第3章　多重名前空間

本章では，グローバルに単一な名前空間を提供しつつ，状況に応じて動的に名前空間の構成を変更可能とする多重名前空間の構成方式を提案する．

3.1　多重名前空間の構成

一つの名前空間は，ドメインネームやUnixのファイルシステムのような木構造をしており，グローバルで単一な名前付けが行われるものとする．提案方式では，そのような名前空間の存在を複数許し，それら複数の名前空間を重ね合わせることで一つの名前空間を仮想的に構成する．この仮想的に構成された名前空間のもとで，指定された資源名の名前解決を行う．ここで，各名前空間の名前は，グローバルで一意な名前付けが行われており，その名前空間の構造は特に構造を持たないフラットな名前空間であるとする．名前空間を重ね合わせる際に，それらの空間の間の全順序を指定する．この全順序の指定をビューパス(view path)と呼び，$N_i (1 \leq i \leq m)$ をそれぞれ重ね合わせる名前空間の名前として，$\{N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_m\}$ のように表記する．ここで指定するビューパスは，名前空間のユーザが名前指定のたびに指定する．ユーザは，ビューパスを適切に変更することで名前空間のカスタマイズを行うことが可能である．ユーザに見える仮想的な名前空間は，N_1 から N_m に至るまでのビューパス内に存在する m 個の名前空間の和をとったものとなる．ここで,「名前空間の和」とは,それぞれの空間に存在する論理資源名の集合の和として定義される．ただし，同一の論理資源名がビューパス内の複数の名前空間に存在している場合は，ビューパスのより上位に存在する論理資源名が選択され，選択された論理資源名と物理資源名の対応に基づいて名前解決を行うものとする．

名前空間に名前を追加する時は，ユーザが指定したビューパスの最上位の空間に名前が追加され，名前を削除する際には，ビューパスのより上位に存在する空間上の名前が削除される．

提案方式では，ユーザが資源名を指定する場合，必ずビューパスと資源名の組を指定する．例えば，図3.1に示した多重名前空間の場合，ユーザがビューパスを $\{\alpha \rightarrow \beta \rightarrow \gamma\}$

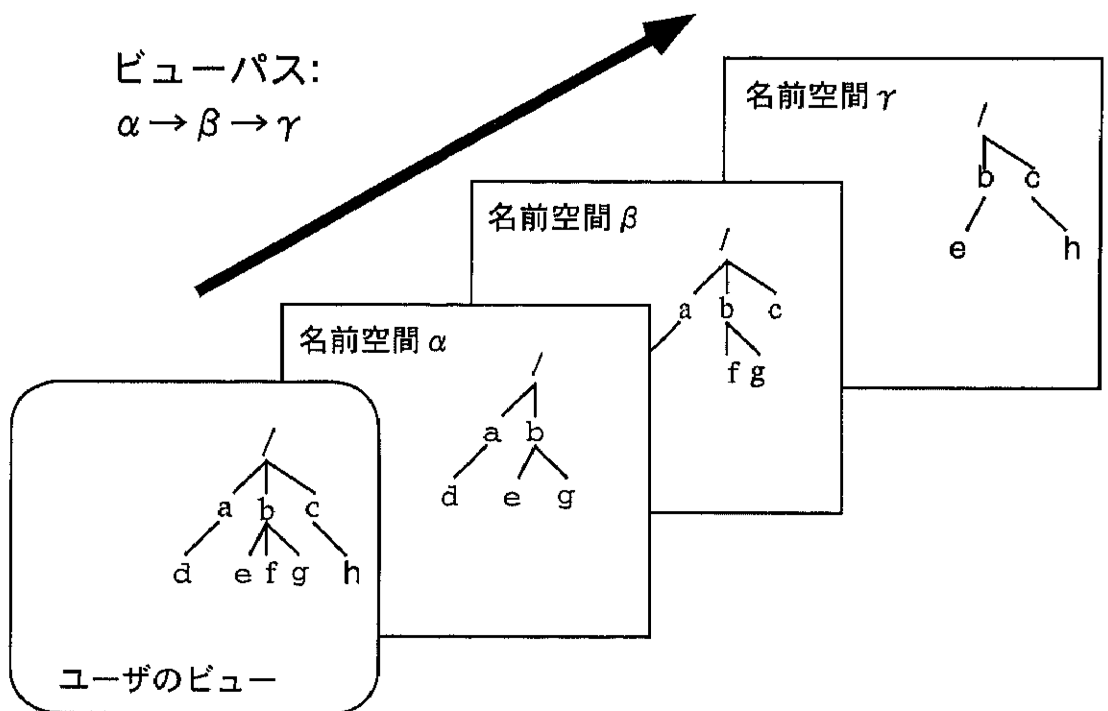

図 3.1: 多重名前空間の例

として，資源名/b/fを指定するときは，組

$$(\alpha \rightarrow \beta \rightarrow \gamma, \text{/b/f})$$

を指定する．この例の場合，名前解決の結果，名前空間 β の資源/b/fが選択されることになる．

提案方式では，上位の名前空間に存在する名前で下位の名前を隠蔽することによって名前空間のカスタマイズを行うため，同一ビューパス内に現れる各名前空間に配置する名前に関して同一の名前付けのルールが存在することを仮定している．名前付けに関するルールを重ね合わせた場合，それぞれのルールの違いから名前付けに関するルールが名前空間内で混在することになる．名前付けに関するルールは，論理的に同じ資源を同じ名前で表現するためには必要不可欠なものであるため，同一のルールが存在する事を仮定することは多重名前空間の使用範囲を限定するものにはならないと考えられる．

