ネーミングシステムを利用した透過的サービス合成システムの設計と実装

南　正輝 †　　齊藤　昭 ‡　　森川　博之 †　　青山　友紀 ‡

† 東京大学大学院 新領域創成科学研究科
‡ 東京大学大学院 情報理工学研究科

様々なサービスのの遍在化が進む今日のインターネットにおいて，サービスへの透過的アクセスを実現する手段として，ネーミングシステムの重要性が高まりつつある．このような中，現在のインターネットではDNSを主体とした様々な透過性の実現が行われている．しかしながら，DNSが担っている透過性の中には，DNS本来の設計と親和性の低い利用形態を要求するものがあり，その影響が懸念されている．このような状況は，コネクティビティ，コンピューティング，コンテンツの遍在化がさらに進み，物理空間へもインターネットが進出するであろう将来には，さらに顕著なものとなることが予想される．このような観点から，本稿ではDNSと相補的な新たなネーミングシステムの一デザインとその初期的実装について述べる．本稿で述べるネーミングシステムは，ネットワーク上に遍在する様々な機能の透過的発見と接続によりサービスの透過的合成を実現するためにデザインされており，機能のインターフェースに着目した名前空間の集約管理と解決機構を備えている．

A Design and Implementation of Naming-based Network Service Synthesizer

Masateru Minami †　Akira Saito ‡　Hiroyuki Morikawa †　Tomonori Aoyama ‡

†Graduate School of Frontier Sciences, The University of Tokyo
‡Graduate School of Information Science and Technology, The University of Tokyo

STONE(Service synThesizer On the NEt) is a new network service platform for supporting environment-aware services. STONE assumes an environment where many kinds of data processing functions with communication device(FOs:Functional Units) are connected to the Internet. STONE discovers FOs transparently and adaptively, then combines FOs to compose an environment-aware service. This paper presents STONE design goal and architecture. In addition, we showed several indoor context-aware applications implemented on STONE platform.

1　まえがき

今日のインターネットにおいては，ネットワークデバイスやプロトコル群の発達によるネットワーク接続の遍在化，小型高速かつ省電力なデバイステクノロジによる計算資源の遍在化，CDN(Contents Distribution Network) 技術や peer-to-peer 技術の登場によるコンテンツの遍在化が進んでいる．また，研究レベルではあるものの，ネットワーク接続が可能な微少センサ技術 [1]，グローバルで大規模な分散ストレージ技術 [2]，必要なときに動的に計算資源を取り出すメタコンピューティング技術 [3][4] などが登場しつつあり，このような動向の延長線上に，IPv6 が提供する広大なアドレス空間と相まって，身の回りの至る所にネットワーク資源 (Connectivity)，計算資源 (Computing)，コンテンツ資源 (Contents) が存在する環境を容易に期待することができよう．本稿ではこのような資源遍在環境を 3C 環境 (3C Everywhere 環境) と呼び，(1) 多種多様で膨大な数のオブジェクトが存在する，(2) モバイルコンピューティングが標準となる，(3) 物理空間へネットワークが進出する，という 3 つの特徴を備える動的な環境であると考える．

3C 環境において，サービスの提供者とユーザの双方にとって好ましいことは，いつでも，どこでも，思いついたときに，すぐ所望のサービスを提供・利用できるプラットフォームが用意されることである．すなわち，3C 環境では透過性と即時性を兼ね備え，完全に自動設定可能なプラットフォームが必要である．このような環境が構築されることで，はじめて，サービス提供者が既存サービスの利用率をさらに向上させたり，いわゆる「気の利いたサービス」[5] や「状況に応じたサービス」[6][7]，あるいは今後登場するであろう革新的なサービスの提供が可能になる．一方，ユーザ側からは，サービス利用に際しての様々な障害が緩和され，サービスを利用した新たな活動に従事することができよう．

このようなプラットフォームの実現に向けて，現在までに自動設定技術を中心に据えた多くのアーキテクチャが提案されている [8][9][10][11]．しかしながら，これらアーキテクチャの多くは，既存のシステム上に単一のミドルウェアプラットフォームを構築し，このプラットフォーム上で様々なサービスを統合することを目指している．このため，各プラットフォーム間での相互接続性の問題や，プラットフォーム内でもインターフェースの標準化が遅々として進まないなど，理想に向けては多くの課題がとり残されたままになっている．これらアーキテクチャの失敗は，積極的に多様性を認めることを行わず，単一のプラットフォーム上に理想環境を構築しようと試みた点にある．

一方，これまでのインターネットに目を向けると，多種多様なサービスが，IP ネットワークと WWW 周辺技術を DNS(Domain Name System/Service) を介して統合することにより，エンドユーザに提供されてきた．ホスト名とドメイ

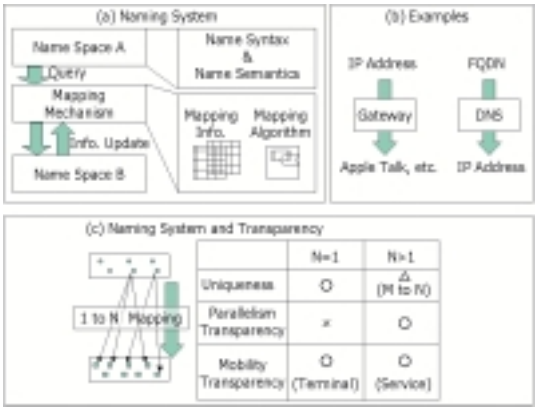

図 1: ネーミングシステムの基本原理

ン名，およびこれに続く論理パスという限定されたセマンティクスとシンタックスではあるものの，DNS を利用することで，WWW 上で提供される多くのサービスに透過性を持って全世界規模でアクセスできることは，ある意味驚嘆に値する．最近では，このような DNS の広域性と透過性を利用し，CDN によってデリバリされたコンテンツへの透過的アクセス，ブロードバンド常時接続サービスの普及に伴う IP 非固定端末への透過的アクセスにも DNS が利用されつつある．このような動きは，膨大な量のコンテンツやデバイスが遍在し，しかもこれらが動的に変化する 3C 環境おいてはさらに顕著になるものと考えられる．しかしながら，後述するように，これらの新たな DNS の利用形態は，FQDN(Fully Qualified Domain Name) と IP アドレスの比較的静的な 1 対 1 マッピングを目的とした DNS 本来のデザインとそぐわないために，新たな問題を生むおそれがある [12][13].

このような点に鑑みると，現行の DNS に改良を重ねて様々な利用形態に対応する，あるいは DNS とは異なるネーミングシステムを新たに導入する，いずれにしても来るべき 3C 環境の利用形態に合った透過性を提供し得るネーミングのフレームワークとデザイン指針が絶対的に必要であると筆者らは考えている．このような考えに基づき，それぞれの視点は多様であるものの，研究レベルではネットワークとネーミングに関する新たなフレームワークの提案も行われつつある [14][15][12].

