ユビキタス環境を実現するサービス合成とそのアクセス制御

岩尾　忠重，益岡　竜介

ユビキタスコンピュータの概念が提唱されてすでに20年以上が経ち，小型高性能のプロセッサにより複雑な処理をする装置の小型化やネットワークの高速化および無線化，さらには，並列分散化したシステムの構築が可能となったため，ユビキタス環境の実現の可能性が高まってきている．そこで，本稿はユビキタス環境を実現する上で重要な技術について概説する．特に，ユビキタス環境下でのサービス合成およびアクセス制御について解説する．

キーワード：ユビキタス，サービス合成，アクセス制御

1. はじめに

最近，ユビキタスという言葉を新聞やニュース等でよく耳にする．ユビキタスとは，ラテン語で「偏在」という意味である．昨今使われているユビキタスは，ユビキタスコンピューティングを略した言葉である．ユビキタスコンピューティングとは1980年代初頭にXerox Parc研究所のMark Weiser[1]らによってコンピュータのあり方について，三つ目の波として研究されているものである．一つ目の波は複数のユーザが1台の大型コンピュータを使用する環境であり，二つ目の波は，1人が1台のコンピュータを使用する環境である．三つ目の波は，1人をさらに多くのコンピュータが取り囲む環境である．そのような環境下では，様々な装置の中にコンピュータが埋め込まれ，ユーザがコンピュータを意識せずに情報の取得やユーザの行いたいことをいつでもどこでも可能になるという概念である．この概念には，コンピュータが偏在するという意味だけでなく，それらが有機的に接続するネットワークの概念も含まれている．

現在，我々の周りを様々な形でコンピュータが取り巻いている．パソコンやPDAはもとより，冷蔵庫や電子レンジ，炊飯器，テレビ，車，カメラなどあらゆる身の回りの製品に組み込まれ，いまやコンピュータの入っていない電化製品はないといっても過言ではない．しかしながら，今日の環境は，まだユビキタス環境とはいえない．様々な電化製品などのデバイスに埋め込まれたコンピュータは，デバイスをうまく動かすことのみを目的としたものであり，単にユーザは個別にそれらデバイスを使うのみである．デバイスたちが互いに協調し，ユーザのしたいことを助けるといったような，デバイス間の自律的な連携は現在のところまだ実現していない．

真のユビキタス環境を実現するには，物理世界から論理世界へのマッピング，デバイスなどを容易に接続可能にするネットワーク，個人に応じたネットワークやサービス構築，サービスとそのユーザインタフェースが重要であり，ハードウェア，ネットワーク，アプリケーション，ユーザインタフェースなど多岐にわたり，かつそれぞれが相補的，有機的に連携する必要がある．これらの低レイヤから高レイヤまでの技術の複合があって初めて真のユビキタス環境が実現される．

本稿では，ユビキタス環境を実現する上で重要な技術について概説する．節2では，ユビキタス環境の実現に向けた様々な研究の取り組みを紹介する．節3では，ユビキタス環境下でのサービス合成およびセキュリティにフォーカスし，富士通研究所での取り組みについて述べる．

2. ユビキタス環境の実現に向けて

現在，多くの研究者がユビキタス環境の実現に向けて研究を行っている．ハードウェアでは，物理世界を論理世界へマッピングするための研究が行われている．具体的には，RFID（Radio Frequency Identifier）[2,

いわお　ただしげ
㈱富士通研究所 Webテクノロジ研究部
〒211-8588 川崎市中原区上小田中4-1-1
ますおか　りゅうすけ
米国富士通研究所 カレッジパーク研究所
8400 Baltimore Avenue, Suite 302, College Park, MD 20740-2496, USA

3]やセンサネットワーク[4]などがある．物理的な装置を接続するネットワークレイヤについては，一時的に構成されるネットワークとしてアドホックネットワークの研究が行われている．また，アドホックネットワークを論理的な個人のネットワークとして取り扱うためのミドルウェアの研究も行われている．作業環境をどこでも利用可能にすることを目的としたSoftware Proxyや自然言語での質問応答システムなどのユーザインタフェースの研究も進められている．