3.2 名前空間のカスタマイズ

多重名前空間を用いた名前空間のカスタマイゼーションは，大きく２つの方法に分けられる．第一の方法は，ビューパスを構成する名前空間を選択することにより，ユーザから見える名前空間を変更することである．第二の方法は，既存の名前空間に対して，ユーザの個人的な名前空間を上位に重ね合わせることにより，下位の名前空間内の資源を変更することなく，ユーザ独自の資源への変更処理を行うことによる．

第一の方法を用いることにより，第１章で述べた単一名前空間の欠点の第一の例である異機種分散環境での問題は以下のように解決できる．図3.2に例を示す．まず，機種(あるいは言語)依存性のない資源と機種依存性の有る資源を別々の空間に分けておく．仮に，機種依存性のない資源の存在する空間名を "Neutral"，機種依存性の有る空間名をその機種名を用いて "Arch-A"，"Arch-B" として生成する．ここで，各名前空間において名前の使用法に関する一定の convention が有ることを仮定する．すなわち，論理的に同一の資源を指し示す名前に対しては機種依存・非依存にかかわらず同一の名前が使用されることを仮定している．ユーザは，機種依存性のない空間の上位に各ユーザが使用する機種に適した空間を置くようにビューパスを設定する．例えば，機種 Arch-A を使用するユーザは，ビューパスを $\{\text{Arch-A} \rightarrow \text{Neutral}\}$ と設定する．また，機種 Arch-B を使用するユーザは同様に $\{\text{Arch-B} \rightarrow \text{Neutral}\}$ と設定する．このビューパスの指定により，各機種のユーザは同一の論理名を使用しつつ，かつ適切な機種依存性の有る資源を指定することが可能となる．提案方式である多重名前空