このような観点から，本稿では 3C 環境を想定し，様々なサービスの透過的，即時的利用をサポートするためのネーミングシステムの一デザインと，その初期的実装について以下のような構成で述べる．まず，2. ではネーミングシステムによる透過性を実現するための基本原理について概説し，ネーミングシステムデザインの指針と課題を現行の DNS を題材に示す．また，ネーミングシステムが 3C 環境において果たすべき役割を明確にする．次いで，3. では 3C 環境において多種多様なオブジェクトを透過的に発見，接続することで，サービスを透過的，かつ即時的に提供するシステムのデザインとアーキテクチャについて述べる．4. では本稿で述べるシステムの中核をなすネーミングシステムについて説明する．さらに，5. ではシステムの初期的実装結果と，システムを用いたアプリケーション例を示し，6. で関連研究との比較を行う．

2 ネーミングシステム

2.1 名前と透過性

図 1(a) にネーミングシステムの基本原理を示す．ネーミングシステムはマッピング元名前空間（図では名前空間 A）とマッピング先名前空間（図では名前空間 B）の 2 つの名前空間と，これらの間のマッピング機構から構成される．ここで，名前空間はセマンティクスとシンタックスによって規定される有限あるいは無限の空間であり，その記述がテキストであるか否かは問わない．通常，マッピング先空間からはマッピング先空間の名前に対して，マッピングの際の判断基準に必要な情報が提供される．マッピング機構は 2 つの空間を変換するためのマッピング情報と，マッピングを行うためのアルゴリズムから構成される．これら 2 つの要素は集中システムあるいは分散システムとして実装され，具体的なネーミングシステムを形成する．

図 1(b) に名前空間とマッピング機構の例を示す．たとえば，空間 A が IP アドレス，空間 B が別のネットワーク層プロトコルのアドレスとすれば，マッピング機構はゲートウェイに相当し，空間 A が FQDN，空間 B が IP アドレスであれば，マッピング機構としては DNS が相当する．このようなモデルはインターネットに代表される分散システム上の至る所で見ることができ，したがって分散システムは多種多様なネーミングシステムの集合によって構成されていると考えることもできる．

ネーミングシステムは，2 つの名前空間とそののマッピング機構のデザインを様々に工夫することによって，分散システムの設計で必須となる透過性を実現することができる．また，マッピング元の名前空間のセマンティクスとシンタックスを適切に設計することにより，マッピング先の名前空間の要素を柔軟に指定するための表現力が提供される．このとき，名前空間に対して必要以上に表現力を与えるようなセマンティクスとシンタックスを規定してしまうと，マッピング機構の負荷が大きくなり，逆にセマンティクスとシンタックスが必要な表現力を提供できない場合には，十分な透過性を提供できなくなってしまう．このため，ネーミングシステムを設計する際には，どのような透過性と表現力を名前に期待するのかという，ネーミングの目的を十分に明確にした上で設計を行う必要がある．

ネーミングシステムで実現可能な透過性は大きく分けて，並列性を隠蔽するための透過性（ここでは並列透過性と呼ぶ）と，移動性を隠蔽するための透過性（ここでは移動透過性と呼ぶ）に分けられる．並列透過性は，同一オブジェクトが複数存在する状況を隠蔽し，複数存在するオブジェクトに対して透過的なアクセスを提供する複製透過性，複数あるオブジェクトの一部に障害が発生しても代わりのオブジェクトにアクセスできる障害透過性，オブジェクトの負荷に応じて適切なオブジェクトにアクセスする負荷透過性などからなる．一方，移動透過性は，一意のオブジェクトアドレス変化を隠蔽する，いわゆるターミナルモビリティ透過性と，デバイスやコンテンツの変化を隠蔽することでサービス単位での透過性を実現するサービスモビリティ透過性に分類できる．

図 1(b) にネーミングシステムと透過性の関係を示す．ネーミングシステムにおけるマッピングはマッピング元とマッピング先の関係が 1 対 N のモデルとなる．このとき，N=1 とするか，N>1 とするかによって提供できる透過性に制限が加わる．

N=1 モデルの場合，マッピング元とマッピング先が 1 対 1 で対応するモデルになり，その最大の特長は，マッピング元の名前に一意性を期待することができる点である．たとえば DNS は広大なインターネット上でホストを一意に特定する目的で設計されているために，本来はこのモデルに属する．しかしながら，一意性を期待することは並列透過性の考え方と相いれないため，N=1 モデルでは並列透過性の実現が困難となる．移動透過性については，一意なオブジェクトのモビリティ（ターミナルモビリティなど）を隠蔽する目的で N=1 モデルを利用できる．たとえば，Dynamic DNS や Mobile IP などが N=1 モデルでの移動透過性に対応する．

N>1 モデルの場合，マッピング元とマッピング先の関係は 1 対多になる．このためマッピング先の一意性を完全に保証す

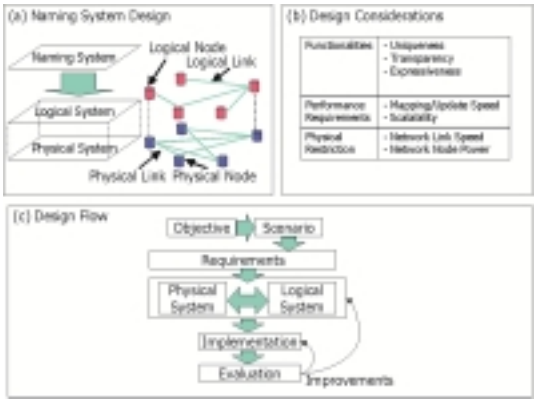

図 2: ネーミングシステムのデザイン

ることは，本来は不可能である．しかし，マッピング元の名前を複数使うこと (M 対 N マッピング) によって，任意の確率で一意性を保証することはできる．なお，名前空間の設計にもよるが，N 対 M マッピングで高い一意性を保証することはマッピング機構の複雑化と高負荷を招くおそれがある．N>1 モデルの最大の特徴は並列透過性を実現できる点にある．すなわち，同じ名前のものであれば等価であるというセマンティクスをマッピング元名前空間に規定し，これをマッピング機構に実際に反映させることによって，複数のマッピング先空間内要素に透過的にアクセスできる．このようにすることで，障害透過性などが実現される．また，マッピングの際の判断基準として，負荷を用いれば，負荷透過性が実現される．移動透過性については，N>1 モデルが一意性保証に適さないという観点から，ターミナルモビリティの実現は難しい．しかし，たとえば，コンテンツマイグレーションにおけるコンテンツアクセスや，移動した先々でのサービスの継続など，サービス単位での移動透過性（サービスモビリティ）が実現できる．

以上述べたように，ネーミングシステムは分散システムでの基本的な要素であり，様々な透過性を実現しうる可能性を秘めるという点で，3C 環境におけるキーテクノロジーの一つである．

2.2 デザインフロー

ネーミングシステムの持つ透過性を十分なパフォーマンスで発揮させるためには，システムの基本原理を現実のシステムへと適用するためのネーミングシステムデザインが必要となる．本節ではネーミングシステムを設計する際の一デザイン指針について述べる．

図 2(a) にネーミングシステムデザインの基本的な考え方を示す．なお，以下では特に断らない限り，分散システム上でのネーミングシステムデザインを考える．

前節で述べたネーミングシステムの構成要素，すなわち 2 つの名前空間と，マッピング情報およびマッピングアルゴリズムは，物理ノード（PC などの演算ノード）と物理リンクからなる物理ネットワークシステムと，論理ノード（ソフトウェアモジュールなど）と論理リンク（ソフトウェアモジュール間の論理的接続など）によって結合される論理ネットワークシステムによって具体化される．このとき，ネーミングシステムに求められる機能・性能用件と物理的制約条件などが課せられ，物理・論理ネットワークシステムがデザインされる．

デザイン時に考慮すべき点を図 2(b) に，デザインの基本的なフローの一例を図 2(c) に示す．ネーミングシステムデザインは一種の分散システムデザインである．分散システムデザインには様々な方法論が存在するが，本フローでは，まずネーミングシステムの目的を定め，ついでネーミングシステムを利用する際のシナリオを決定する．シナリオによりネーミングシステムにおけるマッピングと情報更新のモデルが決まり，このモデルをもとにネーミングシステムの機能・性能要件が決定される．ここで，機能要件とは一意性，透過性，および表現力であり，機能要件に対して名前空間と論理ネットワークシステムがデザインされる．また，性能要件とは，想定するアプリケーションが，どのくらいのスピード，エリアで名前の追加・更新，あるいは名前空間のマッピングを行う必要があるかを定めるものである．性能要件に対して物理ネットワークシステムがこれを満足できなければ機能・性能要件を見直す．機能・性能要件と物理ネットワークシステムの整合性がとれた段階で，ネーミングシステムは物理ネットワークシステム上に実装され，運用・評価される．

