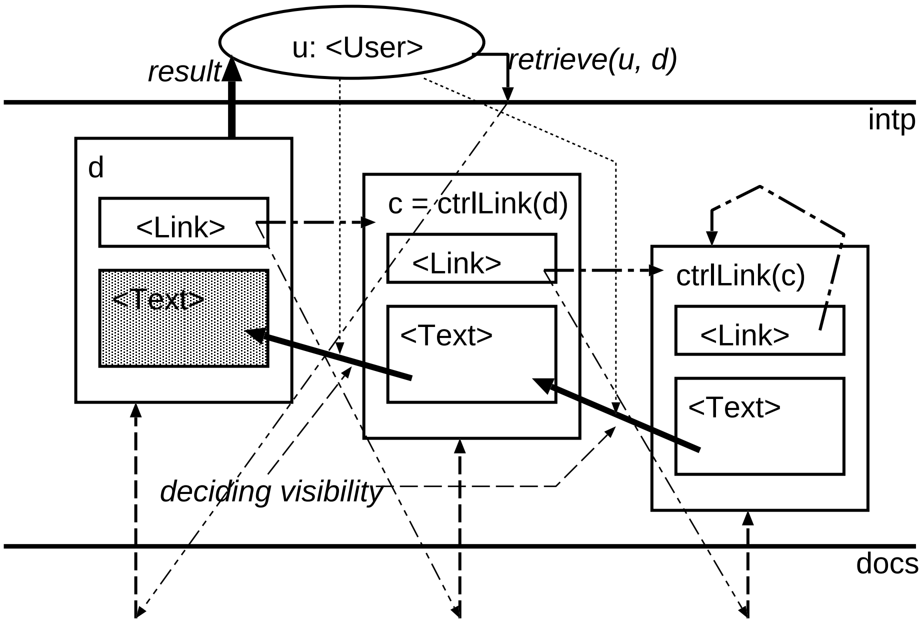


図 6.4: メソッド $retrieve()$ の動作イメージ図

1. $\texttt{auth} : new = authLink(d)$ ならば、same を返して終了。
2. $limit$ が null ならば、$limit$ にはデフォルトの $\langle RelativeDate\rangle$ を入れる。
3. $retrieve(new) \Rightarrow \langle AuthText\rangle$ に、$\langle \mathsf{chown}, d, getAbsouteDate(limit, creationDate(d))\rangle$ を加えたものを $store$ する。

$getOwn(u : \langle UserID\rangle, d : \langle DocID\rangle) \rightarrow \langle Status\rangle$: 文書を譲渡された利用者が $[d]$ を受け取ることを承諾する。これの発行によって初めて、$[d]$ の所有権が $[u]$ に移る。

1. $ud = retrieve(\langle \mathsf{auth} \times u\rangle)$ によって自分の認証文書を取る。
2. $ud \Rightarrow \langle Text\rangle$ から、$\langle \texttt{chown}, d,$ (任意の $\langle AbsoluteDate\rangle)\rangle \in \langle Offer\rangle$ であるような要素 o を取り除く。
3. $o \Rightarrow \langle AbsoluteDate\rangle <= now()$ ならば、outoflimit を返して終了。
4. $store(ud)$ を実行する。
5. $doc = retrieve(d)$ を実行して doc を得る。
6. doc 中の $authLink(doc)$ を $\langle \mathsf{auth}, u\rangle$ に置き換えた文書 d' を作る。文書層に対して $store(d')$ を実行し、その返り値を返す。

$denyOwn(u : \langle UserID\rangle, d : \langle DocID\rangle) \rightarrow \langle Status\rangle$: 文書を譲渡された利用者が $[d]$ を受け取ることを拒否する。

1. $ud = retrieve(\texttt{auth} : au)$ によって自分の認証文書を取る。
2. $ud \Rightarrow \langle Text\rangle$ から、$\langle \texttt{chown}, d,$ (任意の $\langle AbsoluteDate\rangle)\rangle \in \langle Offer\rangle$ であるような要素を取り除く。
3. $store(ud)$ を実行し、その返り値を返す。

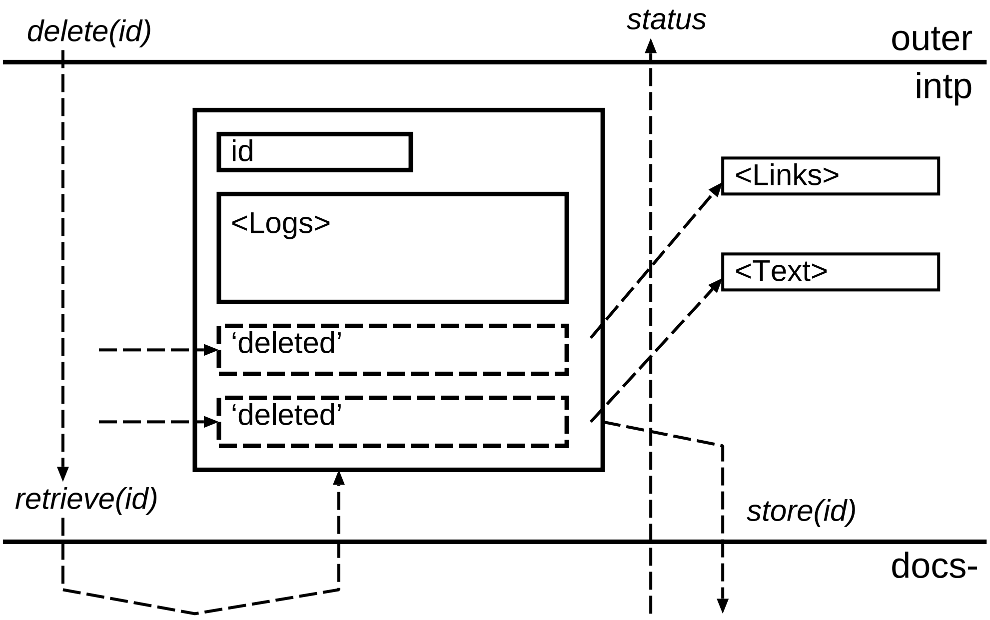


図 6.5: メソッド $delete$ の動作イメージ図

以上の 3 つのメソッドによって、個々の管理者の寿命を越えた文書の管理が可能になる。

$delete(u : \langle UserID\rangle, id : \langle DocID\rangle) \rightarrow \langle Status\rangle$: u の権限で、ID が id である文書を削除しようとする。

1. $d = [id]$ と $[id]$ の制御文書 c を文書層から取得する。
2. $dr \in \langle DomainRule\rangle = getDomainRule(u, c)$ によって適切なルールを得る。
3. $dr \Rightarrow \langle Permission\rangle =$ `delete` ならば、文書層の $delete(d)$ を実行し、その返り値を返す。そうでなければ、forbidden を返す。

イメージ図を図 6.5 に示した。また、`auth` : k にリンクしている文書へのフォローについては、第 6.5 節を参照すること。

$rmauth(u : UserID, k : UserID)$: $[k]$ の管理者のみ実行可能。認証文書 k を削除することによって $[k]$ のアカウントを抹消しようとする。

1. $limit$ が null ならば、$limit$ にはデフォルトの $\langle RelativeDate\rangle$ を入れる。
2. $retrieve(k) \Rightarrow \langle AuthText\rangle$ に、$\langle \texttt{rmauth}, d, getAbsouteDate(limit, creationDate(d))\rangle \in \langle Offer\rangle$ という要素を加えたものを $store$ する。

$getrmauth(u : UserID)$: $[u]$ 自身が自分を消そうとする Offer を承諾する。

1. $ud = retrieve(\texttt{auth} : u)$ によって自分の認証文書を取る。
2. $ud \Rightarrow \langle Text\rangle$ から、$\langle \texttt{rmauth}, d, ($任意の $\langle AbsoluteDate\rangle)\rangle \in \langle Offer\rangle$ であるような要素 o を取り除く。
3. $o \Rightarrow \langle AbsoluteDate\rangle <= now()$ ならば、outoflimit を返して終了。

4. $store(ud)$ を実行する。
5. $delete(u, \mathtt{auth} : u)$ を実行する。

$denyrmauth(u : UserID)$: $[u]$ 自身が自分を消そうとする Offer を取り消す。

1. $ud = retrieve(\mathtt{auth} : u)$ によって自分の認証文書を取る。
2. $ud \Rightarrow \langle Text\rangle$ から、$\langle \mathtt{rmauth}, \mathsf{null} \times$ (任意の $\langle AbsoluteDate\rangle$)$\rangle \in \langle Offer\rangle$ であるような要素を取り除く。
3. $store(ud)$ を実行し、その返り値を返す。

次の2つは、モデル上は重要ではないが、実用上は必要である。

$help \rightarrow \langle Message\rangle$: コマンドの一覧を返す。

$whoami \rightarrow \langle UserID\rangle$: いまログインしている利用者の識別子を返す。

$bye()$: 接続を切断する。

6.4.2　内部で使用するメソッド

$verifyLogin(u : \langle UserID\rangle, pw : \langle Passwd\rangle) \rightarrow \langle boolean\rangle$ $[u]$ が入力したパスワード pw が適切かどうかを検証する。

$getAbsoluteDate(date : \langle Date\rangle, date2 : \langle AbsoluteDate\rangle)$ $date \in \langle RelativeDate\rangle$ ならば、$date2$ を用いて $date$ を絶対時刻 (返り値) に変換する。

$getDomainRule(u : \langle UserID\rangle, c : \langle CtrlDoc\rangle)$ u と現在時刻に適合したルールを c から取り出す。

1. $c \Rightarrow \langle Text\rangle$ 内の各々の $tr \in \langle TimeRule\rangle$ と $now()$ とを比較し、$tr \Rightarrow \langle Date\rangle < now()$ が初めて成立する最も遅い tr を選び出す。
2. 各々の $dr = tr \Rightarrow \langle DomainName\rangle$ について、dr が u に前方一致する最も先頭に近い dr を選択し、返り値とする。