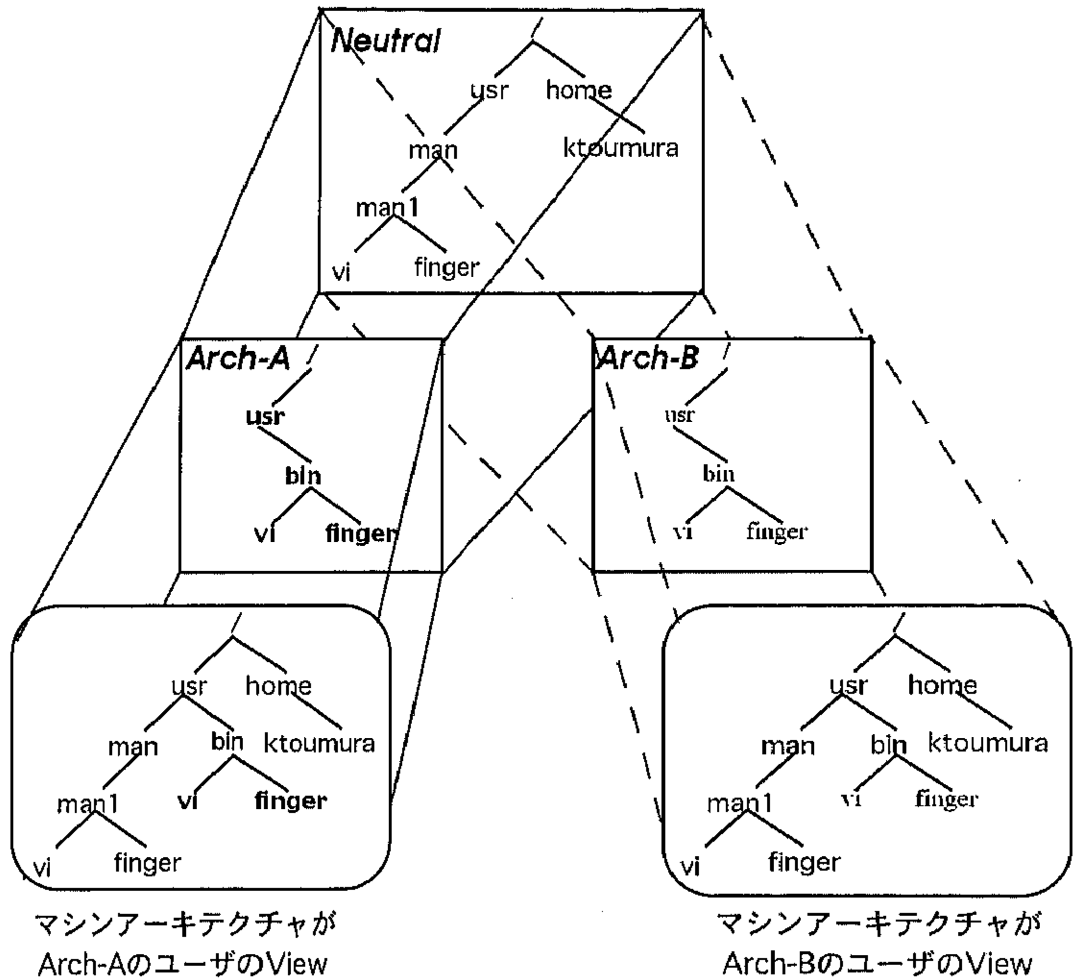

図 3.2: 異機種分散環境での利用例

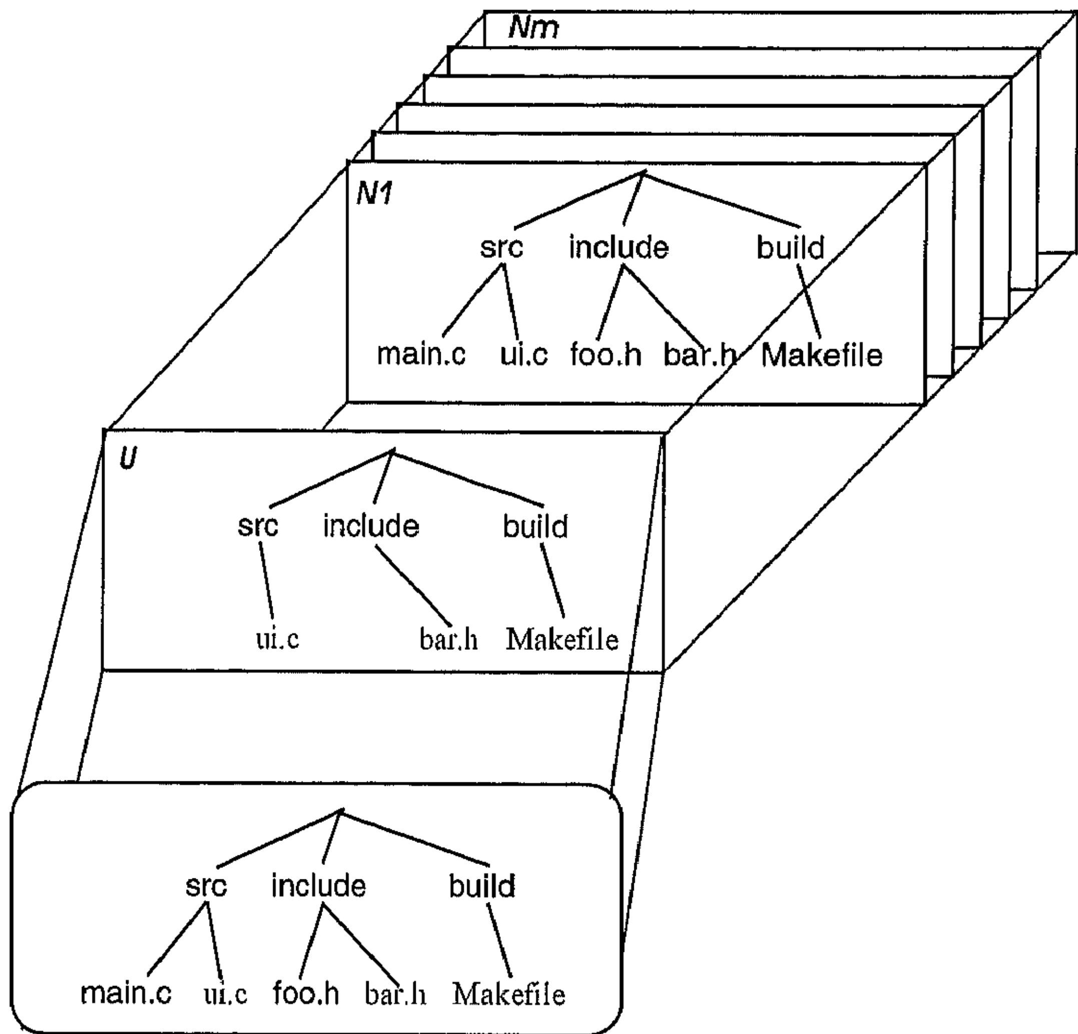

図 3.3: 計算機を用いた設計作業での利用例

間を利用しない場合は，論理資源名の中に機種情報を埋め込むことで機種依存のある資源を識別する ("/usr/Arch-A/bin" 等の名前の利用) 必要や，全く別の名前空間を機種ごとに用意し，機種ごとにそれぞれの名前空間を使用するという ad hoc な方法を用いる事になる．それらの方法には，前者では論理的に同一な資源が異なる名前で参照する必要があること，後者では機種依存性のない資源であっても複数の名前空間に同一のコピーを置く必要があることなどの欠点がある．提案方式では，単一な名前空間をそれぞれ用意するのではなく，重ね合わせとして名前空間を表現するため，機種依存性のない資源に関して複数の名前空間にコピーを置く必要はなく，機種依存性のない資源に関する管理のオーバーヘッドの増大を防ぐことが可能である．

第二の方法を用いることにより，第1章で述べた単一名前空間の欠点の第二の例である計算機を用いた設計作業に関する問題は，以下のように解決できる．図3.3に例を示す．まず，設計中のシステムで使われているビューパスの指定が$\{N_1 \rightarrow N_2 \rightarrow \ldots \rightarrow N_m\}$となっていたとする．設計者は，一時的な変更を加えるためにこのビューパスの最上位に設計者の生成した名前空間Uを付加する．すなわちビューパスは$\{U \rightarrow N_1 \rightarrow \ldots \rightarrow N_m\}$となる．このビューパスによって構成された名前空間に対し，設計者は変更を行う．設計者の行った変更は，最上位の名前空間であるUに対して行われるため，設計中のシステムに対し変更を行うことなく，実験的な変更を行うことが可能である．設計した内容を設計中のシステムに永続的に反映するためには，名前空間の間でファイルのコピーを行う．提案方式では，名前解決ごとにビューパスの指定を変更することが可能である．よって，コピー元のファイル名の指定時のビューパスを設計者の名前空間Uとし，コピー先のファイル名の指定としてN_1からN_mの適切な名前空間を指定し，実験的な変更内容をシステムに反映することが可能である．第1章で述べた単一名前空間の欠点の第三の例も同様に，ユーザの生成した名前空間を最上位に指定することでオリジナルの資源に影響を与えることなくユーザ独自の変更を行うことが可能となる．

