情報処理学会論文誌 Vol. 49 No. 6 1920–1931 (June 2008)

ヰキ$_\phi$によるユビキタス複合サービス開発支援

望月祐洋[†1]

本論文では，ユビキタスコンピューティング環境に存在するさまざまな要素サービスを組み合わせて複合サービスを実現する際に，環境や既存サービスについての予備知識の限られたサービス開発者が単独または共同で開発を進める際に役立つ支援方式を提案する．本方式の特徴は，開発支援ツールのベースとしてWikiを採用することで，要素サービスの設定情報の確認・変更作業や複合サービスの定義作業をWikiページの閲覧や編集作業に整合的に行えるようにした点にある．また，本方式に基づく開発支援ツールは，ユビキタスコンピューティング環境の基盤システムで多く利用されるタプルスペースモデルに基づく間接通信の枠組みとも親和性が高い．本論文では，本方式によるプロトタイプとしてヰキ$_\phi$の設計および実装，さらに複合サービスの構成方法について示す．

Wiki$_\phi$ for Supporting the Development of Ubiquitous Composite Services

Masahiro Mochizuki[†1]

In this paper, we propose a method for supporting independent or a group of service programmers with limited a priori knowledge of environments and existing services to develop composite services by combining a variety of elemental services in a ubiquitous computing environment. A notable feature of the method is to adopt Wiki as a basis for a development tool tailored to the rapid prototyping of ubiquitous composite services. It enables to browse and modify the configuration information of elemental services, as well as to define composite services in a manner consistent with the actions on wiki pages such as browsing and editing them. Furthermore, a development tool based on the method has an affinity with an indirect communication mechanism based on the tuple space model, which is commonly used for ubiquitous computing environments. This paper presents the design and implementation of a prototype system of Wiki$_\phi$, and how to develop composite services using the system.

1. はじめに

現在までに多くのユビキタスコンピューティング[1]の実験環境が構築され，研究が重ねられている[2)–4)]．さらに，それらを発展させるかたちで地理的に分散した同種または異種の実験環境どうしを結合し，相互運用性を向上させる試みも行われてきた[5),6)]．こうした環境で利用者を支援するアプリケーションの多くは，いわゆるサービス指向アーキテクチャ（SOA）に基づいており，同種または異種の単機能な機器どうしが協調することで実現される．また，環境に存在するヒトやモノの構成が経時的に変化するため，サービスの動的構成と持続的提供を目的として，サービス構成要素の発見，構成要素間の通信，サービスの複合，アクセス制御等が基本機能として基盤システム側で提供される．

サービス構成要素をどのような計算モデルで表現するかは，エンドユーザや開発者にとっての理解しやすさ，記述言語の種類，複合サービスの定義方式，採用する通信モデル等と密接に関わっており，主要な研究課題の1つとして従来の提案システムでもさまざまなアプローチがとられている．

たとえば，文献7), 8)ではサービス構成要素を分散コンポーネントとして表現し，サービスを動的に集約して構成されるGUIプログラムと各分散コンポーネント間の通信にRPC/RMIに基づく同期式通信モデルが用いられる．文献9)–12)はLinda[13]のタプルスペースに基づく通信モデル，文献14)，15)ではイベント通信モデル，文献16)では移動コンポーネント，文献17)では階層化された移動コンポーネント（移動エージェント）として表現し，コンポーネントの移動による通信モデルを採用している．

さまざまなモデルに基づくサービスが氾濫するなかで，複合サービスを構成するために個別のサービスごとに機能やインタフェースの情報を収集しつつ開発を進めるのは大変煩雑である．こうした情報の共通化や，各種パラメータの一元管理，開発ノウハウを共有するしくみが望まれる．

開発主体の異なるさまざまなサービスの公開・発見・結合を実現するために，要素サービスのメタデータの記述にWSDL[18]，サービス間の呼び出しにSOAP[19]，カタログ化と検索にUDDI[20]，要素サービス間の連携と複合の記述にBPEL[21]，OWL-S[22]というようにWebサービスの関連技術を採用するアプローチが存在する．たとえば，文献23)で