RFIDは，すべてのものにIDをつけることによって論理的な世界から参照可能とするものである．これによって，コンピュータは物理的なものを参照できることになる．リーダから電磁誘導により電源を供給しIDを返すパッシブ型と，電池を内蔵し自身でIDを送り出すアクティブ型がある．到達距離は，パッシブ型で数cmから30cm程度であり，アクティブ型は数mから10m程度である．IDだけを返すものもあれば，プロセッサを内蔵し内部で演算可能なものもある．物理インタフェースは，ISO 18000により標準仕様が規定されている．RFIDの応用は，図書館や倉庫などの在庫管理や荷物や食品などに取り付け，どのような過程を経たかを記録しておき，後で参照すること（トレーサビリティ）などがある．これらの利用の仕方については，Auto-IDセンターやユビキタスIDを中心に標準化の動きがある．トレーサビリティについては，使い方によってはプライバシの侵害の危険性があり，現在議論が行われている．

センサネットは，物理的な環境の測定を行う．例えば，温度や湿度，ユーザやRFIDの位置の特定なども行う．IDだけでなく周囲の情報を取り込む際に用いられる．大体はマイクロプロセッサと測定用センサを持ち，目的に応じてプログラミングを行う．装置によっては後述するアドホックネットワークに対応したものもある．これらの装置は，1台当たり数万から数十万円と高価であり，大きさもタバコケースサイズである．応用は，ライン管理やビニールハウスの管理などであるが，さらなる小型軽量化と低価格化が望まれる．

アドホックネットワークは，どこでもネットワークを構成することを目的とする．主に無線通信装置を持つノードが複数点在し，メッセージがそれらのノードにより中継され目的のノードに到達する．ユビキタス環境下にあるデバイスなどを束ねるネットワークを形成する．IETFのMANET（Mobile Ad-hoc NETwork）[5]では，アドホックネットワークのプロトコルの標準化を行っている．アドホックネットワークプロトコルは，大別するとプロアクティブ型とパッシブ型に分類される．プロアクティブ型は，事前にルーティングテーブルを相互に交換する方式である．ルーティングテーブルとは，目的のノードにメッセージを送るには隣接するどのノードに送ればよいかを管理するために各ノードが持つテーブルである．パッシブ型はメッセージを送ろうとするときに，目的のノードまでのブロードキャストなどを用いてパスを動的に作成し，メッセージを転送する方式である．それぞれ長所と短所がある．プロアクティブ型は，あらかじめパスを作成するためメッセージをすぐに転送できるが，パスを構成するまでに時間がかかる．パッシブ型は，メッセージをすぐに送り出すことができないが，パスを動的に作成するためノードの移動などに対応しやすい．車や人がノードを持ち歩くような環境では，絶えずネットワークトポロジが変化するためプロアクティブ型では対応が難しい．最近では，AODV（Ad-hoc On-Demand Distance Vector Routing）[6]などのパッシブ型プロトコルやパッシブ型とプロアクティブ型のハイブリッドであるZRP（Zone Routing Protocol）[7]が，有力視されている．

アドホックネットワークは，フラットなネットワークである．このネットワークには，様々な装置やユーザ端末が接続し，様々なユーザが利用する．これらのユーザが互いに干渉することなくネットワークを形成する必要がある．KODAMA[8]は，物理ネットワークと論理ネットワークを分離し，論理ネットワークを階層化コミュニティとして管理する．ユーザの持つプライベートネットワークはコミュニティとして表現される．一つのコミュニティにはポータルとなるポータルノードが存在し，上位のコミュニティには一つのノードとして見える．ポータルノードはそれぞれ固有のルーティングポリシを持つ．コミュニティを用いてユーザのプライベートなネットワークを提供する枠組みである．

Software Proxy[9]は，デバイスやユーザに安全でプライベートかつ効果的なアクセスを提供する．位置検出装置と連動し，ユーザは自身がいる場所を明らかにせずとも必要とするサービスを受けられる．例えば，ユーザが音楽を聴きたいとコンピュータへ指示するだけで，どの場所のスピーカを鳴らすかはユーザの位置によって自動的に決定される．ユーザが移動したとし

ても，そのユーザの移動に伴い音楽の出力先は変更される．また，移動先のスピーカに出力してもよいかどうかは，それぞれスピーカに設定されたアクセスポリシによって決定され，不要な音楽の再生を避けることができる．