3.3 設計と実現

多重名前空間を構成する各々の名前空間は，システム内の複数のマシン上に存在する名前サーバと，その間を結ぶリモートリンクから構成される (図3.4参照)．各名前サーバには，名前空間の一部が保存される．リモートリンクは，各名前サーバに保存されている名前空間の続きに当たる名前空間へのリンクを保持している．リモートリンクの情報はシステム管理者が行うもので，ユーザはリモートリンクの存在を意識する必要はない．

この設計は，広域ネットワーク環境における名前解決システムとして広く使用されている Domain Name System のサーバ構成に基づいている．DNS に関しては第2章を参照されたい．

この設計に基づいて，単純な実装を行うと以下のような方式になる．クライアント側に置かれる名前解決ライブラリは，これらのサーバに順に要求を行うことで名前探索を行う．例として，名前探索を行う場合を考える．

まずクライアントは，探索する名前空間を決定する．これは，多重名前空間のセマンティクスに従うと，ユーザからあたえられたビューパスの最も上位の空間から順に探索を行うことを意味する．クライアントは，探索する空間を決定した後，その単一名

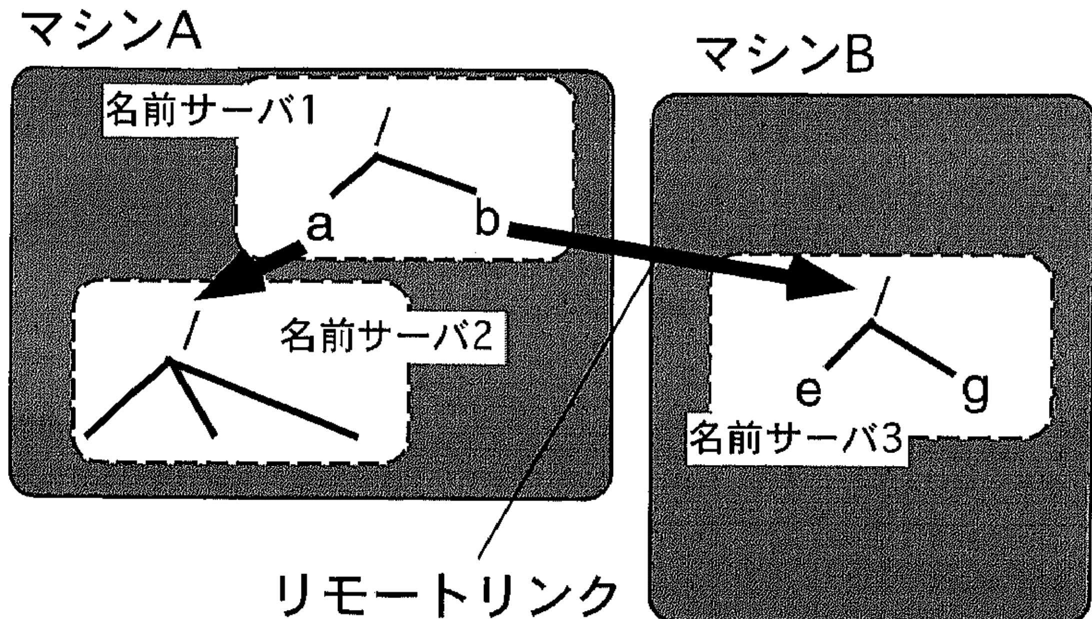

図 3.4: 多重名前空間のサーバ構成

前空間内での探索を行う．単一名前空間内での探索は，名前空間のルートからリモートリンクをたどりながら探索を行う．

3.4 多重名前空間の効率的な実装法

本システムにおける名前解決の処理は，名前空間を選択する処理と，選択された名前空間内で名前解決を行う処理の 2 つの処理に大別される．

3.4.1 単一名前空間内探索の効率化

本提案システムでは，前述した通り単一名前空間が複数の名前サーバから構成されている．単純に単一名前空間内の探索を行うと，ルートサーバから順にリモートリンクをたどることとなり，無駄なトラフィックを発生することになる．そこで，本提案方式では，単一名前空間内を探索する処理に関して，文献 [34] で提案されたプレフィックス・テーブルを用いることで名前解決の効率化を行う．

プレフィックス・テーブルは，クライアントがヒントとして保持するキャッシュ情報である．プレフィックス・テーブルの特徴は，名前のプレフィックスの最長一致で

名前空間名	プレフィックス	位置情報
X	/	192.168.1.240 D1
Y	/	192.168.1.243 D1
Z	/	192.168.1.246 D1