$filter(u : \langle UserName\rangle, d : \langle Doc\rangle, c : \langle CtrlDoc\rangle) \rightarrow \langle Doc\rangle$ 文書 d のうち $[u]$ に見せることが可能な部分を c によって判断して返す。

1. $doc = [d]$ と $[d]$ の制御文書 c を文書層から取得する。
2. $dr \in \langle DomainRule\rangle = getDomainRule(u, c)$ によって適切なルールを得る。
3. $dr \Rightarrow \langle Permission\rangle = \mathtt{meta}$ ならば、$d \Rightarrow \langle Text\rangle$ を forbidden に置き換える。
4. マスクされた d が認証文書であり、かつ $\mathtt{auth} : u \neq authLink(d)$ である場合、d 中の $\langle Passwd\rangle$ 要素を消去する。
5. 置換済みの d を返す。

6.5　エージェント

エージェントは、解釈層内に存在する自立的オブジェクトで、一定の周期でエージェントが到達可能な範囲内にある全ての文書をポーリングし、あらゆる利用者に代わって以下のような作業を

行う。

各々の文書を d とした場合、行う作業は以下の通り。

1. 削除許可が出ている文書について削除の操作を行う。
   (a) 解釈層内の $delete(authLink(d), d)$ を実行する。
2. 返答期限の過ぎた $\langle Offer \rangle$ があった場合、出しっ放しになっている $\langle Offer \rangle$ を消去し、`chown` あるいは `rmauth` 発行前の状態に戻す。
   (a) $type(d) = sfauth$ であった場合、d 内にある全ての $\langle Offer \rangle$ 要素を消去する。
3. 万一リンク構造の壊れた文書が発見された場合 (リンク先の文書などが消滅しているなど)、適切なリンクを張り直す。
   (a) $exists(ctrlLink(d)) = \mathsf{false}$ ならば、$ctrlLink(d)$ を `ctrl` : `root` : `permanent` にする。
   (b) $exists(authLink(d)) = \mathsf{false}$ ならば、$authLink(d)$ を `auth` : `root` にする。

第7章　文書の表現形式と初期文書

7.1　文書の表現形式

本節では、POT 文書によって定められるメタデータの形式について定める。しかし、メタデータに対するデータそのものである、文書のテキスト部分の書式については、何も規定しないことについて注意して頂きたい。

POT 文書はネットワークを通じて交換されることを前提としているから、バイト列として定着させる時の形式を定式化することが重要になる。そこで、POT モデルでは、媒体に記されたり外部を流れたりする際の POT 文書の表現形式として、XML(eXtensible Markup Language)[11] を用いる。

XML は、SGML[7] に類似ながら、しかし SGML よりも大幅に単純な文法で、電子情報をマークアップする規則を定める言語であり、電子情報を構造化する有力な手段としてその将来が嘱望されている。

XML を用いる利点は、本システムが要求するメタデータの構造を簡潔にバイト列の集合として表現できる点である。

また、XML はプレーンテキストの形で表現できる。この特徴は、万一システムの実装が機能しなくなったときにも、いざとなれば直接人間がデータを「読む」ことによって記述されているデータの内容を理解することを可能ならしめる。

制約としては、XML 文書に使用できる文字は制限されているので、バイナリデータを POT 文書の形で交換するときには、適切なエンコーディングによって、XML で使用可能な文字集合の範囲内にデータを写像する必要が生じることがある。

表 7.1 に、XML による文書の表現の一例を掲げる。

この例において、第 4.1 節で解説した文書要素と XML タグとの対応は、表 7.2 の通りである。

なお、表中では型 $\langle A\rangle$ のオブジェクトが要素として型 $\langle B\rangle$ のものを含むとき、その値の集合を「$\langle A\rangle \Rightarrow \langle B\rangle$」と表現している[1]。

なお、書式の詳細については、関連する DTD を付録 B に付したので、そちらを参照されたい。

7.2　初期文書

文書の定義から明らかなように、1 つの文書を登録するに先立っては、メタデータとなる文書へのリンクを張るために、以下の文書が存在しなければならない。

- 仕様書 1

---

[1] この記法は今後も用いる。

```
<?xml version="1.0"?>
<doc version="0.9" id="root:a">
  <logs>
    <logitem id="create" date="2000-01-07T02:42:58"/>
    <logitem id="auth:root" date="2000-01-07T02:42:58"/>
    <logitem id="ctrl:root:permanent" date="2000-01-07T02:42:58"/>
    <logitem id="spec:root:plaintext" date="2000-01-07T02:42:58"/>
    <logitem id="ctrl:root:foo" date="2000-01-07T07:49:19"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:foo"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>This is a sample of POT Document.
</text>
</doc>
```

表 7.1: XML で記した文書の一例

要素	タグ
$\langle Doc \rangle$	`<doc>`
$\langle Doc \rangle \Rightarrow \langle Text \rangle$	`<text>`
$\langle Doc \rangle \Rightarrow \langle Logs \rangle$	`<logs>`
$\langle Doc \rangle \Rightarrow \langle Logs \rangle \Rightarrow \langle LogItem \rangle$	`<logitem>`
$\langle Doc \rangle \Rightarrow \langle Links \rangle$	`<links>`
$\langle Doc \rangle \Rightarrow \langle Links \rangle \Rightarrow \langle DocID \rangle$	`<link>`の `id` 属性
$\langle Doc \rangle \Rightarrow \langle DocID \rangle$	`<doc>`の `id` 属性
……	……

表 7.2: POT 文書の要素と XML による表現との対応

- 認証文書 1
- 制御文書 1 以上

そこで、管理文書の設定を円滑に行うため、以下に示す識別子を持った文書を、初期状態の時点で用意する。

- `auth:root` : 全ての組織の頂点に位置する特別な管理者 `root` の認証文書。
- `ctrl:root:permanent` : 全ての文書の制御文書へのリンクを順に辿っていくと必ず到達する、特別な制御文書。全ての利用者に文書の全ての要素を永久に公開することを保証する。
- `spec:root:plaintext` : 全ての文書の仕様書へのリンクを順に辿っていくと必ず到達する、特別な仕様書。プレーンテキストの構造について記している[2]。
- `spec:root:xml` : 本システムで広く用いている XML の書式を記した文書。その本文の内容は、[11] をプレーンテキストで書き直したものに等しい。
- `spec:root:authspec` : 認証文書の構成について記した文書。全ての認証文書は仕様書にこれを指定する。
- `spec:root:ctrlspec` : 制御文書の構成について記した文書。全ての制御文書は仕様書にこれを指定する。

また、これらの文書の各々のリンク先は、図 7.1 の通りであるものとする。

なお、付録 A に、これらの文書の XML 表現を掲載した。

---

[2] つまりこのシステムは、ASCII 文字によるプレーンテキストが最も信用できる表現形式であるという立場を表明していることになる。

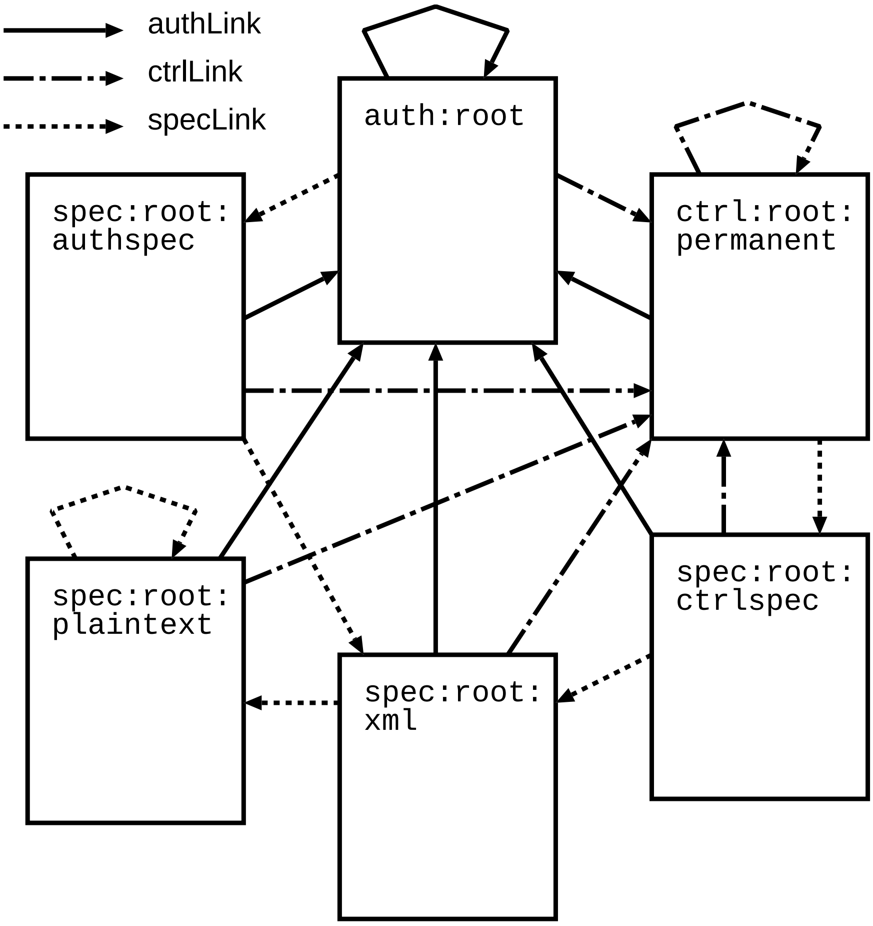


図 7.1: 初期文書間のリンク関係

第8章　下位層の実装のための検討

本章では、物理層や分散層についての利用可能な要素技術や実装の際に注意すべき事項について述べる。

8.1　物理層

記憶媒体の物理的な寿命を超えて情報を永続させるには、適当な時間間隔で古い媒体から新しい媒体に内容を移し、ファイル識別子と媒体上の所在の関係を変更するという方法によることができる。この操作によるパスの変更は、定義上、物理層の外側から見えてはならない。