†1 東京工業大学学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

は，SOAP/WSDL，OWL-S を用いたユビキタス環境における動的なサービス合成手法を提案している．これらの技術の多くは主にエンタプライズ用途に利用されており，一般開発者向けにはより仕様の緩やかで開発の敷居が低い REST，XML-RPC，ATOM Publishing Protocol[24] といった Web API の普及が進んでいる．

このような状況をふまえ，本研究では，一般開発者が複合サービスを開発する際に，要素サービスの設定情報に一元的にアクセスして情報共有できるようにし，開発現場で協働作業を進めつつ複合サービスの構成に必要なグルーコードを記述できるようなラピッドプロタイピングスタイルを支援するしくみを実現することにした．少人数の開発者グループによるユビキタスサービスの設定情報の共有および共同編集を目的として，タプルスペースモデルに基づく Wiki[25] エンジンをベースとするヰキ$_\phi$（「うぃきふぁい」と発音）と呼ぶラピッドプロトタイピング支援ツールを構築した．

ヰキ$_\phi$ では，ユビキタスサービス初期化時に外部制御コマンド，イベントハンドラ，ログ情報，デバッグ出力に関する情報の集合を Wiki ページとして登録することで，従来の Wiki の編集作業と同様の感覚での各種設定情報の共有，共同編集・管理を可能にした．このしくみによって，複数のユビキタスサービスが協調することで構成される新たな複合サービスのラピッドプロトタイピングが容易になり，開発の敷居を下げる効果を期待できる．また，インターネット環境で Web API を利用した Mash Up によってさまざまな派生サービスが登場したように，開発支援ツールを充実させることは，将来的に新たなサービス開発者や新奇性の高いサービスの孵化につながるものと期待される．

本論文の構成は以下のとおりである．まず，次章では，ユビキタスコンピューティング環境を前提とした複合サービスの構成手法について現状をふまえて各種の制約を明らかにしつつ検討する．3 章ではユビキタス複合サービス開発支援ツールであるヰキ$_\phi$ の設計について述べ，4 章ではその実装について説明する．さらに，5 章では複合サービスの構成例について手順を追って概説する．最後に 6 章でまとめと今後の課題について述べる．

2. 複合サービスの構成手法

本論文では，ユビキタスコンピューティング環境に存在する情報機器に対応づけられたソフトウェアサービスを「要素サービス」，複数の要素サービスを組み合わせて実現される新たなサービスを「複合サービス」として言及する．本章では，複合サービスの構成手法にかかる現状，制約，および構成方式の分析を行う．

2.1 現　　状

ユビキタスコンピューティング環境の研究は，テストベッドを構築し，その中に設置された各種センサや制御機器を活用した多様なアプリケーションを構築することで，ユビキタスコンピューティングの未来象を具体的なかたちとして提示するとともに，さまざまな研究課題を明らかにする役割を担ってきた．

当初は，基盤システムの開発者がアプリケーション開発者を兼ねることが多く，また，アプリケーションも特定の動作環境を対象に作り込まれていた．

しかし，さまざまなミドルウェアやアプリケーションフレームワークの研究開発が進められるのにともない，予備知識を必要としない開発手法の重要性が指摘される[16] 等，専門の研究者がデモンストレーション用アプリケーションを作り込む段階から，一般開発者が試行錯誤を通じて新たな複合サービスをプロトタイピングするための開発プラットフォームの構築や，あるいは人手を介さずに複合サービスを自動/半自動的に生成するといったアプローチが試みられている．

2.2 制　　約

本節では，ユビキタスサービス（要素サービスおよび複合サービス）の一般開発者が参入時に行き当たると考えられる各種の制約について概観する．

(1) 基盤システムへのアクセス権限の制約
要素サービスの開発過程で，基盤システム側に予期せぬ不具合が生じた場合に，管理者権限がなければシステムをリセットしたり不具合を直接修正したりするのは困難である．また，基盤システムのシステムインタフェースに変更を加えたいといった要求が生じた場合も同様である．管理者への連絡手段が確保されている必要がある．