これら一連の流れの中で，もっとも難しい点は機能・性能要件を満たすように論理・物理ネットワークシステムをどのように決定するかである．機能要件を満たすように論理ネットワークシステムを設計してしまい，その後，性能要件を満たすように物理ネットワークシステムをデザインする場合も考えられるが，インターネットを対象とする場合には，物理ネットワークシステムがすでに存在するため，これに合わせてネーミングシステムをデザインすることになる．

2.3 DNS と 3C 環境

前節ではネーミングシステムのデザインの概要を述べた．本節では DNS のデザインとその定性的な分析から，3C 環境におけるネーミングシステムのあり方を考える．周知の通り，DNS は本来，Human-readable な FQDN とホストの IP アドレスを，インターネットスケールで，一意に対応づけることを目的としてデザイン・実装がなされてきた．DNS は，名前解決先のホスト IP アドレスは比較的静的であるとした上で，名前の追加・更新や対応する IP アドレスの更新などは各ノード（DNS サーバ）の管理に任せ，名前解決はグローバルで一意に行うというシナリオを想定している．このとき，一意性が機能要件，広域性（スケーラビリティ）が性能要件になる．機能要件に対しては，管理ドメインという観点から名前空間のセマンティクスとシンタックスが設計されており，管理ドメインのもつ階層構造がそのまま論理ネットワークシステムの構造となっている．この論理ネットワークシステムが物理システムであるインターネット上に実装されている．一方，性能要件については，物理ネットワークシステムであるインターネットの構造との整合性がとれるように論理ノードを配置することと，ネームキャッシュと代理サーバの仕組みを導入することによって，実現している．

図 3 に DNS の定性的分析を示す．図 3 において，各グラフの横軸はルートノードからのホップ数を表す．いま，DNS における名前解決時に発生するクエリーサイズおよび名前追加・更新・解決時に発生するクエリーサイズおよび一つのクエリーを処理するために必要な計算資源が図 3(a) のように，全階層において一定サイズとする．図 3(b) に WWW のアプリケーションを想定した場合における DNS の定性的分析を表している．WWW のアプリケーションにおいては，Web サーバのアドレスは基本的には固定と考えられ，このため名前の追加・更新用のクエリーは，ルートからリーフに至るまで少ないと考えることができる．一方，名前解決（マッピング）用のクエリーは，DNS の論理システムの持つ階層構造と，その解決アルゴリズムによってルートノードに近いほど，クエリーが集中する．しかし，ルートノードに集中する傾向のこのクエリーは，Web サーバの名前更新がほとんどないという前提のもとでは，DNS キャッシュによってリーフ寄りに移動させることができる（図 3(b) 上段）．しかし，DNS 実装・運用上の問題

などにより，ルートノードへのクエリー集中の傾向はあり，これに対しては，たとえばルートノードへのクエリーを代理サーバへ委譲することなどにより，見かけ上ルートノードを増やして負荷分散が行われている（図 3(b) 中段）．一方，ノード間をつなぐ物理ネットワークのリンクに関しては，クエリーが集中しやすいルートノードほどバックボーンに近くなるように配置されているため，物理ネットワークとの親和性は非常に高くなっている（図 3(b) 下段）．以上のことから，現在の WWW サービスにおいて，DNS は本来の目的であるインターネットスケールでの Human-readable な FQDN と IP アドレスのマッピングという利用に非常に重要な役割を，十分な実用度で行っている．

しかしながら，最近では，このような DNS の広域性と透過性を利用し，CDN によってデリバリされたコンテンツへの負荷分散的アクセス，ブロードバンド常時接続サービスの普及に伴う IP 非固定端末が提供するサービスへの透過的アクセスにも DNS が利用されつつある．DNS が本来，図 1(c) における N=1 モデルをもとにデザインされていることを考えると，このような並列透過性やサービスモビリティを実現しようという試みは本質的に相いれないものである．このため，CDN サービスにおけるキャッシュの有効期限とルートノードの負荷の問題や，リーフノードにおける DNS サーバへの更新クエリー増加などの問題が発生しつつある．

このような問題に対しては，現状の Web ベースのサービスを想定する場合，たとえばサーバ複製技術などによりある程度は対応可能であろう．しかしながら，本稿で想定する 3C 環境は，多種多様なオブジェクトが無数に存在し，モバイルコンピューティングが標準であり，我々が生活している物理空間へもネットワークが進出する環境である．このような環境で現状の DNS を利用した場合のシナリオを図 3(c) に示す．3C 環境では，多種多様で膨大な量のコンテンツやデバイスの存在と，それらオブジェクトのモビリティによって，名前追加・更新用のクエリーが WWW 想定時に比べて，大幅に増加することが予想できる．また，現状の単一 Web ページが多数のコンテンツから構成されるように，3C 環境では膨大なオブジェクト同士が互いに互いを利用し合うことでサービスが構築されると考えられ，名前解決用のクエリーの量も膨大になると考えられる（図 3(c) 上段）．しかも，動的な環境であるために，DNS キャッシュの有効期限は短くなる傾向になると予想され，ルートノードにおいてもクエリーの量が大幅に増加するであろう．これに対し，DNS のノード数は管理ドメインを記述するというセマンティクスから，管理ドメインの大きさには自ずと限界が生じ，リーフノード数は増加するであろうが，各リーフが管理するオブジェクト数は現在のそれよりも大幅に増加すると考えられる．このため，ルートノード，リーフノードのそれぞれにおいてノードの負荷が上昇し，ネーミングシステムのパフォーマンス，すなわち名前更新・解決のスピードが大幅に低下するおそれがある．一方，ネットワークに関しては，光 WDM 技術などの進歩によって，アクセス側，バックボーン側双方において十分な帯域の確保が可能になると考えられるが，名前追加・更新・解決クエリーが総トラフィック量に占める割合は，現在よりも増加するものと思われる（図 3(c) 下段）．

さらに，3C 環境における DNS の決定的な問題な点は，DNS の名前が管理ドメインというセマンティクスとシンタックスにより規定されている点である．たとえば，ネットワーク化された無数のセンサーを利用するような場合を考える．同一情報の並列度が高くなるセンサーネットワークのようなシステムを利用する際のネーミングでは論理的所属はもはやほとんど意味をなさず，センサーの論理的所属，一意性よりも，むしろ，どのような機能で，どのような地理的空間にあり，その情報をどのように取り出すかという指定が重要になろう．しかしながら，DNS 本来のセマンティクスとシンタックスはこれら

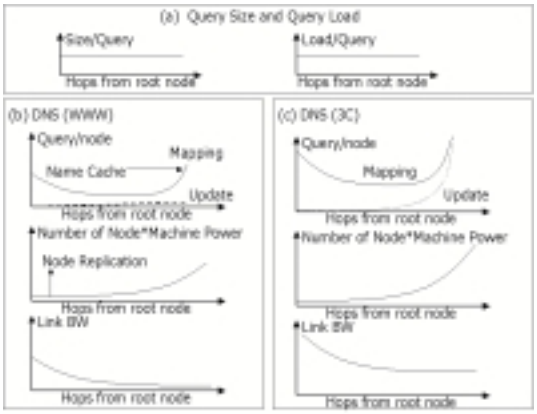

図 3: DNS の定性的分析

要求に対応できるだけの表現力を備えていない．目的もののが何であるかを指定し，これにアクセスする手段は，現在のところ WWW サーチエンジンが主として担っている．しかしながら，集中管理で膨大なオブジェクトの追加・更新・検索を高速に行うことには限界があり，またセンサーネットワークから提供される情報を，Web ドキュメントを対象としたサーチエンジンが適切に扱えるかについては疑問がある．

すなわち，3C 環境においては，アプリケーションシナリオにもよるが，特に現状の DNS および WWW 周辺技術では十分な並列透過性と，物理空間内のオブジェクトを指定するための表現力，および一意ではないオブジェクトのモビリティサポートが十分に行えないと考えられる．このため，3C 環境のアプリケーションシナリオを予想しつつ，必要な機能と性能を備える新たなネーミングシステムが必要となろう．このような観点から，次節以降では，3C 環境において複数のオブジェクトの相互接続によりサービスを合成するアプリケーションシナリオを想定し，そのようなシナリオに適したネーミングシステム，およびそれを利用したサービス合成システムのデザインと実装について述べる．