START (SynTactic Analysis using Reversible Transformations)[10]は，自然言語処理システムであり，理解モジュールと生成モジュールから構成される．理解モジュールは，英文を解析し，その文に含まれる情報について知識ベースを用いて処理する．ユーザは，知識ベースの情報を質問文から得ることが可能となる．このシステムによりユーザは，自然言語による応答が可能となる．

このように多岐にわたる分野の研究がなされている．真のユビキタス環境を実現するには，これらの成果が不可欠である．

本稿でこれらを網羅することは到底できないが，ユビキタス環境を実現する上で重要な要素であるサービス合成とセキュリティについて述べる．サービス合成は，環境にあるサービスを動的に連携させ，ユーザの行おうとする仕事をサポートする．また，その際にサービスのアクセス制御が必要である．

3. サービス合成とアクセス制御

3.1 概要

サービス合成とは，ユーザの要求に応じて動的にサービスを結合することである．ユビキタス環境下では，ユーザが移動し，その場のサービスやデバイスを利用することが想定される．単に一つのデバイス利用するだけならばJINI[11]のようにその場でドライバを探しインストールすることで利用可能となる．しかしながら，複数のサービスやデバイスを連携させるためには，単にドライバをインストールするだけでなく，サービスやデバイス間を連携させるための枠組みが必要である．例として，ユーザAは，自身の持つPDA内に保存しておいたファイルを使ってプレゼンテーションを行う場合を考える．そのユーザのPDA上で動作する“ローカルファイル”サービスとその部屋にある“プロジェクタに投影”サービスを合成し，PDA上のファイルをプロジェクタに投影し，ページの切り替えなどの制御をPDAで行うものとする．この例は，単にプロジェクタのドライバをインストールするだけでは実現できない．“ローカルファイル”サービスと“プロジェクタに投影”するサービスの結合を行わなければならない．一般に個々のサービスは，受付可能な入力や出力の型がそれぞれ異なり，結合できるサービスとできないサービスが存在する．最初にあげたようなサービス合成を実現するには意味レベルでの結合を動的に行い，実行しなければならない．サービスの組み合わせをあらかじめ用意しておく方法は，この場合利用できない．なぜなら，ユーザが持つサービスとの連携を行うためには，ユーザが持つサービスをあらかじめ知ることが必要となるが，ユーザが持つサービスをあらかじめ知ることは一般に困難である．また，一つサービスが増えるたびに，そのサービスと連携できるサービスの組み合わせを作成しなければならず，組み合わせが膨大となり現実的でない．このように，ユビキタス環境下では，サービスの意味レベルに踏み込み動的にサービスを合成する技術が必要となる．

また，ユビキタス環境下のセキュリティは，分散したサービス間で行わなければならない．サービスや装置が互いにPeer-to-Peer (P2P) 通信を行い，場合によっては複数のサービスや装置が連携し，サービスを提供することになる．このとき，悪意のあるユーザが侵入し，サービスや装置を不正に操作することを防ぐ必要がある．あるいは悪意のないユーザであってもその環境に存在するサービスすべてが利用可能というわけではなく，サービスに対するアクセス制御が必要となる．さらに，悪意のあるサービスによるユーザの持つ情報の取得などの攻撃についても防がなければならない．そのアクセス制御は，サービスによってポリシが異なり，さらに一つのサービスであってもユーザによって利用可能範囲が異なる．個々のサービスがそれぞれユーザを認証することは，それぞれが個別にユーザ情報を管理することになるので現実的でない．また，認証サーバなどを置き，集中的な管理を行う場合においてもすべてのユーザを登録する必要がある．なぜなら，ユーザは移動先で環境にあるデバイスやサービスを利用するためである．一つ一つのサービスやデバイスに対し，すべてのユーザの権限を設定することは非常に困難である．ユビキタス環境でサービスを保護し，ユーザを保護するためには，サービスおよびユーザを相互に認証する適切なアクセス管理機構が必要となる．

富士通研究所および米国富士通研究所では，サービス合成およびユビキタス環境下でのサービスに対するセキュリティについてTask Computingおよび Virtual Private Communityとして研究を行っている．