(2) サービスへのアクセス権限の制約
実際の運用環境では，複合サービスを構成する個々の要素サービスの開発者が異なる場合が想定される．また，複合サービスの開発者が要素サービスの開発者と同一とも限らない．適切なアクセス権限が付与されなければ，サービスの設定を変更するのは困難である．

(3) プログラミングモデルおよび通信モデルの制約
開発プラットフォームによって，プログラミングモデルおよび通信モデルが異なるため，たとえば，ある環境では「通信はすべてイベント」，別の環境では「基本構成要素は移動コンポーネント」といったように，アプリケーション開発者は環境ごとにシステム設計者の世界観に従わざるをえない．

(4)　時間的・空間的制約

特定の実行環境を対象とするサービスを実地で開発する場合には，その場所にどのぐらいの時間滞在して設備をどの程度占有できるかといった物理的な制約が生じる．何らかの理由で機器の利用が難しい場合には，その機器がひとまずない状態で開発作業を進める WOz の手法[26),27)] が，また，実地での開発が難しい場合に，要素サービスのシミュレータ等の開発支援ツールの利用が不可欠である．

(5)　導入コストの制約

種々の導入コストの高さが一般開発者の参入障壁になっている．まず，実際の開発に着手するまでに，複数のサーバを起動したり，設定ファイルを編集したり開発環境の設定の煩雑さや管理運用に要するコストが制約となる．次に，上述のプログラミングモデルおよび通信モデルの制約にも関連するが，ユビキタスコンピューティング環境ごとに独自の開発ツールが提供されている場合，開発ツールに慣れるまでの学習コストは膨大である．最後に，実験環境の構築にかかる予算コストが高いため，開発者層が限られてしまう．

2.3　複合サービスの構成方式

本節では，従来のユビキタスコンピューティング環境の研究で報告された開発支援ツールに見られる複合サービスの構成方式について検討する．**図 1** は，複合サービス構成方式の分類を示しており，以下で各方式について述べる．

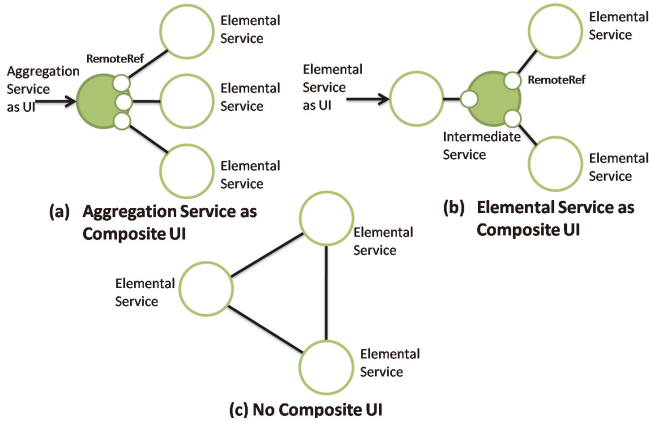

図 1　複合サービス構成方式の分類

Fig. 1　Taxonomy of the structure of a composite service.

まず図 1 (a) の方式では，複数の要素サービス（Elemental Service）の遠隔参照を集約サービス（Aggregation Service）に集約して複合サービスを構成する．また，集約サービスがユーザインタフェースの機能を提供する．開発者からは，集約サービス定義が複合サービスの計算表現として認識される．利用者からは，集約サービスのユーザインタフェースが複合サービスのエンドポイントとして認識される．文献 7)，8) に見られる方式である．

次に図 1 (b) では，複数の要素サービスの遠隔参照を中間サービス（Intermediate Service）に集約して複合サービスを構成する．複合サービスのユーザインタフェース機能は，要素サービスの 1 つが提供する．開発者は中間サービスを通じて要素サービスどうしを関連づける．文献 28) に見られる方式である．