3 デザイン

3.1 シナリオ

3C 環境では，様々な種類・内容のコンテンツ，センサーなどの各種ハードウェアデバイス，プロキシなどのソフトウェアデバイス，様々なプロトコル・ネットワークインターフェースなど，様々な機能オブジェクトが遍在すると考えることができる．このような環境では，その場その場で適当なコンテンツ，デバイス，プロトコル，ネットワークを使って様々なサービスを，いつでも，どこでも，必要なときにすぐに（即時的に）構成できる条件がととのっていると考えることができる．このとき，サービスを動的に構成するにあたっては，必要な機能を持ち，かつ接続可能なオブジェクトの透過的発見が重要となる．サービスの動的な利用や構成をターゲットとしたアーキテクチャには，SLP[16]，Jini[8], UPnP[11] などがすでに存在する．また，研究レベルでは文献 [17][18] などがすでに存在する．しかしながら，これらのアーキテクチャは，オブジェクトに関する様々な情報をフラットな状態でディレクトリサービスに格納し，集中管理する方式を採用している．ネーミングシステムはこのディレクトリによって実現され，オブジェクトの透過的発見ができる．このような管理方式は，情報家電機器などによって提供されるサービスを組織内の LAN 環境で透過的に利用する場合などには非常に適した方式である．しかしながら，サービスの動的構成を必要とするネットワークアプリ

ケーションはローカルエリアに限定されるものではない．

たとえば，音声ストリーミング技術を用いたネットワーク上のニュース放送（音声ソース機能オブジェクト）とスピーカ（音声出力機能オブジェクト）を相互接続したネットワークラジオサービスを考える．このとき，リスナーが移動しても，居間ではテレビのスピーカ，自室では PC のスピーカのように等価な機能を選択していくことで，ユーザに対しては移動透過性を提供しつつラジオサービスを動的に構成・維持する事ができる（サービスモビリティ）．ここまでは，既存アーキテクチャを利用する事で，十分に実現可能である．しかしながら，モバイルコンピューティングが標準的になる 3C 環境では，グローバルエリアにおいても，このようなサービスモビリティのサポートが必要になると考えられる．たとえば，電車内など，音声出力が適切でない環境では，PDA の画面（テキスト表示機能）に，音声データとして到達するニュースをテキスト変換して表示することで，ラジオサービスを新聞サービスへと変化させる必要がある．このとき，音声-テキスト変換プロキシを音声ソースと音声出力間に自動挿入する必要があり，また，場合によってはストリームのビットレートなどを変更する必要も出てくる．このようなシナリオでは，グローバルなエリアで必要な機能オブジェクトを発見し，適切に接続する必要がある．また，グローバルエリアにおいてサービスの動的構成を行うためには，ネットワークの状況に応じて，機能オブジェクトの選択がなされる必要がある．しかしながら，前述の既存アーキテクチャは，ディレクトリサービスがネーミングシステムとしての役割を担うため，グローバルエリアでは明示的にドメインを指定する必要がある．このような指定は，2. で述べた N=1 モデルを利用することになり，並列透過性の実現が本質的に難しくなる．また，ディレクトリ内での名前空間の階層化や集約化については言及されておらず，したがって，名前空間はフラットな構造を持つことになる．フラットな構造を持つ名前空間はスケーラビリティに乏しく，グローバルエリアへの適用は難しい．このため 3C 環境において様々なアプリケーションをサポートしようとするネットワークは，ネットワークとの親和性が高く，サービス構成・維持に必要な機能オブジェクトの透過的発見ができる，N>1 型のネーミングシステムを持つ必要がある．

筆者らは，上記能力を持つネーミングシステムの一デザインを，機能オブジェクトの接続という観点から行い，ネーミングシステムを利用してサービスを動的に構成するシステムである STONE(Service synThesizer On the NEt) の設計・実装を行っている [19][20][21][22]．STONE で用いられるネーミングシステムは，機能オブジェクト間の接続には，機能オブジェクト間でインターフェースが一致する必要があるという考え方から，機能オブジェクトのインターフェースを名前として利用し，インターフェースの持つ集約性を利用することでスケーラビリティの確保をねらっている．以下では，STONE システム全体のデザインをネーミングシステムを中心に述べる．

3.2 STONE システムアーキテクチャ

図 4 に STONE アーキテクチャを示す．STONE は “機能オブジェクト (FO:Functional Object)”，“サービスグラフ (SG:Service Graph)”，および “サービスリゾルバ (SR:Service Resolver)” の 3 つの要素から構成される．

FO はサービスを構成する機能の単位である．ひとつの FO はネットワーク上で単独の機能として存在し，ネットワークを介してデータを受け取り，そのデータに対してに FO 固有の処理を行い，再びネットワークへと情報を送出する働きをもつ．FO は STONE アーキテクチャ内で自身の持つ機能を表現するための名前を持つ．FO は，この名前を SR に登録することで，ユーザや他の FO から参照して利用することができる．

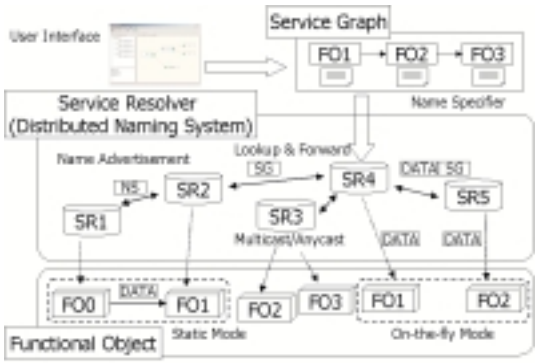

図 4: アーキテクチャ

STONE におけるサービスは，複数の FO 間でのデータの送受と FO におけるデータ処理により実現される．このとき，FO 間の相互関係を記述したものが SG である．すなわち，SG はサービスを構築するための設計図としての役割を持っている．SG 内の FO は，FO を参照するための名前によって記述されている．STONE におけるすべてのサービスは SG により記述される．SG はユーザにあらかじめ与えられていても良いし，ユーザが適切な UI を用いて独自に作成しても良い．

SR は FO の名前の登録・管理を行うとともに，SG に記述されている FO を発見・接続する事でサービスを具体化する機構である．SR は複数個が論理ネットワーク接続されることによりサービスリゾルバネットワーク (SRN:Service Resolver Network) を構成する．サービスリゾルバの基幹部分は SG に記述された FO の名前を管理・解決する分散型ネーミングシステムと，名前解決と同時にアプリケーションで用いられるデータを配送する機構である．SR はクライアントより SG を受信し，SG 内でまず最初に発見すべき FO の名前をクエリーとして求める．なお，複数の FO の発見順序は SG 内の FO をすべて発見するためのコスト（発見速度など）に大きな影響を与えるが，現在の STONE の実装では発見順序は SG に明示的に与えている．当該 SR がクエリーに適合する FO を持つ場合にはその FO へ解決する．適合する機能が当該 SR にない場合には，SR に実装される名前解決アルゴリズムを用いて，適切な隣接 SR にクエリーをフォワードする．SG 内の最初の FO が解決されると，次に発見すべき FO の名前をクエリーとして求め，同様にして名前解決が行われる．SR は名前解決が行われた FO に対して接続命令を発行し，FO はこれを受けて接続先となる FO と，機能間接続プロトコル [23] を持ちいて接続する．このような発見・接続形態を STONE では “スタティックモード” のサービス合成と呼ぶ．一方，SG をヘッダとして，アプリケーションで用いられるデータを配送する場合には，FO への解決と同時に，SR が当該 FO に対して直接データを送付し，当該 FO から出力されるデータに再度 SG を付け，適切な隣接 SR にフォワードされる．このような形態を STONE では “On-the-fly(OTF) モード” のサービス合成と呼ぶ．SR はさらに，名前に基づくマルチキャストやエニーキャストの機能を持っており，FO に付けられた位置非依存の名前と On-the-fly モードによる名前解決により，様々なメトリックでのデータ配送・処理形態をサポートする．このようなデータ配送・処理形態により STONE は様々なサービスを柔軟に生成することができる．