これらについて説明する．

3.2 Task Computing

Task Computing[12, 13]は，ユビキタス環境下でユーザの実施したいタスクのスムーズな実行をサポートするタスク指向のユーザ環境であり，ユーザが要求するタスクとその環境で実際に実行可能なサービスたちとの間のギャップを埋めることを目指す．Task Computingは，ユーザの状況に応じて利用可能なサービスを提示し，サービスの可能な組み合わせとしてのタスクをユーザに提示する．ユーザはタスクの実行を指示するだけで，システムがそのタスクを実行する．タスクの実行は複数のサービスに展開されて実行されるが，ユーザはその詳細を知る必要はない．例えばサービスがどのマシンで走っているのか，Webサービスなのか UPnP[1] サービスなのか，サービス間のデータの受け渡しなど，サービスの実行に関わる詳細はシステムが処理し，ユーザはやりたいこととしてのタスクに集中できる．我々はTask Computingをサポートする実行環境をTask Computing Environment (TCE) と呼び，TCEは次の項目をサポートすることにより，上記のような環境を実現している．

- Task Computingのワークフロー
- タスクとサービスの意味記述
- エンドユーザによるタスクの実行，再利用
- エンドユーザによる生成や破棄などのサービス操作

TCEの具体的なコンポーネントとしては次のようなものがある．

- Task Computing Client (TCC)
- セマンティックな記述を持つサービス
- サービス発見メカニズム
- サービスの動的な生成などを可能にするサービス制御メカニズム

これらのコンポーネントにより，サービスの発見，選択，サービス合成および実行，さらにはサービス生成，破棄までをエンドユーザがダイナミックに行うことを可能にする．ユーザが行おうとするタスクは合成されたサービスとして表現される．TCEでは，Webサービス (WSDL[14]，SOAP[15])，UPnP[16]，OWL[17]，OWL-S[18]などの標準をできる限り用いることにより，個々のコンポーネントが自己完結しながらも，互いに緩やかに連結することを実現した．これにより，アドホックなネットワーク環境においても即座にシステムを展開することが可能となる．

[1] Universal Plug and Play[16]．Intel社やMicrosoft社など650社以上が参画している標準規格で，ネットワーク上のデバイスの発見，自動設定，制御を可能とする．

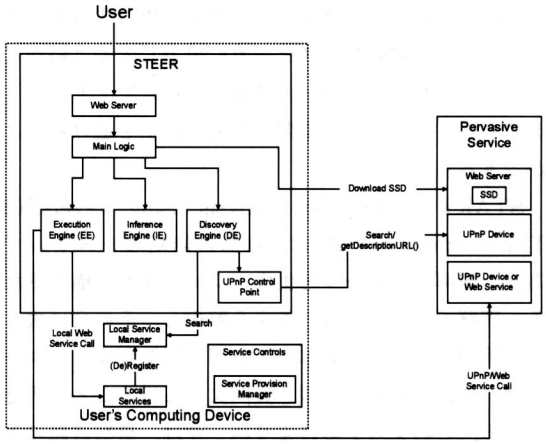

図1 TCEアーキテクチャ

ここではTCEの動作を，Task Computing Client (TCC) の一つであるSemantic Task Execution EditoR (STEER) の面から，例を示しながら解説する．

図1にTCEのアーキテクチャを示す．ユーザが利用するノートPCやPDAなどのような計算機上でSTEERやローカルサービスマネージャなどが動作する．ユーザインタフェースは，他のコンピュータ上で動作させることも可能である．サービスは，サービス発見メカニズムとしてのUPnPデバイス，セマンティックなサービス記述を提供するためのウェブサーバ，および実際のサービスの実現であるWebサービスあるいはUPnPアクションからなる．

タスクを実行するまでのSTEERの動作は二つのフェーズからなる．一つは発見フェーズであり，他方はサービス合成フェーズである．各フェーズについて説明する．

最初のフェーズは発見フェーズである．STEERは，利用可能なローカルおよび環境に存在するサービスを検索し，絶えず更新し続ける．STEERはローカルサービスマネージャに問い合わせ，例えばローカルディスクにあるファイルをWebサーバを通じて提供する“ローカルファイル”サービスなどの利用可能なローカルサービスを発見する．発見された各サービスについて，OWL-Sを用いて記述されたセマンティックなサービス記述 (SSD: Semantic Service Description) を取得し，推論エンジンへ転送する．さらに，STEERはUPnPのデバイス発見プロトコルを用い