図 3.5: プレフィックス・テーブルの初期状態の例

エントリのマッチングを行う点と，それがヒント情報であるという 2 つの点である．全ての名前に対してその位置情報をキャッシュした場合，そのキャッシュの容量は膨大なものとなり，キャッシュのオーバーフローが頻繁に発生してしまう．これは，結果としてキャッシュの性能を落とすものとなってしまう．多数のキャッシュエントリの共通部分としてそのプレフィックスを格納することで，情報を圧縮することができる．このとき，プレフィックスの最長一致でエントリのマッチングを行うため，間違った情報を返す場合がある．例えば，"/usr/bin/ls" というファイルがサーバ A に，また "/usr/local/bin/emacs" というファイルがサーバ B に存在していたとする．また，プレフィックス・テーブル内に "/usr" がサーバ A に存在するという情報が記録されていたとする．この場合，"/usr/local/bin/emacs" とプレフィックステーブルをマッチングすると，"/usr" をプレフィックスとしてサーバ A に存在するという情報が返ってくる．しかし，この場合，サーバ A にアクセスすることで "/usr/local" 以下がサーバ B に存在することが判る．これは，名前空間が木構造をしているため，特定のプレフィックスを持つ名前は，一つのサーバか，そのサーバからたどることのできるサーバに存在しているからである．このため，プレフィックス・テーブルを参照して異なるサーバに問い合わせを行っても問題は発生しない．

プレフィックス・テーブルは，以下の情報を保持している．

- 名前空間名
- 名前のプレフィックス
- 名前サーバの位置情報 (アドレス及びサーバ識別子)

プレフィックス・テーブルは，クライアントが起動した直後には各名前空間のルートを管理しているサーバとその位置情報のみが保持されている (図 3.5)．これは，DNS におけるルートサーバの情報と同様であり，この情報は全てのサーバが既知の情報として保持しているものである．

この初期状態に対し，クライアントが名前解決を行った際に得たリモートリンクの

名前空間名	プレフィックス	位置情報
X	/	192.168.1.240 D1
Y	/	192.168.1.243 D1
Z	/	192.168.1.246 D1
X	/a	192.168.1.241 D1

図 3.6: 情報が追加された状態

情報をプレフィックス・テーブルに追加していくことでキャッシュが形成される．例えば，名前空間 A の "/a" 以下はサーバ「192.168.1.241 D1」に存在することが名前解決で判明した場合，プレフィックステーブルは図 3.6 に示す状態になる．

名前解決を行う際は，その名前とプレフィックス・テーブルのマッチングをとり，最長一致したプレフィックスに対するサーバの位置情報を取得し，そのサーバに対し名前探索を行う．名前空間 X の "/a/b" という名前を例にとる．この名前に最長一致するプレフィックスは，図 3.6 における 4 番目のエントリである．よって，最初の探索ではサーバ「192.168.1.241 D1」へ探索要求を出すことになる．

3.4.2 単一名前空間間探索の効率化

次に，複数の名前空間の中から適切な名前空間の効率的に選択する手法について述べる．

本提案方式では，同じ名前が複数の空間に存在する場合，ユーザがあたえたビューパスにおいて最上位にある空間の名前が選択される．これを実現する最も単純な方法は，ビューパスに示された順で名前空間の探索を行い，該当する名前が見つかった時点で探索を終了するというものである．このようにすると，探索される名前が指定されたビューパスの下位に存在しているとき，名前が情報に存在している場合に比べ，相対的により多くの空間の探索を行わねばならない．この探索を高速化するために，ある名前に関する探索を行った際，クライアントはその名前が存在しなかった空間を情報としてキャッシュしておく．これを名前空間選択テーブルと呼ぶ (図 3.7)．

名前空間選択テーブルの各エントリには，

- 名前
- その名前が存在しない空間名の列

が格納される．一度探索した名前を再び検索する際は，ビューパスのうち名前が存在

名前	名前空間名
/a/b	X, Y
/b/c	X, Z

図 3.7: 名前空間選択テーブルの例

しないことがわかっている空間の探索を省略することで無駄な探索を抑制している．例えば，図 3.7 に示される状態でビューパスが

$$\{X \rightarrow Y \rightarrow Z\}$$

と設定されていたとする．この状態で "/a/b" に対する名前探索を行う際には，名前空間 X および Y への探索は，その空間に名前 "/a/b" が存在しないことが判っているため，探索を省略することが可能である．

このように，名前空間選択テーブルは，存在しない名前をキャッシュすることで探索を省略し効率化を行う．しかし，他のユーザによって名前の登録が行われた場合，名前の存在に関する情報の変化を名前空間選択テーブルに反映させ，名前解決の一貫性を保持させる必要がある．これに対処するため，サーバ内では各名前ごとに**名前空間一貫性情報**を保持している．名前空間一貫性情報には，同じ名前が存在する全ての名前空間名が記録されている．クライアントからその名前の検索が行われた際，サーバは返答に加えて名前空間一貫性情報を返す．クライアントでは，名前空間選択テーブルと名前空間一貫性情報に矛盾が生じていた場合，名前空間選択テーブルを更新し，再度名前解決を行う．

名前空間一貫性情報を常に正しい状態に保っておくには，名前の作成や削除が行われるたびに全てのサーバにその情報を伝達する必要がある．これを行うには，各サーバが他の全てのサーバの位置を記憶しておく必要がある上，大量のメッセージを送受信する必要があり，実行効率を大きく低下させる要因となりうる．この問題に対処するため，本システムではビューパスによる名前空間の順序指定を制限し，名前空間一貫性情報の伝播の必要を最小限に抑える．この制限は，各名前空間を頂点として有向非巡回グラフとして表現される．このグラフの辺を**名前空間順序リンク**と呼ぶ (図 3.8)．このリンクはシステム管理者により設定され，ユーザがあたえるビューパスは，この名前空間順序リンクに従ったもののみが許される．例えば，図 3.8 に示されるリンクが設定されていた場合，ユーザに設定が許されるビューパスの範囲は，$\{X \rightarrow Y \rightarrow Z\}$ や $\{V \rightarrow W\}$ 等に限られ，$\{Y \rightarrow X\}$ 等はビューパスとして許容されない．名前空間の順序指定を制限することにより，ユーザのビューパス選択の自由度は低下するが，