図 5: ネーミング

4 ネーミングシステムアーキテクチャ

4.1 デザイン

STONE のネーミングシステムの目的はグローバルなエリアまでを対象とし，サービスを合成する際に必要で，接続可能であり，かつ負荷などの面で物理システム的にも適切な FO に並列透過性，サービスモビリティをサポートしつつアクセスできることである．具体的なシナリオとしては，3.1 で述べたシナリオのような利用形態を考える．これに対して，必要な機能要件は，(1) 並列透過性およびサービスモビリティのサポート，(2) 接続可能で必要な機能を持つ FO を指定できる名前のセマンティクスとシンタックスである．また，性能要件としては (3) グローバルにスケールすることがあげられる．対象とする物理ネットワークシステムはインターネットを想定する．

STONE ネーミングシステムでは，(1) の要件に対して，N>1 型の位置非依存ネーミングを使用し，名前解決の際に負荷などを考慮して行えるようにメトリックを設定できるようにしている．また，FO が移動してもこれを隠蔽するために，名前解決は hop-by-hop に行うようにしている．(2) の要件に関しては，FO が提供する機能を入出力インターフェースとその関係によって記述する．

図 5 に FO を指定する際の名前を示す．FO の名前は領域名指定子，インターフェース名指定子，およびコンテキスト名指定子の 3 つの名前から構成される．領域名指定子は，DNS などを用いて FO が存在するドメインなどを明示的に指定する場合に用いられる．インターフェース名指定子は STONE アーキテクチャで最も重要なインターフェースによって作られる名前空間の要素を指定するために用いられる．STONE における FO は入力，出力のインターフェースを持ち，それらインターフェース間の演算関係で記述される．インターフェース名はデータフォーマットと通信手段，および演算によって記述される．後述するように，FO の名前解決は，このインターフェース指定に基づいて，サービスリゾルバ間をクエリーがルーティングされることで行われる（インターフェースルーティング）．最後のコンテキスト名指定子はたとえば「地理的に最も近い FO を選択」,「時刻 t の場合には FO2 を選択」などの条件が記述される．

(3) の要件は (3-1) 物理システムとの親和性，(3-2) スケーラブルな名前登録・更新・解決方法の面で設計を行っている．(3-1) に対しては水平分散型のサブメッシュ構造を Gunuetlla の関連技術で用いられる最適化アルゴリズムを導入することで自動構築することで，物理ネットワークと親和性の高い論理システムを構築する [24]．また，負荷分散の観点から，FO の登録を物理ノードの負荷状態を考慮して行えるようにしてある．(3-2) に関しては，名前登録・更新にソフトステートなメカニズムを採用し，また，論理システム内での名前情報を数値によって集約管理する仕組みを導入する．さらに，名前情報は，変化の少なくいインターフェース名のみを論理システム内で共有し，比較的変化の多いコンテキスト名は論理ノード内に保存する．

このような基本デザインに基づいた STONE のネーミングシステムの詳細を以下に述べる．

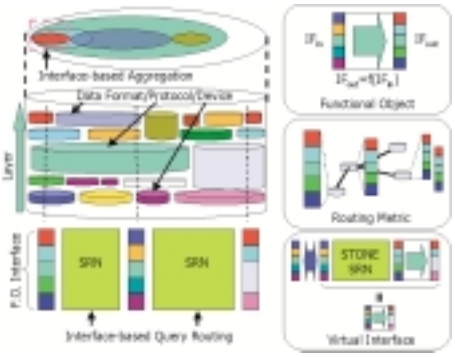

図 6: ネーミングシステムアーキテクチャ

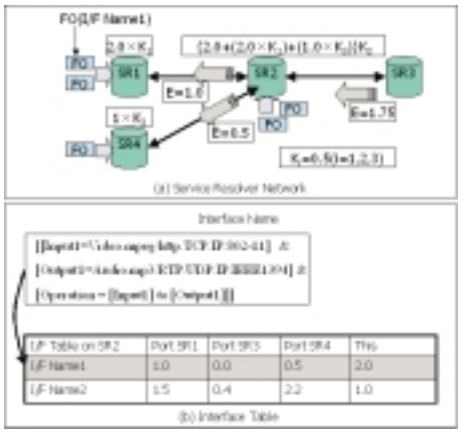

図 7: 名前登録管理および解決機構

4.2 名前の登録と管理

STONE で用いられるネーミングシステムでは，SRN に登録されているインターフェース名（以下 I/F 名）を数値で集約管理し，FO への接続性と評価値に基づいてクエリーをフォワードするアルゴリズムを用いる．図 6 にネーミングシステムアーキテクチャの概要を，図 7 に SRN における名前管理・解決アルゴリズムの概要を示す．STONE システム上に存在する機能は，必ずひとつ以上の SR に名前の登録を行う．どの SR に登録を行うかは FO の実装依存である．また，SR はひとつ以上の SR と論理コネクションを確立する．この論理コネクションは動的に確立できるものとする．どの SR と論理コネクションを確立するかは実装依存である．ここでは，ループが存在しないスパニングツリーで SR の論理ネットワークが構成され，Gnutella の関連技術である Reflector[24] と類似のアルゴリズムによって，物理ネットワークシステムとの親和性がとられるものとする．

SR に登録される FO の名前のうち，I/F 名は各 SR で管理されているインターフェースルーティングテーブルに登録される．一方，コンテキスト名は SR 内のローカルデータベースにフラットな状態で登録される．図 7(b) に SR2 のインターフェースルーティングテーブルの概要を示す．インターフェースルーティングテーブルは，I/F 名に対して，SR 自身 (This フィールド) および SR が確立している論理コネクション（ポートフィールド）に対する評価値を保持する．評価値をどのように設定するかは実装依存であるが，ここでは同一の I/F 名を持つ機能の個数とする．なお，評価値に対して FO の負荷状態，物理ネットワーク的最短経路などを反映させることも可能である．

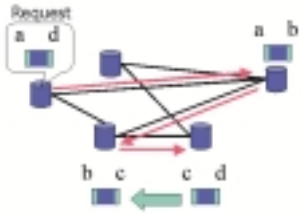

図 8: ヴァーチャルインターフェース

たとえば，InterfaceName1 の I/F 名を持つ機能が 2 個 SR1 に登録された場合には，SR1 が管理する I/F テーブル中の対応する場所において This フィールドの値が 2 加算される．This フィールドはそのテーブルを管理する SR 内に当該 I/F 名を持つ FO が存在することを示している．SR1 は This フィールドに対応する評価値を隣接している SR2 に対して非同期で広告する．広告する際には，SR1 のポリシーにより決定される数値 $K_1(<1)$ を評価値に乗じ，SR1 が確立している論理コネクションすべてに広告する（ここでは 1 つ）．図 7(a) では $K_i = 0.5(i = 1, 2, 3)$ の場合が示されている．

広告を受けた SR2 は自身が管理するテーブル内の対応する場所において，SR1 方向のポートに対応するポートフィールドに数値 $E = 1.0$ を書き込む．また SR4 方向にも FO が存在するため，SR4 方向から広告された数値 $E = 0.5$ を SR4 に対応するポートフィールドに書き込む．SR2 は SR3 方向に SR2 における This フィールドの値 (2)，SR1 方向のポートの値 (1.0)，および SR4 方向のポートの値 (0.5) を足し，数値 $K_2(<1)$ を乗じて他のポートに広告する．この結果 SR3 からみると，各 SR に登録された FO は $E = 1.75$ で見えることになる．