て，そのネットワーク上のサービスを検索する．環境に存在するサービスは，このプロトコルに応答し，STEERはその存在を検出する．検出後，各サービスについて，UPnPのアクションを用いてそのサービスのSSDのURLを取得し，SSD自体を得られたURLからダウンロードする．こうして得られたSSDも推論エンジン（IE）へ転送される．

次にサービス合成フェーズに入る．STEERは推論エンジンに対し，入力を持たないサービスから始め，入出力のタイプがマッチする二つのサービス合成を依頼する．推論エンジンは可能なサービスの組み合わせを返し，STEERはそれらの組み合わせをこの環境で可能なタスクとして合成ページでユーザに表示する．推論エンジンが返すサービスの組み合わせは，サービスの発見された時間やマッチの正確さなどによって順番付けられ，またそれらの組み合わせは，ファイル，住所，スケジュールなどのサービス間の入出力のタイプごとに分類されており，それに従ってメニューが生成される．このようにその合成ページは，発見されたサービスのセマンティックなサービス記述だけから，完全に動的に生成される．

ここで使われる，入力を持たないサービスから始まる二つのサービス合成には，次のメリットがある．

1) 入力が必要ないため即座に実行可能である．
2) 目的語＋動詞の形式をとるので直感的にわかりやすい（英語の場合は“動詞＋目的語”の順になる）．
3) サービスが適度な粒度であり，わかりやすい．

合成されたサービスの実行は，実行ボタンを押すのみである．もし合成したいサービスを変えたい場合には，ドロップダウンメニューから互換性があるサービスを選択するのみである．例えばユーザが自分のコンピュータ上のファイルをプロジェクタに表示したければ，“プロジェクタに投影”サービスと“ローカルファイル”サービスの組み合わせを選択し，その組み合わせが実行されれば，ローカルファイルがプロジェクタへ投影される．

また必要であれば，二つのサービスを合成から始めて，そこからさらに互換性のあるサービスを組み合わせることも可能であり，これにより，さらに複雑なサービス合成およびその実行を行うことができる．

図2にSTEERによるサービスの実行例を示す．仮にいま“プロジェクタに投影”（View on Projector）サービスと“ローカルファイル”（Local File）

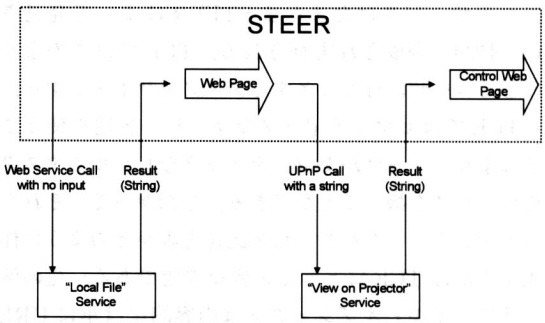

図2　STEERによるサービスの実行

サービスの合成を実行すると仮定する．STEERの実行エンジン（EE）は，サービス合成の実行とUPnPとWebサービスの呼び出しを行う．合成は意味レイヤで行われるためSTEERは，UPnPとWebサービスの入出力パラメータと意味オブジェクトの間のギャップを埋める必要がある．STEERの実行エンジン内でセマンティックなサービス記述が持つgrounding情報を元にUPnPとWebサービスのパラメータたちと意味オブジェクト間の変換を行う．このメカニズムによって，STEERは，意味レベルでUPnPとWebサービスを呼び出すことが可能となる．

最初のサービスである“ローカルファイル”サービスが呼び出されると，ローカルファイルの選択ダイアログが表示される．ユーザは該当するファイルをこのダイアログから選択する．選択されたファイルは，Webフォルダへコピーされ，そのサービスはそのファイルを示すURLを返す．そのURLはSTEERによって「ファイル」という意味のオブジェクト（意味オブジェクト）に変換される．

次のサービスである“プロジェクタに投影”サービスがその「ファイル」意味オブジェクトと共に呼び出される．STEERは“プロジェクタに投影”サービスのgrounding情報を元に「ファイル」意味オブジェクトから，そのサービスの実行のための文字列としてのURLを取り出し，そのパラメータを使って，（UPnPサービスである）“プロジェクタに投影”サービスを実行する．そのサービスは与えられたURLからファイルをダウンロードし，プロジェクタに表示する．このようにして，もともとユーザのコンピュータ上にあったファイルがその部屋のプロジェクタに表示されることになる．