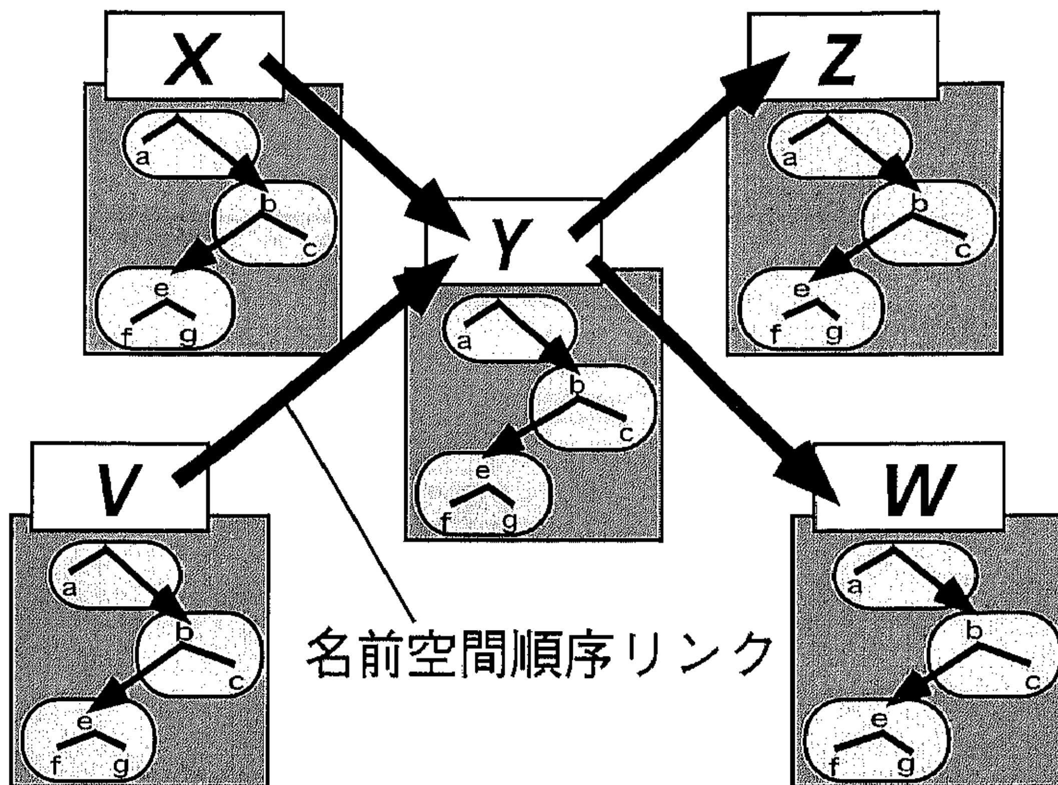

図 3.8: 名前空間順序リンク

名前空間順序リンクが管理者によって適切に設定されれば，実用上は問題ないと考えられる．

名前空間順序リンクを用いた名前空間一貫性情報の伝播は以下のように行われる．名前の作成が行われた空間のサーバは，名前空間順序リンクの先にある空間のサーバに対し，作成または削除されたファイルの情報を送る．これを受信したサーバは，これを名前空間一貫性情報に加えるとともに，名前空間順序リンクにしたがって次の名前空間のサーバに送信する．これによって，指定可能なビューパス上で新しく名前が作成された空間より後方に指定される可能性のある全ての名前空間において名前空間一貫性情報が更新される．

3.5 実験

第 3.4 節で述べた効率化方式による多重名前空間の名前解決システムの性能向上を測定するため，実験を行った．

実験環境を表 3.1 に示す．実験では，1 つのクライアントが 1000 回のリクエストを

表 3.1: 実験環境

計算機	Sun Netra t1 10 台
CPU	UltraSPARC-IIi, 440MHz
メモリ	512MBytes
ネットワーク	100BASE-TX FastEthernet(スイッチを経由し接続)
OS	Solaris 2.6
実装言語	C (Sun RPC[29] を利用)