また、複写先の媒体をその都度新しい種類のものにすることによって、媒体の技術的進歩にも漸進的な対応が可能になる。

複製、定期的なバックアップ、RAID[16] などの技術も応用できよう。

単独の媒体の寿命を延ばす研究としては、HD-ROM[17] がある。HD-ROM は、媒体に物理的に微細な傷をつけることで情報を記憶するというもので、磁気を使用しないため、耐用年数が非常に長く取れるとのことである。

8.2　分散層

8.2.1　既存のツール

同一の内容のファイルを広域的に共有するための規則あるいはツールとしては、以下のようなものがある。

URN[14] IETF にて審議中。URL の拡張として、するものである。現在はまだ仕様の詳細について議論が進められている段階であるが、永続的命名の標準になることが期待されている。

PURL[21] HTTP を利用したサービスで、特定のサイトに属する URL から実際の URL をリンクすることで実際の URL の移転の影響を吸収するというものである。将来的には URN ベースに移行することも可能だという。

DOI[8] これも、永続的な名前を可変な URL に結びつけるもので、既にサーバが稼働中である。

Handle[19][18] DOI の実装に用いられているサブシステム。ハンドルと呼ばれる識別子を複数の URL などにリンクする仕組みを提供するほか、ハンドルを用いた管理者の管理など、POT システムと共通する特徴が多い。

なお、完全に公開されているとは限らない文書をサイト間で交換する際、通信の安全性を確保するには、「分散層内用サイト間認証」が必要になるであろう。これには、Netscape 社が開発した SSL[9] やその後継プロトコルである TLS[2] が応用できる。これらは、OSI の 7 層モデルトランスポート層とネットワーク層の間に、情報暗号化のための層序を挿入して回線を安全にするというものである。

8.2.2　文書の不必要な拡散への注意

制御文書で定めた一定の範囲の外側に対しては秘密である筈の文書が、分散層による頒布の結果、外部に洩れる危険が存在するので、文書に許された公開範囲を解釈層で読み取り、分散層に命じて頒布範囲を制限させる工夫が実装上必要になるであろう。

第9章　文書関係モデルなどの検証

第4 ~ 7章で与えたモデルは、第2.2節に示した要件を最低限満たしていると考えられるが、これのみでは前述の全ての要求を満たすことができないかもしれない。ゆえに本章では、文書関係モデル (第4.2節) を中心に、POTモデルに更に加えるべき制約を検証しつつ定める。

9.1　所有者自身による書き換え

本モデルにおいては、文書の少なからぬ部分を書き換えることができる。文書自身の書き換えは、エージェントなどによる例外を除いては所有者以外には不可能である。しかし、所有者自身によるものでも、書き換えを行うとリンクの整合性などが損なわれるおそれがあるので、不都合な部分を検証し、それについては実装で禁止するようにすべきであろう。

ここでは、管理者自らが `chtext` コマンドや `delete` コマンドによって文書を書き換えることの是非を検討する。

9.1.1　通常文書

本文 これの書き換えを認めると、システム本来の大目的を満たすことができなくなるので、不可。

認証文書へのリンク 他の管理者への文書の受渡しのためには変更できることが必要。可。

制御文書へのリンク 管理スケジュールを変更する必要が生じたときのために、可。

仕様書へのリンク 本文が変更されないならば文書の仕様も普遍なので不可。

削除について: 原則として可 (削除可能である要件は後述 (以下同))。

9.1.2　仕様書

本文 仕様が変更されることはあってはならないので、不可。

認証文書へのリンク 通常文書と同じ理由で可。

制御文書へのリンク ある仕様に則った文書が存在しなくなったとしても、仕様そのものが消滅するわけではない。ゆえに、`spec:permanent` 以外の制御文書を指定し直すことは不合理であるから不可。

仕様書へのリンク 通常文書と同じ理由で不可。

type	text	*authLink*	*ctrllink*	*speclink*
ordn	×	○	○	×
spec	×	○	×	×
auth	×	○	△	×
ctrl	○	○	△	○

表 9.1: 各種文書の各要素の変更の是非

9.1.3　認証文書

本文 パスワードなどを変更するために必要。可。

認証文書へのリンク 組織構造の一貫性を維持できなくなるので、不可。

制御文書へのリンク 条件つきで可 (条件は次節で検討する)。

仕様書へのリンク 本文の内容を変えても、フォーマットに変わりはないので、変更の必要がない。不可。

9.1.4　制御文書

本文 管理スケジュールの変更のために必要。可。

認証文書へのリンク 単純に所有者を変えることを意味するので、可。

制御文書へのリンク 条件つきで可 (条件は次節で検討する)。

仕様書へのリンク 認証文書と同じ理由により、可。

なお、制御文書の削除は、これを禁止すべきである。それは「管理計画を放棄すること」を意味し、そのようなことは認められるべきでないからである。
これらの結果ををまとめてると、表 9.1 のようになる。なお、「△」は、「後述の条件つきで可」を表している。
これらの変更の是非は、以下において確定の事実として扱う。

9.2　所有者以外による書き換え

次に、文書 D の所有者以外の人物が文書 D に関係する情報を操作することによって、所有者のあずかり知らぬ間に文書の内容が改竄されてしまう危険について検討する。

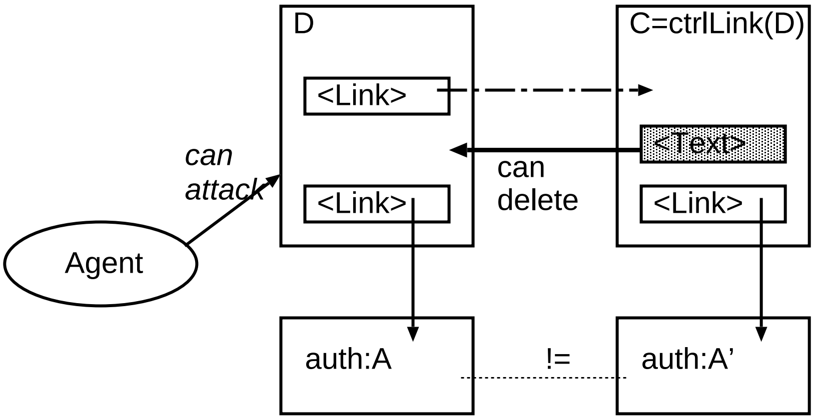


図 9.1: 削除の危険が生じる状況

9.2.1　リンク先の文書が他人に書き換えられる場合

文書 D の管理者 A と D の制御文書 C の管理者 A' が異なる場合、A の意志に関わらず、A' が勝手に C の内容を置き換える危険がある。しかし、D の内容の変更は A にのみ許されているので、次節に見る削除の場合を除き、D そのものが破壊される心配はない。

9.2.2　エージェントによる書き換え

第 6.5 節において導入したエージェントによって文書の一部が書き換えられるケースは、以下の 2 通りである。

1. $chown$ コマンドで指定する文書取得期限が切れることによって、受取人の認証文書の Offer 要素が消去される。
2. 文書 D の管理者 A と D の制御文書の C の管理者 A' が異なり、かつ C が D に削除の許可を出している場合 (図 9.1)、A のあずかり知らぬうちに A' の指示によってエージェントが C を削除してしまう。

つまり、今のままのモデルでは、エージェントを経由して文書を削除される危険が存在する。

また、2. が起こってしまった場合、エージェントの巡回が来るまでの間、削除された文書にリンクがある文書は、文書構造が壊れてしまうことになる。

これを回避するためには、

- A と A' が等しくない文書の一覧を A が容易に知り得るようにする
- A と A' が等しくない文書を直接削除する代わりに、エージェントは $chown$ や $rmauth$ コマンドで出現するような $\langle Offer\rangle$ 要素を A の認証文書に加え、注意を促す

のような対応をとる必要があろう。

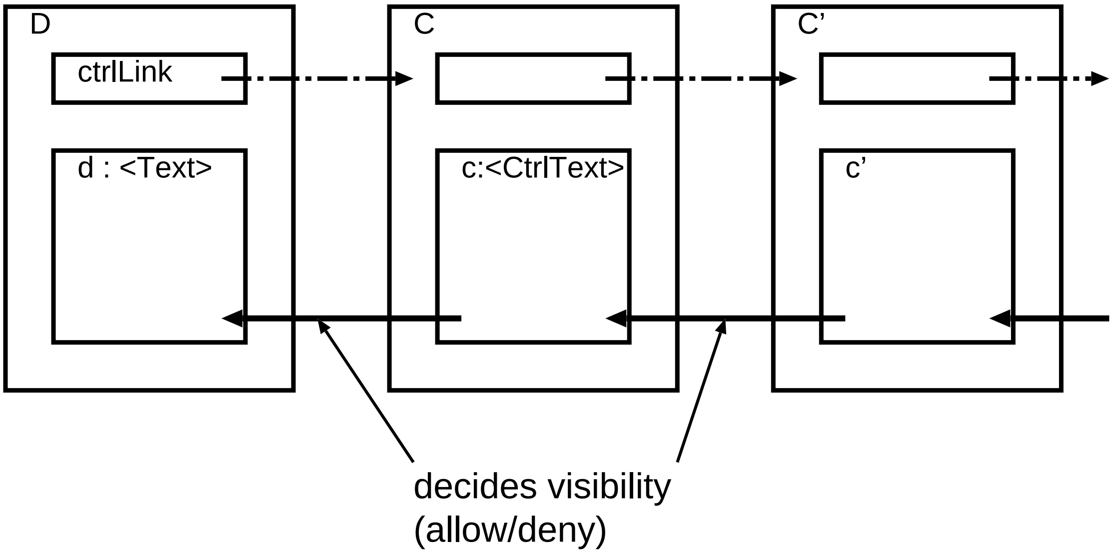


図 9.2: d と c の「見え方」について

以上のような対応をとらなければ、どうしても文書を自分で削除の危険から守りたい場合には、利用者は自ら直接の制御文書の所有者になる必要がある。

9.3　公開/非公開によって生じ得る事態

本モデルでは、文書の本文について、時刻毎に異なった 公開/非公開 の設定をすることが可能である。

本節では、これらの設定の組み合わせによって、閲覧を禁止したはずの文書が見えてしまうことがないかどうかを検証する。

制御文書の本文の主な存在理由は、文書 (の部分ごと) に、時間別の公開範囲を指定することであった。しかし、文書の一部の閲覧が許可されているのに他の部分が許可されていない場合、様々な不具合の起こることが予想される。