なお，FO は定期的に SR に登録更新を行い，期限内に更新が行われない場合には，FO を直接管理する SR は当該 FO を抹消すると共に，その FO に対応する評価値をインターフェースルーティングテーブル上で減算する．このようにすることで FO が移動などによって別の SR に登録した場合でも，その変化を隠蔽することができる．また，障害等によりある SR の論理コネクションが切断された場合には，その SR 方向のポートフィールドがインターフェースルーティングテーブルから抹消される．

以上の動作が各 SR において非同期で繰り返されることにより，テーブルの動的管理が行われ，I/F 名が評価値の形で共有される．評価値は I/F 名の近さ（その I/F 名を持つ FO にたどり着くまでに経由する SR 数）と，その I/F 名を持つ FO の数を畳み込んでいる．すなわち，ある I/F について，あるポート方向の数値が高いということは，そのポート方向にその I/F 名をもつ FO の数が多く，しかも FO にたどり着くまでに経由する SR 数が少ないということを示している．

4.3 ヴァーチャルインターフェース

STONE では，ある 2 つの FO 間を接続する際に，評価値に加えて，その接続性についても考慮して行われる．この接続性はヴァーチャルインターフェースによって拡張が行われ，本来は直接つながらない機能同士を，変換機能を自動的に挿入することで接続可能になる．ヴァーチャルインターフェースは，たとえば，前述のシナリオにおいて，音声出力をテキスト出力に変更する際に，音声-テキスト変換機能を自動挿入する場合などに利用される．

図 8 にヴァーチャルインターフェースの概念を示す．いま，図に示すように a → b, b → c, c → d の 3 つの FO が存在する場合を考える，このときそれぞれの FO は SR に対して登録を行い，前述の登録管理アルゴリズムによって広告が行われていく．このとき各 FO 間が接続可能で，かつ各 FO が接続を認めた場合，複数の FO をまとめて一つの FO と見なすことができる．このようにして構成される FO をヴァーチャルインターフェースを持つ FO とし，新たな FO と同様に広告が行われる．このようにすることで，本来は存在しない単体の a → d の FO があたかも存在するかのように，見せることができる．

4.4 名前解決

再び，図 7 に戻り，名前解決アルゴリズムについて述べる．ここでは SR3 において InterfaceName1 の名前解決要求が発行されるとする (領域指定名はないものとする)．SR3 はテーブルを検索し接続性と評価値を前述の方法によって計算し，その後 This フィールドの値をチェックする．This フィールドが 1 以上，すなわち SR3 に FO が 1 つ以上登録されている場合には，コンテキスト名を用いてそれにマッチする FO が存在するかどうかを調べる．コンテキストにマッチする FO がヒットした場合にはその FO にバインドする．ここでは SR3 に所望の FO が登録されていなかったとする．この場合，SR3 はテーブル内のポートフィールド内で，もっとも評価値の高いポート（ここでは 1 つしかないので $E = 1.75$ を持つ SR2 となる）に名前解決要求をフォワードする．SR2 は名前解決要求を受け，SR3 で行われた処理と同様の処理を再度行う．SR2 で解決できなかった場合には，評価値 $E = 1.0$ を持つ SR1 方向に名前解決要求をフォワードする．SR1 においても解決できない場合には SR2 に戻り，SR2 において 2 番目に評価値の高い SR4 へと名前解決要求をフォワードする．なお，SG には経由した SR の情報が書き込まれており，SR がこれをチェックすることで一旦探索を行った SR を再度探索することが無いようになっている．このようにして，名前解決要求がテーブルの評価値に基づいて SRN 上でフォワードされていく．このとき要求には TTL(Time To Live) を設け，所定数の SR を経由しても FO が発見できなかった場合には解決失敗のエラーを，要求発行元 SR に接続されているクライアントアプリケーションに返す．

このような解決アルゴリズムは，できるだけ数が多く，SR 経由数が少なく，しかも接続可能な FO の名前解決を試みる性質を持っている．このような性質は FO の動的接続，特に OTF モードで FO を動的にバインドする際に都合がよい．しかしながら，I/F 名の数が多いポート方向に探しに行くという性質上，スパニングツリーのトポロジーでは，バインドされにくい（発見候補になりにくい）FO が存在してしまう．この問題に対しては，SR がすべてのポートに名前解決要求のフラッディングを行う手法が考えられるが，この手法では SR や下位ネットワークの負荷が増大するため好ましくない．このため，たとえば確率的にフォワードする方向を決定したり，ネームキャッシュを導入したり，FO がアクセス頻度に応じて登録先 SR を変更したり，あるいは SRN に複数の経路を設けたりする方法を本稿で述べたアルゴリズムと組み合わせる方法を検討していく必要がある．

5 実装

5.1 サービスリゾルバの実装

現在筆者らは，機能ユニット，サービスグラフ，サービスリゾルバ，クライアントアプリケーション (UI)，スタティックおよび OTF モードを利用したアプリケーションの実装を行っている．

実装において SR のすべてのコンポーネントは Java2 SDK 1.3 Standard Edition を用いて記述され，Microsoft Windows

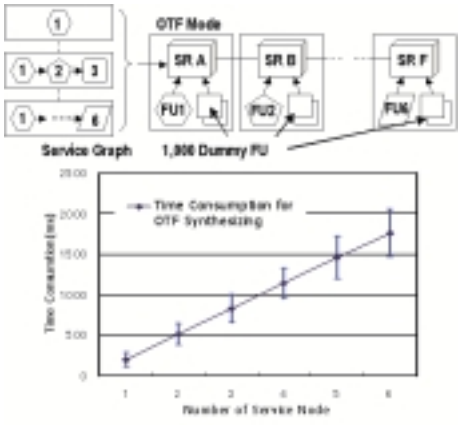

図 9: サービスリゾルバ基本性能

2000 の PC を 6 台用いて動作させている．SR は 2. で述べたスタティックモードと OTF モードの双方のサービス合成モードをサポートする．FO としてはビデオ入出力ソフトウェア，音声入出力ソフトウェア，および PC をフロントエンドとした家電機器などを実装した．ソフトウェア機能の FO の実装には，主として Java Media Framework のライブラリを使用した．また，家電機器などのハードウェア機能は，デジタル I/O ボードや AD/DA ボードを Microsoft Visual Basic を介して制御している．FO の実行環境においては，FO のリソース消費に応じて，PC 一台あたり FO1 個〜10 個を割り当てた．FO が動作する PC の OS には Windows 2000 と Linux を使用した．SR および FO 間の接続には 100Mbps の LAN を利用し，各 PC は IP で通信を行う．

ユーザはクライアントアプリケーションを用いてサービスリゾルバネットワークにサービスの合成を要求する．現在実装されているクライアントアプリケーションは，SRN から FO に関する情報を検索する機能を持ち，ユーザがさまざまな SG を作成できるような GUI ベースのエディタである．エディタは SG を XML として出力し，SR が提供する API を介して SR に渡される．このとき，ユーザが SG 内で使用する FO をあらかじめ一意に定めておきたい場合（たとえば映像出力先を明示指定する場合）などがあり得るため，SR がクライアントアプリケーションに提供する API として，I/F 名に基づく FO 発見かフラッディングによる FO 発見かの指定，発見する範囲の指定 (発見までに経由する SR ホップ数の最大値の指定)，条件に合致する FO を 1 つだけ発見するか，あるいはすべて発見するかの指定などの機能検索用 API が提供されている．また，サービス合成時のエラー処理，サービス合成状況の問い合わせなどを行うための API も提供される．

SR と FO 間には，FO 登録・抹消処理，スタティックモードにおける機能間接続指示発行，OTF モードにおけるデータの送受などを行うための基本的な API が定義されている．また，現在の実装では，FO において生ずるさまざまな障害は，クライアントアプリケーションに通知されるため，FO とクライアント間にはエラー通知用の API が定義されている．

まず，FO は起動すると DHCP を利用してネットワークのブートストラップを行う．ネットワークに接続可能となった FO は SR を発見し，自身を SR に登録する必要がある．このとき，SR の発見には Jini で用いられているマルチキャストディスカバリなどの仕組みを用いることができるが，本実装では Gnutella の Host Cache[24] のメカニズムを採用したブートストラップ用サーバ (DSR: Domain Service Resolver) を別途用意し，そのアドレスを DHCP オプションによって与える方式を採用している．登録先 SR の決定にはランダムな決定，