ここで重要な点は，あらかじめ二つのサービスの間で何らかの取り決めがあったわけではないことである．

二つのサービスはたまたまSTEERによって発見され，動的に合成され実行される．TCEではこのようにしてユーザの行おうとするタスクをサポートする．

TCEではセマンティックなサービス記述を使うことにより，ユーザがセマンティックなレベルでタスクやサービスを扱うことができる．したがって，それらのセマンティックなサービス記述を誰がどのように作成するかはTCEにとって重要な課題である．先の例の「ファイル」オブジェクトは内容的には単にURLの文字列のみの単純なものであるが，STEERが取り扱うものには住所やスケジュールなどのもっと意味的に複雑なものもある．意味オブジェクトとサービスのパラメータの変換処理では，意味オブジェクトが複雑になれば，それに応じて変換情報作成は難しくなる．我々は，エンドユーザがそれらの情報を作成することは期待せず，SE（Systems Engineer）などの該当するデバイスやWebサービスの専門家によってセマンティックなサービス記述（SSD: Semantic Service Description）が作成されると考えている．我々はOntoLinkと呼ばれるGUIベースのツールをそれら専門家によるSSDの作成をサポートするために提供している．OntoLinkは，WebサービスのWSDLのパラメータと意味オブジェクトを結合するSSDを視覚的に作成する環境を提供する．OntoLinkはOWL-SによるSSDを生成し，その中にWSDLのパラメータと意味オブジェクト間の変換のためのXSLTスクリプトを埋め込む．STEERの実行エンジンでは，サービス実行時にそれらのXSLTスクリプトを用いることにより，WSDLのパラメータと意味オブジェクト間の変換を実現している．

OntoLinkは，同様に異なるオントロジの意味オブジェクトの間での関係付けもサポートする．パラメータの変換は，STEERが意味のレイヤでの操作を可能とするが，それですべての変換の問題が解けるわけではない．ユビキタス環境下においても使われる装置やサービスなどのオントロジ（意味オブジェクトの辞書に相当するもの）は比較的少ない数になるのではないかと考えているが，それでもそれらのオントロジ間の互換性の問題が残る．OntoLinkでは異なるオントロジの意味オブジェクトを変換するセマンティックなサービス記述を作成する．これらのセマンティックなサービス記述は，その中にXSLTスクリプトを持ち，そのSSDだけで実行可能になっている．異なるオントロジの意味オブジェクトを扱うサービスたちも，OntoLinkで作成された上記のサービスを通じて組み合わせて使えることが可能になる．

3.3 Virtual Private Community (VPC)

VPC[19, 20]は，ユビキタス環境下でデバイスやサービス間が連携する際のセキュリティフレームワークを提供する．VPCは，サービスおよびユーザを保護し，ユーザおよびサービスの認証を行い，適切なアクセス管理機構を提供する．VPCでは，サービスやユーザはエージェントとして取り扱われる．サービスの実行時にコミュニティを作成し，そのコミュニティへの参加およびコミュニティ内での役割の割付を安全に行う．

図3にVPCの概念図を示す．エージェントはそれぞれ属性を持つ．コミュニティは，ポリシ（Policy Package）によって定義される．Policy Packageでは，属性から役割の割付ルールが記述されている．例えば，太陽マークの属性を持つエージェントは，Role 1を持つことが可能である．コミュニティへ参加するエージェントはそのコミュニティを定義しているPolicy Packageを取得し，Policy Packageを評価する．この演算は，エージェント自身によってJavaCard[21]のような安全な領域である耐タンパ性デバイス内で実行される．耐タンパ性デバイスは，ユーザですら自由にデバイス内部の情報を変更することはできないため，悪意のあるユーザからサービスを守ることができる．また，Policy Packageは電子署名がなされており，正規の製作者以外のものや捏造されているものは検出でき，悪意のあるサービスからユーザを守ることが可能である．