任意の文書を D、D の制御文書を C とし、D, C の本文をそれぞれ d, c とおく。このとき、任意の利用者にとって、d と c の「見え方」には、以下の 4 通りの可能性があることになる (図 9.2 参照)。

1. d も c も閲覧可能
2. d も c も閲覧不可
3. c は閲覧可能だが、d は不可
4. d は閲覧可能だが、c は不可

1. の場合は運用上全く問題がなく、2. も両方とも見えないのだから、D の閲覧に関しては矛盾が生じない。3. は、c によって d が見えなくなっているに過ぎないので、矛盾のある状態ではない。

しかし 4. の場合、d のための制御情報を c は取得できず、そのため、d の公開/非公開が決定できなくなってしまう。この状態は、C の制御文書 C' の状態次第によって起こり得る。しかも、制御文書へのリンクは `ctrl:root:permanent` に至るまでの順列をなすので、上の問題を敷衍すると、連鎖のいかなる段階においても、$C^{(n)}$ の本文が閲覧できるのに $C^{(n+1)}$ の本文が閲覧できない状態は不都合であるということになる。そこで、この問題を回避するための条件について、次の節で考察する。

9.3.1　制御文書の大小について

次節の議論に用いる概念として、制御文書の (通時的) 公開範囲、及びそれらの「広い/狭い」について定義を行う。

制御文書 C の時刻 t に対する公開範囲 ($range(C,t) \rightarrow \langle User \rangle *$) を、

$$range(C,t) \equiv \bigcup_{u \in U} \{\forall u \forall d(permit(u,d,getAbsoluteDate(t,creationDate(d)) = \mathsf{true}),$$

即ち、「C を参照している任意の文書 d について、d の作成時刻から t だけ経過した時刻において、d の本文を見ることのできる利用者の集合」と定義する。ここで U は全ての利用者の集合、$permit(u,d,t) \rightarrow \langle boolean \rangle$ は、利用者 u が時刻 t に文書 d の本文を読めるか否かを返す関数である。また、$getAbsoluteDate$ と $creationDate$ の定義は、第 6.3 節を参照のこと。

次に、制御文書 C の通時的公開範囲 ($range(C)$) を、

$$range(C) \equiv \bigcup_{[0,\infty)} range(C,t)dt$$

と定義する。

次に、制御文書 C と C' の大小関係について、あらゆる時刻 t に対して $range(C,t) \subset range(C',t)$ であるとき $C < C'$ と、$range(C,t) = range(C',t)$ であるとき $C = C'$ であると定義する。

以上の定義より、range(C) の大小関係には推移律が成立する (証明は自明なので略)。

9.3.2　$range(C)$ に関する命題

時刻ごとの公開範囲について

$t < t'$ならば、$range(C,t) > range(C,t')$ とすることには意味がない。なぜなら、公開してしまったものは、向こうが忘れてくれない限り、取り返しがつかないからである。即ち、

$$t < t' \text{ならば } range(C,t) \leq range(C,t')$$

とすることを強く推奨する。

第 9.3 節中 4. のような矛盾を生じさせないためには、任意の文書 d とその制御文書 $C = ctrl(d)$、さらに $ctrl(d)$ の制御文書 $C' = ctrl(C)$ において、$C < C'$ でなければならない。

さらに、C' を含むあらゆる文書が制御文書を持つことを考慮に入れると、C から $[\mathtt{ctrl:root:permanent}]$ に至るまでの列において、

$$C < C' < ... < [\mathtt{ctrl:root:permanent}](*)$$

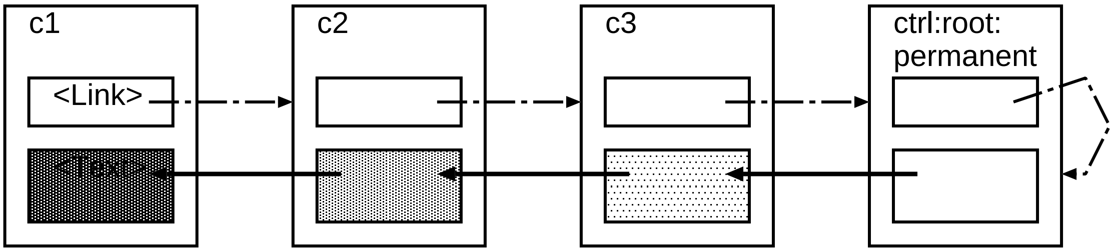


図 9.3: 制御文書の連鎖のイメージ図

とならなければならない。この様子をイメージ化したのが図 9.3 である。この図では、C の大きさをテキスト部分の色の明るさで表している。

9.4　リンクの循環

同一種類の文書への全てのリンクへの連鎖は、リンクに一切変更が加えられないと仮定した場合、初期文書に向かう単純な木構造をなし、初期文書に存在するものを除いて、ループは出現しない。

証明は単純で、

1. 初期文書 (認証文書ならば `auth:root`、制御文書ならば `ctrl:root:permanent`、仕様書ならば `spec:root:plaintext`) はノードが 1 つだけなので木である
2. 木 x のいずれかの葉から x に含まれない新しい要素に辺を 1 つ伸ばしたものもまた木である

の 2 点から帰納的に証明できる。

ところで、リンクの張り変えによって循環したリンクが出現すると、解釈層内の $filter$ メソッドが無限ループに陥るなど、本システムによるリソースの管理に、あるいはメソッドの実行に混乱が生じるおそれがある。特に文書の公開範囲の決定には制御文書が再帰的に用いられるので、制御文書から制御文書へのリンクは要注意である。そこで、循環リンク (図 9.4 参照) は発生するか否か、発生するならばその条件は何かを、各種類のリンクについて検証する。

9.4.1　自己ループ

リンクが自分自身を指し示す長さ 1 の循環は、実装上も検出はさほど困難ではないので (調べる文書は 1 つだけでよい)、これを認めることにする。初期状態では、`auth:root`, `spec:root:plaintext`, `ctrl:root:permanent` がこれに該当する。特に `spec:root:plaintext` は、仕様書の定義からして自己ループを作らざるを得ない。よって、以下では、長さ 2 以上のループを問題とする。

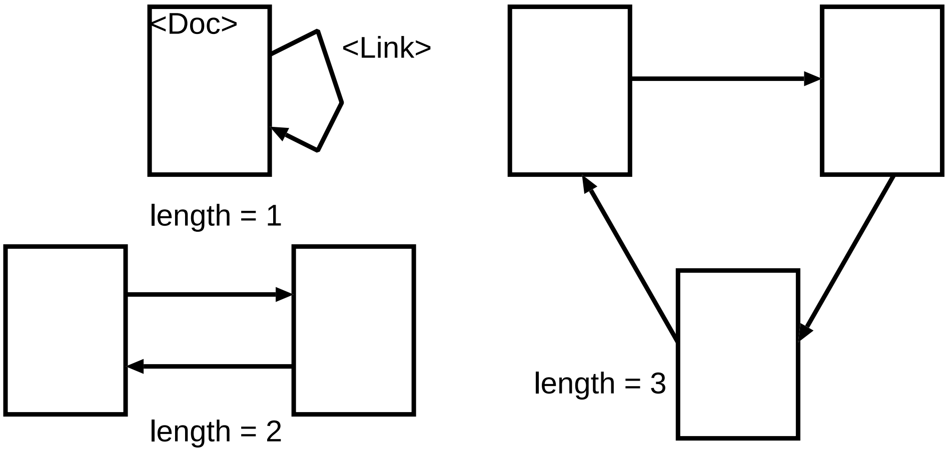


図9.4: 長さ1・2・3の循環リンクの見本

9.4.2　長さ2以上のループ

既に見たように、仕様書が含む仕様書へのリンクや認証文書が含む認証文書へのリンクは変更を禁止したので、ループの問題は生じない。

残る問題は制御文書へのリンクの連鎖であるが、前節の(∗)に基づいて、公開範囲が「狭まる」ようなリンク $range(C_1) > range(C_2)$ である2つの制御文書 C_1 と C_2 について C_1 から C_2 へのリンクを禁止するようにすれば、制御文書の読み出しの連鎖は、無事`ctrl:root:permanent`まで行きつくことができるようになるので、ループの危険は生じない。

9.5　Spoofingの防止

本モデルでは、下位の層から持ってきた情報を最終的に公開するか否かの判断を、解釈層という高いレベルの層で行っている。従って、公開してはいけない情報が公の回線の上を流れる危険が存在するので、spoofing(回線の盗聴)による被害を防ぐために、層同士の接続に気を遣う必要がある。