図 10: アプリケーション実装例

Ping 応答速度に基づく決定など，さまざまな方式が考えられる．本実装においては，ローカルエリアの実装という観点から，SR の負荷を分散させる目的で，平均 CPU 負荷状態が最も低い SR を選択するポリシーを採用している．なお，1 つの FO の登録先 SR は複数決定可能であり，複数接続を行う場合には SR 負荷の低い順に選択を行う．FO の登録処理は Jini などが採用しているソフトステートな登録・変更・抹消を採用している．登録が完了した FO はサービスリゾルバあるいは他の FO からのデータ処理依頼を待つ．また，それと同時に処理依頼の頻度などを観測し，必要に応じて登録先 SR の変更などの最適化処理を行う．

一方，SR は FO と同様の仕組みで SR の発見と選択を行う．その後，選択された SR 群に対して論理コネクションを確立することで SRN 内のサービスリゾルバとして動作を開始する．その後，SR は FO の登録・更新・抹消処理，インターフェーステーブルの交換，サービス合成処理，SRN 最適化処理（論理コネクションの変更，SR の分割・統合処理など）をイベントに応じて行う．

実装したサービスリゾルバのサービス合成能力に関する簡単な基本性能評価を行った．評価結果を図 9 に示す．本評価では 6 台の SR A〜F にあらかじめ 1000 個のダミーの FO を登録し，さらに SR A〜F に FO 1〜6 をそれぞれ登録した．SRN 上でインターフェーステーブルが交換され，各 SR の保持する情報が定常状態になった後，FO が 1 個〜6 個で構成される SG を SR A に投入し，サービスが OTF モードで合成されるまでの所要時間を 100 回測定した．すなわち，FO の縦続接続数を 1 個〜6 個まで可変したときのサービスリゾルバネットワークのスループットを測定した．

測定結果において，各点は所要時間の平均値，交差線は所要時間の標準偏差である．測定結果より，1 つの FO を解決し，FO に命令を発行するまで平均約 200ms を要し，FO-FO 間の伝送には平均約 310ms を要する結果となった．FO-FO 間伝送に関する所要時間が大きい理由としては，抽象度の高い Java 処理系の I/O スピードがボトルネックになっているためと考えられる．また，サービス構成の所要時間に大きな分散がある原因は，SRN がアプリケーションレイヤのフォワーディングを行うため，実行環境における他プロセスの影響などを受けるためであると考えられる．今後，SRN 上で OTF モードを用いてリアルタイム性の高いサービスを合成する場合には，これら問題の理解と解決が必要となると考えられる．

5.2 アプリケーションの実装

STONE のアプリケーションの一つとして，筆者らは現在 STONE ルームと呼ばれる室内向けアプリケーション実装環境を構築している．図 10 に実装状況を示す．STONE ルームは縦約 8m ×横約 8m ×高さ約 2.5m のスペース内に，小型デバイスなどを取り付けられるようなトラスを設置し，各種マルチメディアコンテンツの入出力機器，センサ類が設置されている．また，既存の AV 機器類をネットワークから制御するために，赤外線トランスミッタを室内に配置し，赤外線リモコン信号をこれらトランスミッタから全室内に放射するシステムを実装した．さらに，室内環境で様々なアプリケーションを作成するためには，特に地理的位置情報のサポートが重要であり，超音波を用いた屋内測位システム [27] や，物理オブジェクトとユーザ間のインタラクションから位置情報を得る ALS(Adjacency-based Location System)[26] などが実装されている．また，STONE のサービスはすべて SG により作成を行う必要があるため，SG を作成するためのツールを開発している．

現状の STONE ルーム内でデモンストレーション可能なアプリケーションは，無数の FO や SR を想定した大規模なものではなく，我々の身の回りにあるような機材を中心とした情報家電的なアプリケーションが多い．本稿では実装したアプリケーションのうち，On-the-fly モードを利用したアプリケーションとしてモバイルプレゼンテーションサービス，およびエンコーダエニーキャストサービスについて概説する．

モバイルプレゼンテーションサービス：モバイルプレゼンテーションサービスは STONE の On-the-fly モードを利用した典型的なアプリケーションである．On-the-fly モードは，SG による FO の発見と，データ配送を同時に行えるため，柔軟なサービスを合成することができる．たとえば，ある同じコンテキスト名（たとえば所属グループ名）をもつ画像表示機能を提供する FO に対してマルチキャストやエニーキャストを行うことができる．その反面，音声などジッタや遅延に敏感なアプリケーションには適さない．モバイルプレゼンテーションサービスでは，プレゼンテーション画像を提供する画像出力 FO と，これを受信する画像入力 FO を SG に記述し，プレゼンテーション用の画像を SG と共に STONE の On-the-fly モードで転送することで実現されている．画像出力用の FO には，画像選択用のインターフェースも持たせており，このインターフェースとプレゼンテータの端末の GUI ボタン FO をスタティックモードで合成している．On-th-fly モードを使うため，位置情報に応じてプレゼンテータの最寄りのディスプレイに画像を表示すると同時に,「聴衆」というコンテキスト名でグループ化されるデバイスの画面にこの画像をマルチキャストする事ができるようになっている．

エンコーダエニーキャストサービス：エンコーダエニーキャストサービスも，STONE の On-the-fly モードを利用したアプリケーションである．エンコーダエニーキャストサービスでは，ある複数のマルチメディアコンテンツをほかのフォーマットのファイルにエンコードしたい場合に，それぞれのコンテンツに対してエンコードサービスを実現するための SG（コンテンツ出力 FO，エンコーディング用 FO，およびコンテンツ保存用 FO から成る SG）をヘッダとして付与し SRN に送出する．このとき，エンコーディング用 FO に，SR のエンコーディング機能の付加状態をメトリックとしたエニーキャストオブションを付け，サービスリゾルバネットワークに送出することで，エンコーディング処理の並列化を行うサービスである．エンコーディング用 FO はそれを実行する PC の CPU 負荷状態を評価値として広告しており，SR は負荷条件を満たす FO に対しランダムにエンコーディング対象のンテンツを配送する．エンコードされたコンテンツは，SG に記載されている場所に自動保存される．

6 関連研究

現在，Jini, UPnP など数多くのサービスプラットフォームが提案されている．これらは自動設定技術，相互接続性などを中心としたミドルウェアである．また，サービスを合成するという観点からは，DANSE[28], VNA[18]，Hoddes らの研究 [17][29]，Ninja[30] などがある．しかしながらこれらのプラットフォームや研究では，透過性を提供するネーミングシステムを，基本的にディレクトリサービスとして考えている．これに対し，STONE のネーミングシステムは，ネットワークとの親和性，インターフェースに着目した名前空間の集約管理，名前に基づくデータ配送などを考慮して設計されている．

ネーミングシステムを用いて様々な透過性を実現しようとする試みとして INS[14] がある．INS は STONE と同様に水平分散型のネーミングシステムを利用し，名前のセマンティクスについては規定せず，最大限の表現力を持たせる点に特徴がある．しかしながら，INS では表現力を尊重した結果，フラットな名前空間をすべてのノードで共有しなければならないという問題を抱えている．このため INS では共有コスト削減よりも，共有した名前の高速検索に注力している．一方 INS のような名前空間全共有型と対極のアプローチとして名前共有を行わない Gnutella[24] のような名前管理方式がある．しかしながら Gunutella のような仕組みでは，フラッディングによる名前解決のコストが非常に高い．これらのアプローチに対し STONE では機能の発見と接続の観点から名前空間を設計し，名前の共有コストや解決コストの削減を試みている．