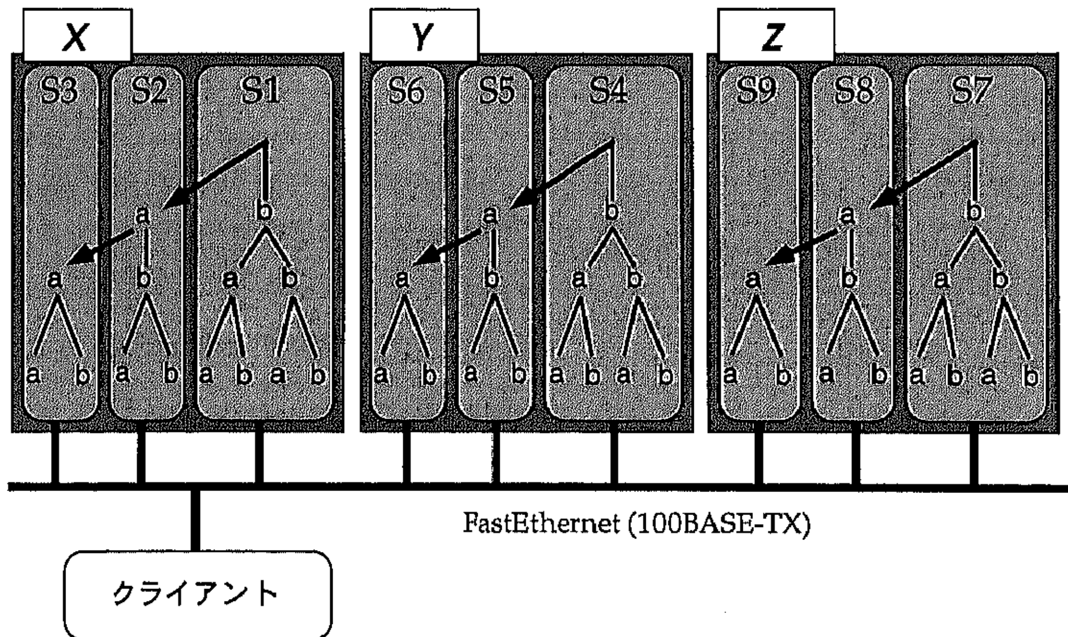

図 3.9: 実験時のマシン構成

発行した際に,

- クライアントのレスポンスタイム
- サーバに到着したメッセージの数

を測定した.

クライアントが発行するリクエストは,

- 名前の追加
- 名前の探索
- 名前の削除

の3種類に分類することができる. このうち, 名前の追加及び名前の削除を, 名前空間に対する更新操作として考える. そして, 全てのリクエストに含まれる名前空間に対する更新操作の割合をリクエスト列における更新率として定義する.

実験で使用したクライアントのリクエスト列は, 更新率を1%, 10%, 50%と変化させた. これは, 更新頻度の増加によって, サーバ間の一貫性情報に関する通信が増加し, サーバの負荷が上昇するかを調べるためのものである.

リクエスト列は一様乱数を元に生成を行い, 全ての名前は一様な確率で出現するようにした. 実験時のマシン構成からわかる通り, 各名前空間における名前は"/a/a/a", "/a/a/b", ..., "/b/b/b" の8種類が有り, それぞれ図3.9のように配置されるため, 各名前空間のルートサーバに50%の確率, ルートサーバからリモートリンクで繋がれた名前サーバにそれぞれ25%の確率で探索要求が行われることになる.

3.5.1 クライアントのレスポンスタイム

図3.10にクライアントのレスポンスタイムを示す. ここでいうレスポンスタイムとは, クライアントが要求をサーバに送信してからクライアントが要求に対する返答を受け取るまでの時間である. この結果から, 提案方式による効率化によってレスポンスタイムが向上していることが判る. また, 選択テーブルによる性能向上は, プレフィックス・テーブルによる性能向上に比べ, 更新率の大小による変化が大きいことが判る. これは, 今回の実験では名前の更新が行われた際でもプレフィックスに関する情報は変化しないためである. すなわち, 名前の変更はすべて単一サーバ内での名前の出現と消滅であり, リモートリンクの変更という操作は今回の実験ではリクエスト列に含まれていないため, プレフィックス・テーブル内の情報が名前の更新によって無効化されることがないことによる.