具体的には、以下の箇所(図9.5の点線部分)について、外部にさらされた回線を保護する措置が必要である。

- 分散層同士
- 文書層と分散層の間
- 解釈層と文書層の間
- 解釈層と外部の間

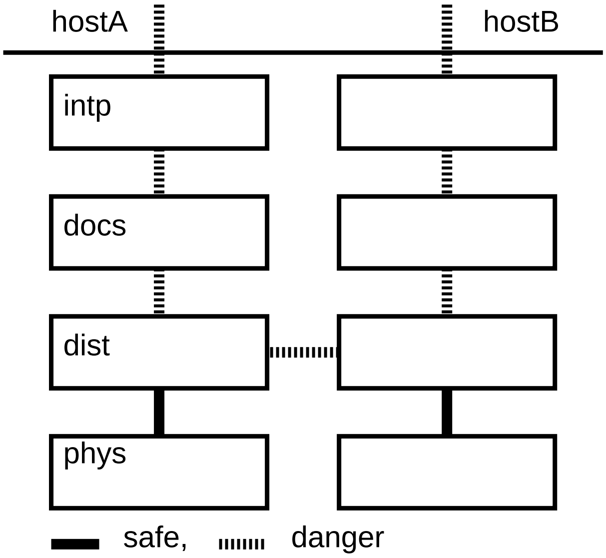


図 9.5: 安全な回線と危険な回線

第10章　ユースケース

本節には POT システムに対するいくつかの操作のシナリオを掲げ、システムの内部 (ただし文書層以上) でどのような操作が行われることになるかを示す。

10.1　ログイン

POT にアクセスする全ての者は、最初にログインを行う。この章の残りの節における操作は、全て本節の操作を経てから行われる。

1. 利用者は外部から解釈層にアクセスし、解釈層に対して login(u, pw) を発行する。
2. 解釈層サーバは利用者からパスワード pw と利用者識別子 u を受け取り、内部で $verifyLogin(u, pw)$ を実行する。
3. $verifyLogin(u, pw)$ の結果が true ならば、サーバは利用者を u と見なして接続を続ける。false ならば、サーバは利用者との接続を断つ。

10.2　利用者の登録

利用者 $[X]$ が新たに利用者 $[X : Y]$ を登録しようとする場合。

1. 利用者 $[X]$ は、$storeauth(X, Y, c \in \langle DocID\rangle, pw \in \langle Passwd\rangle)$ によって新ユーザとそのパスワードを登録する。
2. 利用者 $[X : Y]$ は、直ちに $chtext(X : Y, \texttt{auth} : X : Y, \mathsf{null}, pw' \in \langle Passwd\rangle)$ によってパスワードを入れ換えることが期待される。

10.3　通常文書/仕様書/制御文書の登録

1. 利用者は解釈層に対して $store(type, u, id, ctrl, spec, t)$ を発行する。
2. 解釈層は第 6.4.1 節の $store$ の項にある通りの動作を行う。
3. 上を承けて、文書層も第 6.3.1 節の $store$ の項にある通りの動作を行う。
4. 解釈層は文書層からのステータスを受け取り、利用者に返す。

10.4　通常文書/仕様書/制御文書の閲覧

1. 利用者は解釈層に対して $retrieve(u, id)$ を発行する。
2. 解釈層は (以下同じ) 文書層に $retrieve(id)$ を発行し、$[id]$ を入手する。
3. $[id]$ から $c := ctrlLink([id])$ を取り出し、再び文書層にアクセスして $[c]$ を取り出す。
4. $filter(u, d, c)$ で d に然るべきマスクをかけ、外部に渡す。

10.5　本文の書き換えによる制御文書の置き換え

1. 利用者は解釈層に対して $chtext(u, id, t)$ を発行する。ここで id は、制御文書の識別子である。
2. 解釈層は第 6.4.1 節の $chtext$ の項にある通りの動作を行う。
3. $chtext$ の返り値をを利用者に返す。

10.6　リンクの変更による制御文書の置き換え

1. 利用者は解釈層に対して $chctrl(u, id, t)$ を発行する。ここで id は、通常文書の識別子である。
2. 解釈層は第 6.4.1 節の $chctrl$ の項にある通りの動作を行う。
3. $chctrl$ の返り値をを利用者に返す。

10.7　管理者の交替

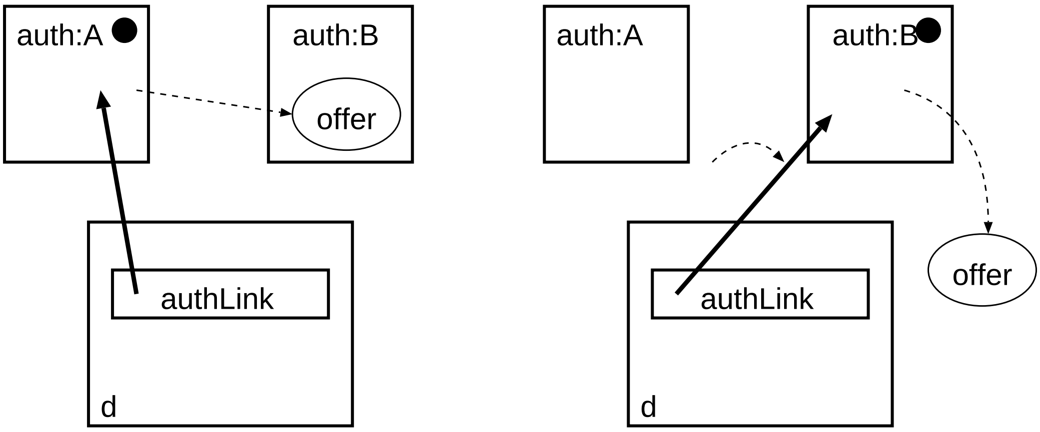
利用者 $[A]$ が自らの所有する文書 d を別の利用者 $[B]$ に譲るときの操作を示す。図 10.1 にイメージ図があるので参照されたい。

1. $[A]$ が解釈層に対して $chown(d \times B \times limit| \in \langle\rangle)$ を発行すると、[`auth` :B] に $\langle Offer\rangle$ 要素が追加される。
2. $[B]$ はそれを読み、$getown(d)$ か $denyown(d)$ を実行する。
3. 2. で $[B]$ が $denyown(d)$ を実行したときは、[`auth` :B] の $\langle Offer\rangle$ 要素が消えて終わる。
4. 2. で $[B]$ が $getown(d)$ を実行したときは、[`auth` :B] の $\langle Offer\rangle$ 要素が消去されるのに加えて、$ctrlLink(d)$ が書き換えられる。
5. $[B]$ が何らかの行動を起こす前に期限が来てしまった場合は、$[B]$ が $denyown$ を発行した場合と同じ結果になる。

ここで、$[A]$ が $[B]$ 以外の人物 $[B'], [B''], \ldots$ に対しても文書 $[d]$ に関する $chown$ を発行していた場合、$[B]$ が $getown$ によって $[d]$ を得ても、$[B']$ などが後から $[d]$ を掠めとることは可能である。したがって、$[A]$ は、複数の人物に対して $chown$ を発行することには慎重にならなければならない。

10.8　利用者の抹消

利用者 $[A]$ が利用者 $[A : B]$ を抹消するときの動きは以下のようになる。



1. chown(A, B, d)　　2. getown(B, d)

図 10.1: $[A]$ から $[B]$ への文書 d の譲渡

1. $[A]$ が解釈層に対して $rmauth(d \times B \times limit \in \langle Date \rangle)$ を発行すると、[`auth` :B] に $\langle Offer \rangle$ 要素が追加される。
2. $[B]$ はログインの際にそれを読み、$getrmauth(d)$ か $denyrmauth(d)$ を実行する。
3. 以下は解釈層レベルでの話になる。2. で $[B]$ が $denyrmauth(d)$ を実行したときは、[`auth` :B] の $\langle Offer \rangle$ 要素が消えて終わる。
4. 2. で $[B]$ が $getrmauth(d)$ を実行し、かつ [`auth` :B] に削除許可が出ている場合は、$\langle Offer \rangle$ 要素が消えるのに加えて、$delete$([`auth`:B]) が発行される。
5. $[B]$ が何らかの行動を起こす前に期限が来てしまった場合は、$[B]$ が $denyrmauth$ を発行した場合と同じ結果になる。

第11章　実装・実験

11.1　プロトタイプ

第4~7章で詳述したモデルの動作の可能性を調べる実験のため、第6章に記したAPIのうち、次に記した以外の部分をカバーするプロトタイプの実装を作成した。

- 分散層の分散をさせる部分
- 第9.2.2節で述べた、文書を削除されてしまうことへの対策
- 第9.3.1節で述べた、$range(C \in \langle CtrlDoc \rangle)$の大小を判断する部分
- 解釈層での$chtext$における、入力された$\langle Text \rangle$要素の整合性の判定
- $verifyLogin$における「凝った」認証方法。本実装では、単純に文字列を比較する以上のことは行っていない。

実装に用いた言語は、Java(JDK1.1.7)である。加えて、XMLを扱うために、IBM東京基礎研究所で開発された、XML for Java 1.1.14[12]を使用した。また、使用したプラットフォームは、主にLinux2.0.36である。

なお、ソースコードを付録Cに掲載したので参照のこと。

11.1.1　実装の特徴

本プロトタイプは以下のような特徴を持つ。

- 層ごとに独立したパッケージ構成
- TCP/IPソケットによるサーバ接続
- テキスト指向のインターフェース。これにより、telnetコマンドで直接サーバと対話することが可能になる。

11.2　実験

前節で説明した実装を用いて、第10章に記したユースケースが実際にうまく動作するか否かについて実験を行った。

前章の各々のユースケースについての実行結果(クライアント画面の再現)を以下に掲載する。

11.2.1　ログイン

以下は、利用者 `root` がログインの手続きを行ったところである。

```
<arcadia:/home/users/itogawa/research/code/pot/user>telnet localhost 19999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
login: root
password: toor
Welcome, root.

.
POT> whoami
! root
POT> bye
bye
Connection closed by foreign host.
```

11.2.2　利用者の登録

以下は、初期状態での唯一の利用者 `root` が `root:foo` を登録するところである。

```
POT> storeauth foo ctrl:root:permanent
password:foo
! OK
POT> get auth:root:foo
<?xml version="1.0"?>
<doc version="0.9" id="auth:root:foo">
  <logs>
    <logitem id="create" date="2000-01-17T22:54:26"/>
    <logitem id="auth:root" date="2000-01-17T22:54:26"/>
    <logitem id="ctrl:root:permanent" date="2000-01-17T22:54:26"/>
    <logitem id="spec:root:authspec" date="2000-01-17T22:54:26"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:authspec"/>
  </links>
  <text>
    <passwd>foo</passwd>
    <announce>
      <message>Hajimemashite!</message>
    </announce>
  </text>
</doc>
.
```