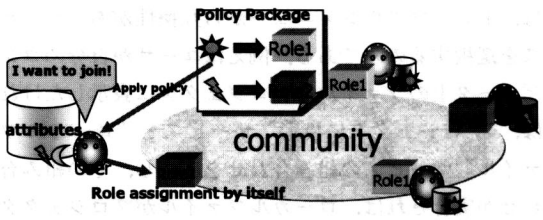

図3　VPCの概念図

<policy package>	::=	<rule> <role> <content>
<rule>	::=	<condition> <role names>
<condition>	::=	*Combination of PKI certificates*
<role>	::=	<role name> <program name> <init description>
<content>	::=	<content name> <content path>

図4　Policy Packageの構造

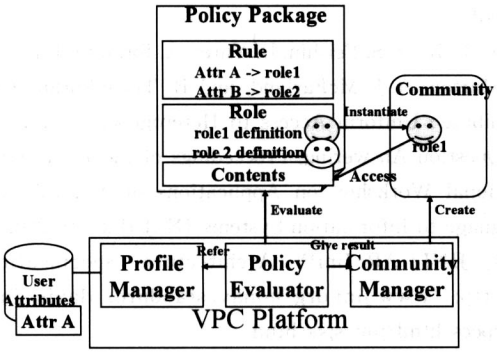

図5　VPCプラットフォーム

図4にPolicy Packageの構造を示す．Policy Packageは，役割の割付ルールと役割の定義およびそのサービスで使用するコンテンツから構成される．割付ルールは属性と役割名のセットであり，VPCの実行エンジンは定義された属性を持つとき該当する役割を割り付ける．条件となる属性は，ANDやNOTを用いて組み合わせることが可能である．属性は，PKIの証明書である．役割は，役割名と該当するプログラムコードおよび初期化パラメータセットである．コンテンツは，サービス中のコンテンツ名と実際のパスから構成される．コンテンツ名でアクセスを行うと実際のパスから取得され，パスは，URLもしくはファイルパスである．ファイルパスを指定されたときは，Policy Packageと共にファイルも添付される．

図5にVPCの実行環境であるVPCプラットフォームの概略を示す．VPCプラットフォームは，主にプロファイルマネージャ，ポリシ評価部，コミュニティマネージャから構成される．プロファイルマネージャは，エージェントの属性を管理し，ポリシ評価部は，与えられたPolicy Packageをプロファイルマネージャに属性値を問い合わせながら評価し，このエージェントが持つべき役割を決定し，役割プログラムをコミュニティマネージャへ登録する．コミュニティマネージャは，実体化された役割プログラムを管理し，他のVPCプラットフォームの同一コミュニティを接続し，論理的に一つのコミュニティとする．

Policy Packageは再配布可能である．Policy Packageを取得したユーザは，いつでもこのPolicy Packageを利用可能である．ユーザが適切であれば上に述べた機構により，適切な役割プログラムが起動し，そのユーザにあったサービスを受けることができる．

3.4　応用例

Task ComputingおよびVPCを用いた例を示す．ある研究所のある部屋にプロジェクタとプリンタが設置されている．プロジェクタは，誰でも利用可能であるが，プリンタは，その研究所員もしくは利用許可を与えられた訪問者のみが利用できる．プロジェクタのPolicy Packageは，TRUE→SHOWであり，プリンタのPolicy Packageは，RESEARCHER | VIP→PRINTとする．SHOWおよびPRINTは，VPCの役割プログラムであり，Task ComputingのSTEERのサービス呼び出しプログラムとする．STEERは，VPCによって導出された割付プログラムをサービス実行時に起動する．また，TRUEは常に真となる属性である．これらのPolicy Packageは，その研究所の電子署名がなされている．

例えば，教授A，学生Bがこの研究所を訪れ，それぞれ訪問カードを受け取るとする．カードは，VPCプラットフォームが動作するJavaCardであり，AのカードにはVIP属性が，BのカードにはVISITOR属性が書き込まれている．訪問者A，Bと研究者Cは，プロジェクタとプリンタのある会議室で打ち合わせを行う．Bは自身が持つノートPCのローカルファイルをTask Computingのサービス合成を用いてプロジェクタに投影し，プレゼンテーションを行う．このとき，プロジェクタのPolicy Packageは属性TRUEであるので，Bはプロジェクタを利用できる．Bがその資料を印刷しようとすると，プリンタのPolicy PackageがBのノートPCに取り込まれ，評価される．Policy PackageはRESEARCHER | VIP→PRINTであるが，Bの属性はVISITORであるので，役割プログラムが割り付けられないため，STEERはPRINTサービスを起動できず，Bは印刷をすることができない．このファイルを教授Aのノート PCに転送し，Aが印刷する．Aの属性はVIPであるので，そのファイルを印刷することができる．