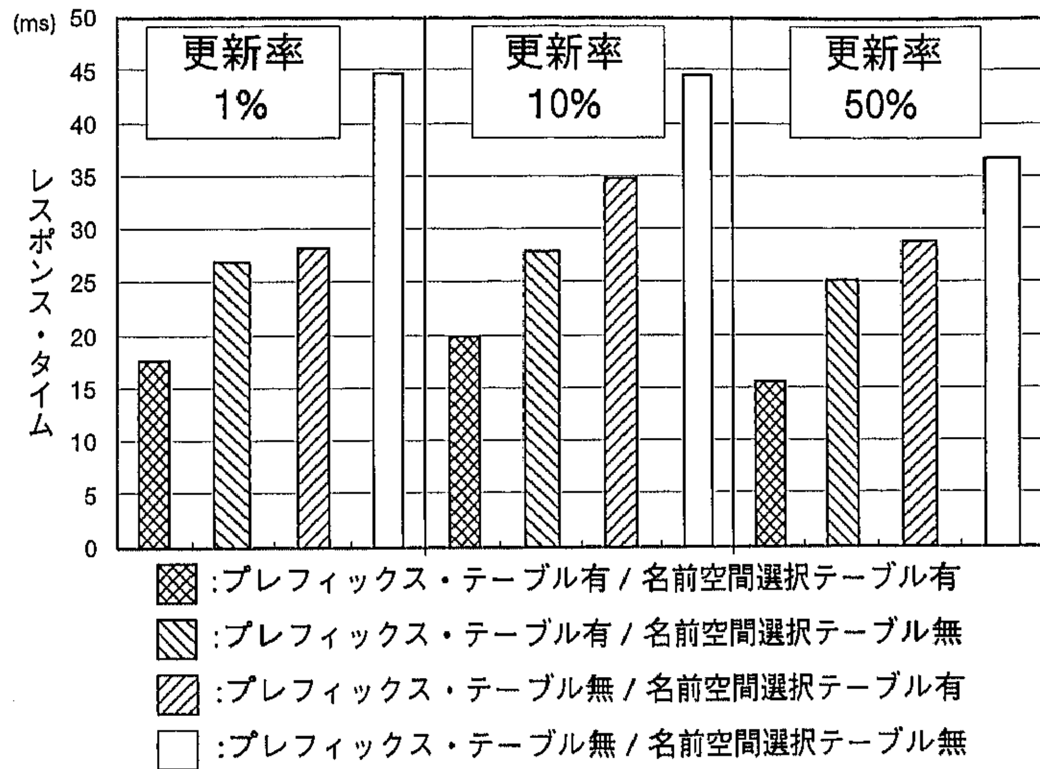

図 3.10: クライアントのレスポンスタイム

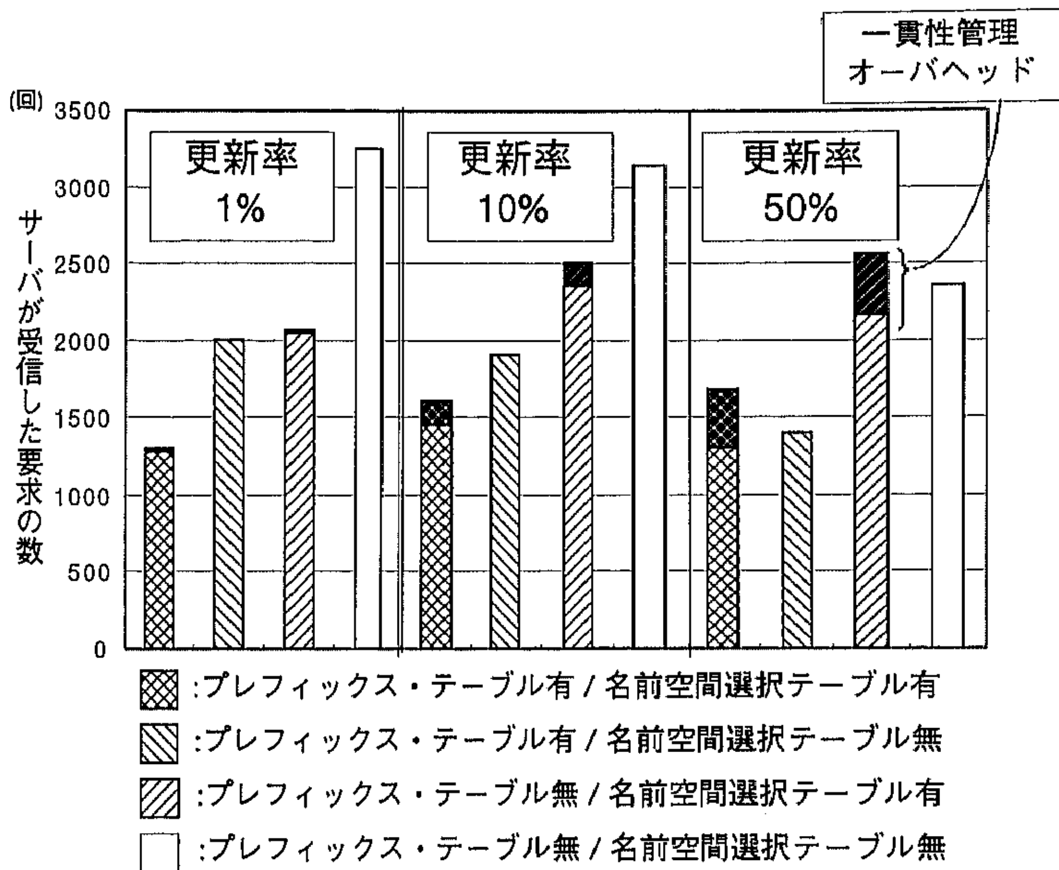

図 3.11: サーバが受信した要求の数

3.5.2 サーバが受信した要求の数

図 3.11 にサーバが受信した要求の数を示す．ここでいう要求とは，クライアントが発行した要求に加え，サーバ間で交換される一貫性管理のための要求も含まれる．後者の要求は，グラフ中では黒い部分で示される．

クライアントからの要求に関しては，第 3.5.1 節で示した結果と同様の傾向を示している．これは，リモートリンクで接続された先の名前空間にある名前に対する要求において，クライアント側のキャッシュ機構により直接的にサーバを探索することができるからである．また，名前空間選択テーブルによっても同様の効果が得られる．

しかし，サーバ間では名前が更新されるたびにその情報を交換する必要があるため，名前の更新率があがればあがるほど，そのコストは大きいものとなる．一貫性情報の交換は，名前空間選択テーブルを導入したときにのみ必要となる．これは，プレフィックス・テーブルによる探索の効率化においては，その情報は単なるヒントとして用い

られるため，キャッシュを用いた探索が失敗した場合でも，クライアントは単にその情報を無効化し再度探索を行えばよいためである．これに対し，名前空間選択テーブルによる探索の効率化は，ある空間に名前が存在しないことを情報として保持しているため，その空間における名前に対する探索が行われないため，クライアント側の探索ルーチンはサーバからの何らかの情報提供がなければその情報を無効化すべきことを検知することが不可能である．

この実験結果では，更新率が増加するほど，一貫性管理のオーバヘッドが高くなり，更新率が50%の場合においては，名前空間選択テーブルを導入することによってサーバ間の通信が増大してしまっていることがわかる．

3.6 まとめ

本章では，名前空間を単純かつ柔軟な方法でカスタマイズ可能とする多重名前空間の提案を行った．多重名前空間では，単一な名前空間を複数用意し，それらを重ね合わせることで名前空間のカスタマイズを行うことが可能である．また本章では，多重名前空間における名前解決を効率的に実現するための方法として，プレフィックス・テーブルと名前空間選択テーブルという２つの機構をクライアント側に作成し，サーバ側で名前空間選択テーブルの一貫性保持を補助する方式を提案した．そして実験において，これらの実現方式によって名前解決の効率化が行えることを示した．