11.2.3　パスワードの変更

次に、先ほど登録された利用者 `root:foo` が自分でパスワードを変更する。

```
POT> chtext auth:root:foo
<text>
<passwd>FOO</passwd>
</text>
.
! OK
```

変更の結果を確認する。

```
POT> get auth:root:foo
<?xml version="1.0"?>
<doc version="0.9" id="auth:root:foo">
  <logs>
    <logitem id="create" date="2000-01-17T22:54:26"/>
    <logitem id="auth:root" date="2000-01-17T22:54:26"/>
    <logitem id="ctrl:root:permanent" date="2000-01-17T22:54:26"/>
    <logitem id="spec:root:authspec" date="2000-01-17T22:54:26"/>
    <logitem id="text" date="2000-01-17T22:55:18">
      <announce>
        <message>Hajimemashite!</message>
      </announce>
    </logitem>
    <logitem id="text" date="2000-01-17T22:55:59">
      <announce>
        <message/>
      </announce>
    </logitem>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:authspec"/>
  </links>
  <text>
    <passwd>FOO</passwd>
    <announce>
      <message/>
    </announce>
  </text>
</doc>
.
```

announce タグが自動的に付加されているのにも注意。以下は、権限のない者がパスワードを書き換えようとして失敗した例である。

```
Welcome, root:foo.

.
POT> chtext auth:root
<text>
<passwd>cracking</passwd>
</text>
.
! not the owner.
```

11.2.4　制御文書/通常文書の登録

以下の4つの文書を登録してみる。

- `ctrl:root:foo:c1`
- `ctrl:root:foo:c2`
- `ordn:root:foo:o1`
- `ordn:root:foo:o2`

登録状況は以下の通り。

```
Welcome, root:foo.

.
POT> store ctrl c1 ctrl:root:permanent
<text>
    <time value="A-0">
            <range value="root">meta</range>
            <range value="root:foo">all</range>
    </time>
    <time value="A-0-0-0-0-10">
            <range value="root">text</range>
    </time>
</text>
.
! OK
POT> store ctrl c2 ctrl:root:foo:c1
<text>
    <time value="A-0">
            <range value="root">meta</range>
            <range value="root:foo">all</range>
    </time>
    <time value="A-0-0-0-0-5">
            <range value="root">delete</range>
    </time>
</text>
.
! OK
POT> store ordn o1 ctrl:root:foo:c1 spec:root:plaintext
this is o1.
.
! OK
POT> store ordn o2 ctrl:root:c2 spec:root:plaintext
this is o2.
.
end{verbatim}}

\subsection{通常文書の閲覧}
登録から10分以内に{\tt ordn:root:foo:o1}を{\tt root}の資格で
見に行くと下のようになる。
{\small\begin{verbatim}
<?xml version="1.0"?>
```

```
<doc version="0.9" id="ordn:root:foo:o1">
(中略)
  <text>forbidden</text>
</doc>
.
```

下は、登録から5分以上経った後に`ordn:root:foo:o2`を見に行ったところである。

```
POT> get ordn:root:foo:o2
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o2" status="deleted">
  <logs>
    <logitem id="create" date="2000-01-17T23:32:42"/>
    <logitem id="auth:root:foo" date="2000-01-17T23:32:42"/>
    <logitem id="ctrl:root:foo:c2" date="2000-01-17T23:32:42"/>
    <logitem id="spec:root:plaintext" date="2000-01-17T23:32:42"/>
    <logitem id="forbidden" date="2000-01-17T23:45:21"/>
    <logitem id="forbidden" date="2000-01-17T23:45:21"/>
    <logitem id="text" date="2000-01-17T23:45:21">this is o2.
</logitem>
  </logs>
  <links>deleted</links>
  <text>deleted</text>
</doc>
.
```

上から、エージェントが動いていることが明らかである。ログに`forbidden`という文字列が入ってしまっているのは、何らかのバグであると思われる。

11.2.5　本文の書き換えによる制御文書の置き換え

```
POT> whoami
! root
POT> get ordn:root:foo:o3
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o3">
  <logs>
    <logitem id="create" date="2000-01-18T00:06:52"/>
    <logitem id="auth:root:foo" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c3" date="2000-01-18T00:06:52"/>
    <logitem id="spec:root:plaintext" date="2000-01-18T00:06:52"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:foo:c3"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>forbidden</text>
</doc>
.
```

この状態から、

```
POT> whoami
```

```
! root:foo
POT> chtext ctrl:root:foo:c3
<text>
    <time value="A-0">
            <range value="root">all</range>
    </time>
</text>
.
! OK
```

とすると、下のように本文が見えるようになる。

```
POT> whoami
! root
POT> get ordn:root:foo:o3
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o3">
  <logs>
    <logitem id="create" date="2000-01-18T00:06:52"/>
    <logitem id="auth:root:foo" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c3" date="2000-01-18T00:06:52"/>
    <logitem id="spec:root:plaintext" date="2000-01-18T00:06:52"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:foo:c3"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is o3.
</text>
</doc>
.
```

11.2.6　リンクの変更による制御文書の置き換え

```
POT> whoami
! root:foo
POT> store ctrl c4 ctrl:root:permanent
<text>
    <time value="A-0">
            <range value="root">meta</range>
    </time>
    <time value="A-0-0-0-0-10">
            <range value="root">text</range>
    </time>
</text>
.
! OK
POT> chctrl ordn:root:foo:o3 ctrl:root:foo:c4
! OK
```

と置き換えてから改めて `ordn:root:foo:o3` を読み出してみたら以下のようになった。

```
Welcome, root.
```

```
.
POT> get ordn:root:foo:o3
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o3">
  <logs>
    <logitem id="create" date="2000-01-18T00:06:52"/>
    <logitem id="auth:root:foo" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c3" date="2000-01-18T00:06:52"/>
    <logitem id="spec:root:plaintext" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c4" date="2000-01-18T00:32:07"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:foo:c4"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is o3.
</text>
</doc>
.
```

以下は、権限のない者 (文書の所有者以外) が制御文書へのリンクを変更しようとして失敗した例である。

```
POT> get ordn:root:foo:o3
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o3">
  <logs>
    <logitem id="create" date="2000-01-18T00:06:52"/>
    <logitem id="auth:root:foo" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c3" date="2000-01-18T00:06:52"/>
    <logitem id="spec:root:plaintext" date="2000-01-18T00:06:52"/>
    <logitem id="ctrl:root:foo:c4" date="2000-01-18T00:32:07"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:foo:c4"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is o3.
</text>
</doc>
.
POT> whoami
! root
POT> chctrl ordn:root:foo:o3 ctrl:root:permanent
! not the owner.
```

11.2.7　管理者の交替

```
POT> whoami
! root
POT> chown auth:root:foo ordn:root:x
```

```
! OK

Welcome, root:foo.
<offer type="chown" id="ordn:root:x" limit="2000-01-18T01:11:12"/>
.
POT> getown ordn:root:x
! OK
POT> get ordn:root:x
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:x">
  <logs>
    <logitem id="create" date="2000-01-17T23:52:37"/>
    <logitem id="auth:root" date="2000-01-17T23:52:37"/>
    <logitem id="ctrl:root:permanent" date="2000-01-17T23:52:37"/>
    <logitem id="spec:root:plaintext" date="2000-01-17T23:52:37"/>
    <logitem id="auth:root:foo" date="2000-01-18T00:41:59"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is a document x.
</text>
</doc>
.
```

11.2.8　利用者の削除

削除のためにわざと`root:bar`という利用者を作ってみた。

```
POT> whoami
! root
POT> storeauth bar ctrl:root:permanent
password:bar
! OK
POT> rmauth root:bar
! OK
```

しかし、

```
Welcome, root:bar.
<offer type="rmauth" limit="2000-01-18T00:48:40"/>
.
POT> getrmauth
! forbidden
```

となった。原因は、auth:root:barの制御文書が削除許可を出していなかったからであった。

11.2.9　破綻検査

第9.3.1節の原則に反した制御文書をわざと作り、破綻することを確認し、さらにエージェントがそれを修復するところを確認する。むろんこれは$range$の判断を実装していれば起こり得ない

事態だが、未実装であるし、何よりも、エージェントの修復機能を試すことができるので、行うことにした。

```
Welcome, root:foo.


.
POT> store ctrl c6 ctrl:root:foo:c2
<text>
    <time value="A-0">
            <range value="root">all</range>
    </time>
</text>
.
! OK
POT> store ordn o6 ctrl:root:foo:c5 spec:root:plaintext
this is o6.
.
! OK
```

これで、c2 ← c6 ← o6 というリンクの連鎖ができ、かつ c6 が 5 分後に壊される手筈が整ったことになる。(5 分後)

```
Welcome, root.


.
POT> get ctrl:root:foo:c6
<?xml version="1.0"?>
<doc version="0.9" id="ctrl:root:foo:c6" status="deleted">
  <logs>
    <logitem id="create" date="2000-01-19T19:10:24"/>
    <logitem id="auth:root:foo" date="2000-01-19T19:10:24"/>
    <logitem id="ctrl:root:foo:c2" date="2000-01-19T19:10:24"/>
    <logitem id="spec:root:ctrlspec" date="2000-01-19T19:10:24"/>
    <logitem id="forbidden" date="2000-01-19T19:16:00"/>
    <logitem id="forbidden" date="2000-01-19T19:16:00"/>
    <logitem id="text" date="2000-01-19T19:16:00">
      <time value="A-0">
        <range value="root">all</range>
      </time>
    </logitem>
  </logs>
  <links>deleted</links>
  <text>deleted</text>
</doc>
.
POT> get ordn:root:foo:o6
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o6">
  <logs>
    <logitem id="create" date="2000-01-19T19:10:34"/>
    <logitem id="auth:root:foo" date="2000-01-19T19:10:34"/>
    <logitem id="ctrl:root:foo:c6" date="2000-01-19T19:10:34"/>
    <logitem id="spec:root:plaintext" date="2000-01-19T19:10:34"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
```

```
    <link id="ctrl:root:foo:c6"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is o6.
</text>
</doc>
.
```

読めてしまった。がこれは、解釈層の実装において、「$\langle Permission \rangle$ が不明なものは公開する」ということにしているからに他ならない。(エージェント再通過後)

```
Welcome, root.

.
POT> get ordn:root:foo:o6
<?xml version="1.0"?>
<doc version="0.9" id="ordn:root:foo:o6">
  <logs>
    <logitem id="create" date="2000-01-19T19:10:34"/>
    <logitem id="auth:root:foo" date="2000-01-19T19:10:34"/>
    <logitem id="ctrl:root:foo:c6" date="2000-01-19T19:10:34"/>
    <logitem id="spec:root:plaintext" date="2000-01-19T19:10:34"/>
    <logitem id="ctrl:root:permanent" date="2000-01-19T19:24:53"/>
  </logs>
  <links>
    <link id="auth:root:foo"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>this is o6.
</text>
</doc>
.
```

$ctrlLink$ が変更されていることに注目していただきたい。

第12章　結論

12.1　モデルは要件を満たしているか?