移動透過なサービスを提供する手段としては，アプリケーションレイヤでのアプローチ，Mobile IP などを用いるネットワークレイヤでのアプローチなどがある．ネットワークレイヤでの移動透過性実現は，移動中のデータ到達の継続性やリアルタイム性が重要となるアプリケーションに適する．これに対し Dynamic DNS や Globe[31] などのアプリケーションレイヤでの移動透過性実現は，データの継続性やリアルタイム性などは考慮せず，広域におけるオブジェクトの IP アドレス変化をスケーラブルに隠蔽することを主な目的としている．これらが提供する移動透過性のターゲットは，一意なオブジェクトの移動透過性である．これに対し，STONE のネーミングでは一意なオブジェクトの移動透過性ではなく，サービスの構成要素に対する移動透過性に焦点を当てている点でこれらの技術や研究とはターゲットが異なっている．

7 まとめ

本稿では 3C 環境を想定し，様々なサービスの透過的，即時的利用をサポートするためのネーミングシステムの一デザインと，それを用いたサービス合成システムである STONE のデザインおよび初期的実装について述べた．本稿で述べたネーミングシステムは，ネットワーク上に遍在する様々な機能の透過的発見と接続によりサービスの透過的合成を実現するためにデザインされ，機能のインターフェースに着目した名前空間の集約管理と解決機構を備えている．また，ヴァーチャルインターフェースにより，機能間接続の際に，変換機能などを自動挿入することができる．ネーミングシステムおよび STONE システムは，STONE ルームと呼ばれる，屋内実装環境において実装が行われ，本稿ではその初期的評価とアプリケーション実装例を示した．

今後は，システムの詳細なパフォーマンスの測定を行うともに，Hash-based の検索アルゴリズム [32][33] を用いた高速な名前解決の方法について検討を行う必要がある．また，現在

の STONE のネーミングシステムは広域における機能の接続という観点からしかデザインが行われていない．これに対し，3C 環境は機能だけでなく，様々なコンテンツも遍在する環境であり，たとえば文献 [34] や [35] などのアプローチを参考にしつつ，コンテンツをどのように扱うかについても検討していく必要があろう．さらに，アプリケーションについては，屋内環境で実装が行われている現状のアプリケーションをさらに充実させるとともに，広域を対象としたコラボレーション系のアプリケーションや，DNS などの既存システムとの連携についても考えていく予定である．

参考文献

[1] J. Kahn, R. Katz and K. Pister, “Mobile Networking for Smart Dust,” Proc. ACM/IEEE Int’l. Conf. MobiCom 99, August 1999.

[2] UC Berkeley Computer Science Division, The OceanStore Project, http://oceanstore.cs.berkeley.edu/ .

[3] SETI@Home Project Web Site: http://setiathome.ssl.berkeley.edu/.

[4] Distributed.net Project Web Site: http://www.distributed.net/.

[5] T. Suda, T. Itao, T. Nakamura and M. Matsuo, ”Adaptive Networking Architecture for Service Emergence,” the Trans. Inst. Electronics Commun. Engineers of Japan (IECEJ), Invited Paper, Vol.J84-B, No.3, pp.310-320, 2001.

[6] Anind K. Dey, “Understanding and Using Context,” To appear in Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing, 5(1), 2001.

[7] A. Hopper, “The Royal Society Clifford Paterson Lcture, 1999 Sentient Computing,” http://www. uk. research. att. com/ ˜ah/, 1999.

[8] Sun Microsystems, Jini Network Technology, http://www.sun.com/jini/ .

[9] HAVi White Paper, http:// www. havi. org/.

[10] Salutation Architecture Specification; http:// www. salutation. org/ specordr.htm.

[11] Microsoft, Universal Plug and Play specification, http://www.upnp.org/ .

[12] M. Gritter, D. R. Cheriton, “An Architecture for Content Routing Support in the Internet,” Proc. Usenix Symposium on Internet Technologies and Systems, March 2001.

[13] Jaeyeon Jung, Emil Sit, Hari Balakrishnan and Robert Morris, “DNS Performance and the Effectiveness of Caching,” Proc. ACM SIGCOMM Internet Measurement Workshop, 2001.

[14] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley, “The design and implementation of an intentional naming system,” Proc. 17th ACM SOSP, December 1999.

[15] A. Vahdat, M. Dahlin, T. Anderson and A. Aggarwal, “Active Names: Flexible Location and Transport of Wide Area Resources,” Proc USENIX Symposium on Internet Technologies and Systems, 1999.

[16] J. Veizades, E. Guttman, C. Perkins and S. Kaplan, ”Service Location Protocol”, IETF, RFC2165, 1997.

[17] T. Hodes and R. Katz, “Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients,” Proc. MOBICOM 97, 1997.

[18] J. Nakazawa, Y. Tobe, and H. Tokuda, “VNA: A Serverless Distributed Architecture for Integrating Information Appliances,” IEICE Trans. July 2001.

[19] 南 正輝, 森川 博之, 青山 友紀, “ネットワークサービスシンセサイザのデザイン”, 電子情報通信学会技術研究報告, IN2001-192, March 2001.

[20] 森川 博之, 南 正輝, 青山 友紀, “STONE: 環境適応型ネットワークサービスプラットフォーム”, 電子情報通信学会技術研究報告, IN2001, May 2001.

[21] 杉田 馨，澤畠 康仁, 南 正輝, 森川 博之, 青山 友紀, “ネットワークサービスシンセサイザにおけるサービス発見・合成システムの設計と実装”, 電子情報通信学会技術研究報告, IN2001-193, March 2001.

[22] 杉田 馨，南 正輝, 森川 博之, 青山 友紀, “ネットワークサービスシンセサイザアーキテクチャの設計と実装”, マルチメディア，分散，協調とモバイル (DICOMO2001) シンポジウム,June 2001.

[23] 澤畠 康仁, 杉田 馨, 南 正輝, 森川 博之, 青山 友紀, “ネットワークサービスシンセサイザにおける機能接続・管理機構の設計と評価”, マルチメディア，分散，協調とモバイル (DICOMO2001) シンポジウム,June 2001.

[24] Andy Oram, “Peer-to-Peer: Harnessing the Power of Disruptive Technologies”, O’Reilly, March 2001.

[25] 大石 佳敬, 南 正輝, 増田 勝, 森川 博之, 青山 友紀, “水平分散型情報管理・検索システムの適応的調整機構に関する一検討”, マルチメディア，分散，協調とモバイル (DICOMO2001) シンポジウム,June 2001.

[26] 南　正輝, 太田垣 淳一，森川 博之，青山 友紀，“隣接関係を利用した位置情報システムの提案と実装”，電子情報通信学会技術研究報告, IN2001, May 2001.

[27] 石　聖弘, 南　正輝，森川　博之，青山　友紀, “An Implementation and Evaluation of Indoor Ultrasonic Tracking System”, 情報処理学会 MBL 研究会，May 2001.

[28] 板生 知子, 松尾　真人,“適応型ネットワーキングサービス環境 DANSE,” 電子情報通信学会論文誌, B Vol. J82-B No.5, 1999.

[29] T. Hodes and R. Katz, “A Document-based Framework for Internet Application Control,” Proc. 2nd USENIX Symposium on Internet Technologies and Systems, 1999.

[30] Steven D. Gribble, et al., “The Ninja Architecture for Robust Internet-Scale Systems and Services”, Special Issue of Computer Networks on Pervasive Computing, 2000.

[31] M. Steen, F. Hauch, P. Homburg and A. Tanenbaum, “Locating Objects in Wide-Area Systems,” IEEE Communications Magazine, January 1998.

[32] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” Proc. ACM SIGCOMM 2001, San Deigo, CA, August 2001.

[33] Ben Y. Zhao, John Kubiatowicz and Anthony Joseph, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing,” UCB Tech. Report UCB/CSD-01-1141, 2001.

[34] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” Proc. ACM SIGCOMM 2001, San Diego, CA, USA, August 2001.

[35] Benjamin Paul Castro, Parviz Chatschik Bisdikian and Maria Papadopouli, “Locating application data across service discovery domains,” Proc. 7th annual international conference on Mobile computing and networking, 2001.