このように各サービスが持つポリシによって，ユーザのサービスに対するアクセスの制御を行うことができる．

4.　まとめ

本稿では，ユビキタス環境を実現するための技術について述べた．我々の研究グループは中でもサービスの動的合成およびそのアクセス制御にフォーカスしている．これまで述べたように様々な分野で多種多様な

研究がなされており，必ずや近い将来ユビキタス環境が現れると思われる．携帯電話はすでにいつでもどこでも利用できる時代となっている．便利さは増したが，その反面携帯電話を用いた様々な犯罪も増加している．ユビキタス環境は突然現れるのではなく，徐々に浸透していくと予想される．このとき我々ユーザは，便利になる反面，プライバシの侵害などの危険にさらされる可能性も否定できない．技術的な整備もさることながら，ユビキタス社会がもたらす効果や問題点を利用する側の立場で議論する必要もあると思われる．

参考文献

[1] M. Weiser: Some Computer Science Issues in Ubiquitous Computing, Commun, ACM, vol. 36, no. 7, pp. 74-84, 1993.

[2] 佐藤一郎：スマートタグRFIDタグ：技術動向と影響，情報処理 2004/1, pp. 58-62, 2004.

[3] Auto-ID Center, http://www.autoidcenter.org/

[4] Stardust, http://www.xbow.com

[5] MANET, http://www.ietf.org/html.charters/manet-charter.html

[6] C. Perkins: Ad hoc On-Demand Distance Vector (AODV) Routing, http://www.ietf.org/rfc/rfc3561.txt

[7] Z. Haas and M. Pearlman: The zone routing protocol (ZRP) for ad hoc networks, draft-ietf-manet-zone-zrp-00. txt work in progress, http://wnl.ece.cornell.edu/Publications/draft-ietf-manet-zone-zrp-02.txt

[8] G. Zhong, K. Takahashi, S. Amamiya, T. Mine, and M. Amamiya: KODAMA project, AAMAS 2002, pp. 43-44, 2002.

[9] M. Burnside, D. Clarke, S. Raman, S. Devadas, and R. Rivest: Access-Controlled Resource Discovery, http://www.lcs.mit.edu/research/abstracts/pdf/59.pdf

[10] B. Katz, S. Felshin, D. Yuret, A. Ibrahim, J. Lin, G. Marton, A. J. McFarland, and B. Temelkuran: Omnibase: Uniform Access to Heterogeneous Data for Question Answering, Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002), 2002.

[11] JINI: AR-Jini™ Architecture Specification, http://www.jini.org/nonav/standards/davis/doc/specs/html/jini-spec.html

[12] R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin: Ontology-Enabled Pervasive Computing Applications, IEEE Intelligent Systems, vol. 18, no. 5, pp. 68-72, 2003.

[13] R. Masuoka, B. Parsia, and Y. Labrou: Task Computing—The Semantic Web meets Pervasive Computing—, LNCS 2870, pp. 866-881, 2003.

[14] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl

[15] SOAP, http://www.w3.org/TR/SOAP/

[16] UPnP Forum, http://www.upnp.org

[17] Web-Ontology (WebOnt) Working Group, http://www.w3.org/2001/sw/WebOnt/

[18] DAML Services, http://www.daml.org/services

[19] T. Iwao, S. Amamiya, G. Zhong, and M. Amamiya: Ubiquitous Computing with Service Adaptation Using Peer-to-Peer Communication Framework, FTDCS 2003, pp. 240-248, 2003.

[20] T. Iwao, S. Amamiya, K. Takahashi, G. Zhong, T. Kainuma, L. Ji, and M. Amamiya: Information Notification Model with VPC on KODAMA in an Ubiquitous Computing Environment and its Experiment, LNAI 2782, pp. 30-45, 2003.

[21] JavaCard: Java Card Platform Security, http://java.sun.com/products/javacard/JavaCardSecurityWhitePaper.pdf