本節では、第2章で述べた要件を、POTモデルが満たしているかどうかについて検討する。

12.1.1　第2.1.8節での要件

まず、第2.1.7節で述べた3つの要件を満たしているかを検討しよう。

機密性(Confidentiality) パスワード認証、及び、認証文書の所有権移動を固定したことによる公開範囲の保証があるので、アクセスは保護されているといえる。

一貫性(Integrity) 第6章で見た通り、パスワード変更など、管理者のみに文書の変更を許しているので、文書は安全に保たれる。

原本性(Authenticity) 全ての文書は、常に存在するAuthDoc(=利用者)に関連付けられており、その変更に関する安全性についても第9章で保証済である。

12.1.2　第2.2節での要件

続いて、第2.2節で策定した幾つかの要件について考察する。

文書の寿命が媒体の寿命に拘束されないようにすること：物理層を設定したことによって達成。

任意の情報の形式に関する情報を保持すること：仕様書という種別の文書を作ったことによって達成。

文書に永続的な名称が与えられること：文書識別子によって達成。

ライフサイクル管理を支援すること：制御文書という種別の文書を作ったことによって達成。

Worldwideな管理組織を確保すること：認証文書もまた分散層を通してWorldwideに運用されることによって達成。

各文書の管理責任が明らかにされること：各文書から認証文書へのリンクを張ることによって達成。

文書の引き継ぎ(管理者の変更)が確実に行われること：解釈層の$chown, denyown, getown$という一連のコマンドによって達成。

文書の改竄に対策を施すこと：ログインをパスワード認証にすることと、文書の書き換えが所有者にしかできないこととによって達成。

文書の流出に対策を施すこと：制御文書に公開範囲を設定することによって、一応の歯止めはかけている。しかし、本格的な流出対策を行うには、文書を利用者に渡すときにサーバの鍵による電子署名を付加するなどの対策が必要となるであろう (第 12.4.2 節参照)。

第三者が任意の文書の管理の適切さを検証できること：利用者が変更することができないログ部を POT 文書に作り、さらにそれが常に公開されることにしたことによって達成。

ネットワークを通じた文書情報の交換を容易にすること：分散層を設けることによって達成。

以上、いずれの要件も、何らかの形で達成されていることが確認できた。

12.2　モデルの評価

12.2.1　長所

初のモデル：本研究は、電子情報を長期的に保存する際にどのようなモデルが必要になるかを包括的に述べ、プロトタイプまで実装したものは、著者の知る限りでは、本研究を以て嚆矢とする。この研究をきっかけに、この方面の議論が活発になることを望むところである。

メタデータの安全性：本モデルの下位の階層ではログを含むメタデータをデータと同じ資格で格納しているので、データの関係性について、データと同じだけの安全性を保つことができる。

柔軟な文書構造：文書層はリンクの具体的種類を定義していないので、将来このモデルに何らかの関係を追加で定義したくなったときも、解釈層のみを改造するだけで容易に拡張できる。

12.2.2　短所

媒体への制約：このモデルにおいて、文書が正しく保護されるか否かは、ひとえに認証層におけるパスワード認証に懸かっている。言い替えれば、媒体は常にオンラインに接続された状態にあり、必ず上位の階層の適切なコマンドを経て登録・更新されることを要請している。即ちこれは、悪意の第三者が媒体を外して改竄した場合、上位の階層から改竄を見分けることは不可能であることを示している。

従って、上位の階層を経ない情報のやりとり (例えば、媒体に記された POT 文書 (の XML による表現) を CD にプレスして配布するなど) は、媒体を直接操作されることはないという原則を崩すことになるので、前節のモデルを使用する限り不可能となる。

信頼性の管理者への依存：本モデルでは、文書ごとに認証文書を用いて管理者を定めている関係上、管理文書のその時々の管理者については全面的に信用するしかないし、また、前項に記した媒体への制約から、媒体の管理者も信用するしかない。

12.3　想定する利用例

筆者が想定しているシステムの利用例は、主に以下の二つである。

12.3.1　官公庁の文書の出納

まず、筆者が念頭においているのは、官公庁などの公的セクターに於ける電子文書の保管システムである。

本モデルは、今まで公開の手段がないため退蔵されてきた電子的資料を、できるだけ公開しつつ保存できるように設計されている。このようなデザインのシステムは、情報公開が今後ますます求められる公共機関の情報管理システムに適していると考える。

12.3.2　電子情報の個人的蒐集

次に考えられる大きな使用例の類例が、個人レベルの電子的情報の蒐集である。

一般には入手が困難であるような資料性の高い情報を蒐集・公開している篤志家は、WWWの世界には多く見られる。しかし、彼ら篤志家はしばしば個人的・趣味的なボランティアでサーバを運営している場合が少なくなく、公的セクターのように存立基盤が安定しているわけでは必ずしも無い。故に、サイトの存続が怪しくなっても安心して情報をよそに譲渡できるようなシステムがあれば、彼らを益するところ大であろう。

そもそもインターネットとは、速報性のみにおいて有用であるのみならず、従来の方法では配布・入手の困難な文献を 容易に全世界に知らしめる手段でもある筈である。自らが蒐集した情報を永続的に公開することを保証するシステムの存在は、篤志家による情報の供与をより盛んにすると考えられる。

12.4　今後の課題

最後に、本研究の範囲内では果たせなかったトピックスを幾つか掲げ、今後への展望に代える。

下に見るように、モデル全体に対する「やり残し」の事項は少なくないが、下記のような課題群を適切な位置に置くことのできる枠組を作ったという意味では、確かに本研究による成果はあったものと考える。

12.4.1　分散層周り

第8.2.2節でふれたような効果的な文書の分散のさせ方についての考察、例えば、ドメインとサイトを対応させ、文書の流出を防止するなどのモデルの追求は、システムのパフォーマンスを向上させるうえで重要となるであろうが、本研究ではfuture workに終わってしまった。

12.4.2　認証に関して

本研究の範囲内ではパスワードは素通しにしてしまったが、パスワードの保存に暗号化を採用した場合、具体的な暗号方式については、技術革新に従って暗号方式を順次変更することができる。認証層の強度は暗号方式の強度に依存しているので、これは重要なポイントである。

ただし、ファイル形式の仕様書と異なり、暗号処理系は POT システム自体が扱うものなので、暗号方式の更新に伴って暗号モジュールを開発する手間は避けられない。

12.4.3　暗号関連技術の再検討

文書の要素としては署名を含めないと第 4.2.3 節では述べたが、これは、システムの頑健性を高めるために、永続的要素以外の部分で暗号技術を用いることを否定するものではない。従って、文書保護のための物理層に記されるレベルで文書を暗号化するとか、文書の流出対策として閲覧時などに電子署名を加えるなどといった方面の研究も、今後の課題であるといえよう。

12.5　結語

本研究では、電子情報の長期的管理技術の不備に鑑み、POT という長期管理のためのモデルを提案した。POT モデルは命名規則・文書モデル・階層的アーキテクチャ等から成り、近年発達しつつある各方面の情報技術を取り込み、適切に位置付けつつ、長期的に一貫した文書管理の体系を提供することを可能にする。また、そのモデル (の一部) に基づいた実装を作成・実行し、モデルが確かに実装可能であることを示した。

謝辞

この論文を執筆するに当たっては、本当に様々な方々のお世話になりました。

指導教官である広域システム科学系の川合慧教授からは、研究の進め方について多大な教唆を頂きました。

同系の山口和紀教授には、情報基盤センターと兼担になられたにもかかわらず、私の至らぬ原稿に手取り足取りつきあって下さり、感謝の言葉もございません。

KTYY 合同ゼミの皆様からは、両教授に負けるとも劣らぬご指導の数々、暖かい声援、ゼミでのツッコミなど、やはり大変お世話になりました。本論文の脱稿直前に誤植のチェックをして下さいました白石路雄氏と柏原賢二氏には、特に感謝申し上げます。

この学位論文のもととなった論文 [24] が世に出るきっかけを作って下さった中谷多哉子氏には、本当に頭が上がらない思いです。

プロトタイプの実装上欠かすことのできなかった XML for Java の作者である田村健人さんには、どちらに家があるかは存じませんが、もはや足を向けて眠れない気分です。

講演会のために来日し、研究に多大なヒントを下さった David O. Stephens 氏、並びに氏の来日に尽力された ARMA 東京支部ならびに記録管理学会の皆様にも感謝致します。

最後に、記録管理の世界に興味を抱くきっかけを作って下さった北海道札幌南高等学校 (当時) の小川洋介教諭 (当時) と高橋一寿君にも、これをご覧になる筈はないと知りつつも感謝するものです。

以上の方々に対し、重ねて深くお礼申し上げます。

参考文献

[1] Nabil ADAM and Yelena et al. YESHA. Strategic directions in electric commerce and digital libraries: Towards a digital agora. *ACM Computing Surveys*, Vol. 28, No. 4, pp. 818 – 835, December 1996.