最後に図 1 (c) では，複数の要素サービスから複合サービスが構成されるが，ユーザインタフェース機能は提供されない．複合サービスとして独立の計算表現は持たず，個々の要素サービスの設定を行うことで，要素サービス間の協調が実現される．

3.　ヰキ$_\phi$ の設計

本章では，前章をふまえてユビキタス複合サービス開発支援ツールであるヰキ$_\phi$ の設計について述べる．

3.1　要　　求

前節の制約をふまえ，複合サービスの開発支援への要求を以下の項目ごとに説明する．

- 開発情報の蓄積・共有を支援すること．制約 (1)，(2)，および (4) をふまえ，ユビキタスコンピューティング環境内で利用可能な要素サービスの設定情報を適切な権限管理のもとで閲覧し，複合サービスの構成に必要な情報を収集しやすいこと．また，必要に応じて設定情報を即時変更できること．さらに，要素サービス，複合サービスの開発に関わる情報のカタログ化やノウハウの蓄積を体系的に行えること．さらに，個人だけでなく開発者グループでの情報の蓄積・共有を支援できること．
- 要素サービスの開発コストを低減させること．複合サービスを構成するための前提となる要素サービスを充実させるためには，制約 (3) および (5) をふまえ，多様な開発プラットフォームで構築された既存サービスを取り込める柔軟な枠組みである必要がある．また，新規サービスを開発する場合にも，要素サービス開発者の使い慣れたツール，プログラミング言語，プラットフォームの選択の自由を確保できる必要がある．
- 言語中立な複合サービスを定義できること．要素サービスの場合と同様に，制約 (3)

をふまえ，特定のプログラミング言語やモデルに極力依存しない複合サービスを定義できることが望ましい．また，その際に前章の複合サービスの構成方式図 1 (a) および図 1 (b) で見たように，複合サービスを独立の計算表現として記述できることが望ましい．その利点として，複合サービス単位での設定情報の管理や，サービスの環境やコンテキストへの適応を実現する場合に，複合サービスと要素サービスとの再対応づけを柔軟に行えることがあげられる．

- 開発ツールの導入・利用コストが低いこと．制約 (5) をふまえ，一般開発者によるサービス開発への敷居を下げるために，複合サービスの開発に汎用性の高い手段を提供することが望ましい．
- 機能拡張が容易であること．さまざまな先行研究で指摘されているように，ユビキタスコンピューティング環境では経時的にヒトやモノの構成が変化するために漸次拡張性が必要不可欠である．

3.2 設計方針

前節の要求をふまえ，以下の設計方針をとる．

- Wiki をベースとする設定情報の閲覧・共同編集支援方式を採用する．ローカルエリアネットワーク上に存在する要素サービスのメタデータの閲覧，複合サービスのラピッドプロトタイピング，開発環境や開発方法に関する文書情報の閲覧・編集，さらにそれらの機能を複数人で共用できるようにする．この方針を体系的に実現するために Wiki を活用する手法を採用する．編集操作が簡便で，一般に普及しているツールをベースにすることで，開発の敷居を下げる効果が期待される．また，Wiki 自体が Web 技術に基づいているため，既存のさまざまな Web API を利用したサービスとの接続性も高い．なお，Wiki の特徴については次節で詳述する．
- タプルスペースによる間接通信モデルを採用する．タプルスペースを Wiki のページ情報を蓄積するストレージ兼通信バスとして利用する．文献 29) で指摘されるとおり，タプルスペースに基づく通信では，メッセージの送受信者を時間，空間，（送信者側での）同期の面で分離できるため，モバイル・ユビキタスコンピューティング環境との親和性が高く，文献 9)–12)，30) で通信基盤として採用されている実績がある．このため，Wiki をベースとする情報閲覧・共同編集機能とユビキタスコンピューティング環境の基盤サービスとのシームレスな垂直統合を期待できる．また，要素サービスの開発に多くの言語をサポートする目的にも，タプルスペースとの通信に Linda の in，out，rd という基本操作の API を提供すればよいため，開発コストを抑制するのに都合がよい．さらに，機能拡張性の面でも，基盤サービスとは独立に Linda クライアントとして拡張機能部分を開発し，システム実行時に動的に追加しやすい．
- Wiki 操作に整合した複合サービス構成方式を実現する．本研究の大きな特徴であるが，Wiki 操作を基本にして，複合サービスの構成をどこまで支援できるかをプロトタイプシステムの構築を通じて検討する．