[2] T. Dierks and C. Allen. The tls protocol version 1.0, January 1999. RFC2246.

[3] Distributed.net. http://www.distributed.net/.

[4] R. Fielding, et al. Hypertext transfer protocol – http/1.1, January 1997. RFC2068.

[5] International Organization for Standization. http://www.iso.ch/.

[6] International Organization for Standization. Data elements and interchange formats – information interchange – representation of dates and times. ISO 8601:1988.

[7] International Organization for Standization. Information processing – text and office systems – standard generalized markup language (sgml). ISO 8879:1986.

[8] International DOI Foundation. The document object identifier. http://www.doi.org.

[9] Alan O. Freier, Philip Philip Karlton, and Paul C. Kocher Kocher. The ssl protocol version 3.0. Internet Draft, March 1996.

[10] W3C PICS Interest Group. Platform for internet content selection (pics). http://www.w3.org/PICS/.

[11] W3C XML Working Group. Extensible markup language (xml) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210.

[12] Tamura Kent. Xml for java. http://www.trl.ibm.co.jp/projects/xml/xml4j_j.htm.

[13] Carl Lagoze, Clifford A. Lynch, and Ron Daniel Jr. The warwick framework: A container architecture for aggregating sets of metadata. http://cs-tr.cs.cornell.edu:80/Dienst/Repository/2.0/Body/ncstrl.cornell%2fTR96-1593/html/.

[14] R. Moats. URN syntax, May 1997. RFC2141.

[15] The Commission on Preservation, Access, and Inc. The Research Libraries Group. Preserving digital information. http://www.rlg.org/ArchTF/tfadi.index.htm, May 1996. Report of the Task Force on Archiving of Digital Information.

[16] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). ACM SIGMOD Conference, Chicago, IL, June 1988.

[17] David O. Stephens. 文書管理における電子化の世界的動向. ARMA International 東京支部講演会, July 1998.

[18] Sam X. Sun, Sean Reilly, and Larry Lannom. Handle system namespace and service definition, July 1999. Internet Draft, draft-sun-handle-system-def-01.txt.

[19] Sam X. Sun, Sean Reilly, and Larry Lannom. Handle system overview, July 1999. Internet Draft, draft-sun-handle-system-03.txt.

[20] Ralph Swick, et al. Resource description framework. http://www.w3.org/RDF/.

[21] The PURL Team. Persistent url home page. http://www.purl.org/.

[22] Ron Vetter. Urls in print: To cite or not to cite. *IEEE Computer*, Vol. 31, No. 2, pp. 114–117, February 1998.

[23] Stuart Weibel and Eric Miller. Dublin core. http://purl.oclc.org/metadata/dublin_core/.

[24] 伊戸川暁, 川合慧, 山口和紀. 電子文書の長期保管のための枠組:POT. 情報処理学会論文誌: データベース, Vol. 40, No. SIG3(TOD1), 1999.

[25] 日本国国会. 電子計算機を使用して作成する国税関係帳簿書類の保存方法等の特例に関する法律. 官報, 1998. 平成 10 年法律第 25 号.

[26] 日本国国会. 行政機関の保有する情報の公開に関する法律. 官報, 1999. 平成 11 年法律第 42 号.

付録A 初期文書

ここには、`spec:root:xml` を除く5つの初期文書の姿を掲載している。

auth:root

```
<?xml version="1.0"?>
<doc version="0.9" id="auth:root">
  <logs>
    <logitem id="create" date="1999-12-22T06:12:27"/>
    <logitem id="auth:root" date="1999-12-22T06:12:27"/>
    <logitem id="ctrl:root:permanent" date="1999-12-22T06:12:27"/>
    <logitem id="spec:root:authspec" date="1999-12-22T06:12:27"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:authspec"/>
  </links>
  <text>
    <passwd>toor</passwd>
    <announce>
      <message/>
    </announce>
  </text>
</doc>
```

ctrl:root:permanent

```
<?xml version="1.0"?>
<doc version="0.9" id="ctrl:root:permanent">
  <logs>
    <logitem id="create" date="1999-12-22T06:12:27"/>
    <logitem id="auth:root" date="1999-12-22T06:12:27"/>
    <logitem id="ctrl:root:permanent" date="1999-12-22T06:12:27"/>
    <logitem id="spec:root:ctrlspec" date="1999-12-22T06:12:27"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:ctrlspec"/>
  </links>
  <text>
<time value="A-0">
<range value="root">all</range>
</time>
  </text>
</doc>
```

spec:root:plaintext

```
<?xml version="1.0"?>
<doc version="0.9" id="spec:root:plaintext">
  <logs>
    <logitem id="create" date="1999-12-22T06:12:27"/>
    <logitem id="auth:root" date="1999-12-22T06:12:27"/>
    <logitem id="ctrl:root:permanent" date="1999-12-22T06:12:27"/>
    <logitem id="spec:root:plaintext" date="1999-12-22T06:12:27"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>
The text is written by plain ASCII text format.
  </text>
</doc>
```

spec:root:authspec

```
<?xml version="1.0"?>
<doc version="0.9" id="spec:root:authspec">
  <logs>
    <logitem id="create" date="2000-01-11T18:54:16"/>
    <logitem id="auth:root" date="2000-01-11T18:54:16"/>
    <logitem id="ctrl:root:permanent" date="2000-01-11T18:54:16"/>
    <logitem id="spec:root:xml" date="2000-01-11T18:54:16"/>
  </logs>
  <links>
    <link id="auth:root"/>
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>&lt;?xml version="1.0" encoding="EUC-JP"?&gt;
&lt;!ENTITY % auth INCLUDE&gt;
&lt;!ENTITY % ordn IGNORE&gt;
&lt;!ENTITY % ctrl IGNORE&gt;
&lt;!ENTITY % potbase SYSTEM "potbase"&gt;
%potbase;
</text>
</doc>
```

spec:root:ctrlspec

```
<?xml version="1.0"?>
<doc version="0.9" id="spec:root:ctrlspec">
  <logs>
    <logitem id="create" date="2000-01-11T19:10:45"/>
    <logitem id="auth:root" date="2000-01-11T19:10:45"/>
    <logitem id="ctrl:root:permanent" date="2000-01-11T19:10:45"/>
    <logitem id="spec:root:xml" date="2000-01-11T19:10:45"/>
  </logs>
  <links>
    <link id="auth:root"/>
```

```
    <link id="ctrl:root:permanent"/>
    <link id="spec:root:plaintext"/>
  </links>
  <text>&lt;?xml version="1.0"?&gt;
&lt;!ENTITY % ctrl INCLUDE&gt;
&lt;!ENTITY % ordn IGNORE&gt;
&lt;!ENTITY % auth IGNORE&gt;
&lt;!ENTITY % potbase SYSTEM "potbase"&gt;
%potbase;

</text>
</doc>
```

付録B 設定ファイル

実験中に用いた各層のサーバのための設定ファイルをここに掲載した。なお、これらのファイルは、全て`/home/users/itogawa/ research/potroot/conf` に置かれていた。

phys(物理層サーバ用設定ファイル)

```
# Configure file for Phys.
LogFilePath     /home/users/itogawa/research/potroot/logs/phys
#LogFilePath     stdout
ListeningPortNumber     19993
URLCatalogPath  /home/users/itogawa/research/potroot/conf/urns
BasePath        /home/users/itogawa/research/potroot
```

dist(分散層サーバ用設定ファイル)

```
# Configure file for Dist.
LogFilePath     /home/users/itogawa/research/potroot/logs/dist
ListeningPortNumber     19995
URLCatalogPath  /home/users/itogawa/research/potroot/conf/urns
FeiendHostNames VECTOR arcadia graf gimlet
DomainName graco.c.u-tokyo.ac.jp
RemoteHostName localhost
RemotePortNumber        19993
```

docs(文書層サーバ用設定ファイル)

```
# Configure file for Docs.
LogFilePath     /home/users/itogawa/research/potroot/logs/docs
ListeningPortNumber     19997
URLCatalogPath  /home/users/itogawa/research/potroot/conf/urns
RemoteHostName  localhost
RemotePortNumber        19995
```

intp(解釈層サーバ用設定ファイル)

```
# Configure file for Intp.
DefaultLimit    A-0-0-0-0-30
LogFilePath     /home/users/itogawa/research/potroot/logs/intp
```

```
ListeningPortNumber     19999
URLCatalogPath  /home/users/itogawa/research/potroot/conf/urns
RemoteHostName  localhost
RemotePortNumber        19997
LocalRoot       root
```

urns(urn 設定ファイル)

(空であった。)

付録C ソースコード

次のページより、POT プロトタイプシステムのソースコードを掲載する。なお、ソースコードは、`http://www.c.u-tokyo.ac.jp/ itogawa/study/pot/pot-0.9.tar.gz` からも入手可能である。付録 A 及び B のデータと合わせて、あなたも POT プロトタイプを体験することができる。

実行のために必要な準備は以下の通りである。

1. JDK1.1.7 と XML for Java 1.1.14 以上が手元の環境にあることを確認する。
2. ソースを (付録 A と B も含め) 入力あるいはダウンロードする。
3. 必要ならば (まず間違いなく必要であろうが)、設定ファイルや `pot/Init.java` 中のパスを書き換える。
4. コンパイルする。
5. 下記の注意事項に従って設定ファイルと初期ファイルを配置する。
6. ソースのあるところにパスを通す。
7. 「`java pot.Init`」を実行する。
8. localhost の 19999 番ポートに接続する。

また、以下の点にも注意して頂きたい。

- 付録 A のファイルのそれぞれを置くべきパスは、本プロトタイプの場合、「(媒体のルートパス)/(文書識別子の:を:/に置き換えたもの)」となる。
- 付録 B の設定ファイルは、必ず正確なパスに置くこと。