3.3 Wiki の特徴と役割

Wiki とは，複数の利用者が Web ブラウザ上で Wiki 記法と呼ばれる単純な記法を用いて Web ページを共同編集することで，Web コンテンツの公開と共有を支援するソフトウェアである[⋆1]．

Wiki 記法は Wiki エンジンの実装の増加とともに，さまざまな派生的な記法が登場してきており，互換性に難点があるが，多くの Wiki エンジンの実装では，Wiki コマンドを定義することで機能拡張する手段を提供しており，Wiki 記法から HTML への変換ルールをはじめ，ユーザが手軽に独自のカスタマイズを行える．

Wiki にはコンテンツの共同編集・管理ツールとしての側面と，個人データの編集・管理ツールとしての側面がある．我々は，これまでに後者の側面に着目し，タプルスペースを介した複数の計算機間のインタラクションを通じてビデオクリップを生成し，コンテンツのメタデータ，コンテンツの再生制御 UI，コメント情報等を Wiki ページとして一元的に管理するシステムの構築を行った[32)]．

この特定用途に限定されたシステムの構築を通じて，規定形式のタプルを Wiki ページと対応づけることで，Wiki ページの閲覧・編集・保存操作とタプルスペースによるアプリケーション間の非同期イベント通信，データの蓄積・検索といった機能を効果的に結びつけられること，また，既動作部分とは独立して機能を拡張しやすいことを確認している．

このような経験をふまえ，Wiki の有するコンテンツの共同編集・管理ツールとしての側面に着目し，複合サービスの要素サービスとなるソフトウェアの設定情報を Wiki ページとして一元的に管理し，設定情報の集約，確認，変更，さらに複合サービスの定義までを，一般利用者になじみ深く使い勝手の良い Wiki の操作を通じて行うための汎用的な枠組みを提案する．

近年，Wiki を文書管理だけでなく，ソースコード，関連ドキュメント，コンパイル，実行，デバッグまでサポートするソフトウェア開発のためのプラットフォームとして発展させ

⋆1 なお，Wiki の成立・発展の過程については文献 31) に詳しい．

る試みも提案されており[33]，また，オープンソースの Wiki エンジンの実装でも，さまざまな拡張モジュールが開発されている．本研究はユビキタスコンピューティング環境を対象とする複合サービスのラピッドプロトタイピング用ツールとして Wiki の発展の可能性を模索する新たな試みの 1 つと位置づけられる．

なお，本研究では複合サービスを構成する要素サービスに対応づけられた機器間のデータ通信，制御通信は，外部ソフトウェアによる制御機能を機器側で提供する，またはシステム基盤と機器の間にプロクシモジュールを配置することでソフトウェア層で一元的に扱うアプローチをとるため，機器間通信の相互運用性については直接扱わない．機器どうしの相互運用性については HAVi，DLNA といった機器メーカ主導の標準仕様策定の取り組みや，研究コミュニティでは文献 34) といった提案がなされている．

4. 実　　装

本章では，前章の内容をふまえたヰキ$_\phi$ のプロトタイプ実装について述べる．

4.1 実装概要

まず，**図 2** に実装の概要を示す．

基本的には，有線/無線ネットワーク接続されたサーバ機器ごとにディスカバリサービス（Discovery Service），タプルスペースサービス（Tuple Space）が動作し，その上でユビキタスサービス（Ubiquitous Service）が動作する．このユビキタスサービスは要素サービス（Elemental Service）に相当する．要素サービスは，何らかの制御機器に対応づけられたプロクシ，GUI アプリケーション，レガシーアプリケーション等さまざまな用途のものが想定される．要素サービスはタプルスペースサービスとクライアントライブラリ（Client Library）経由で通信する．

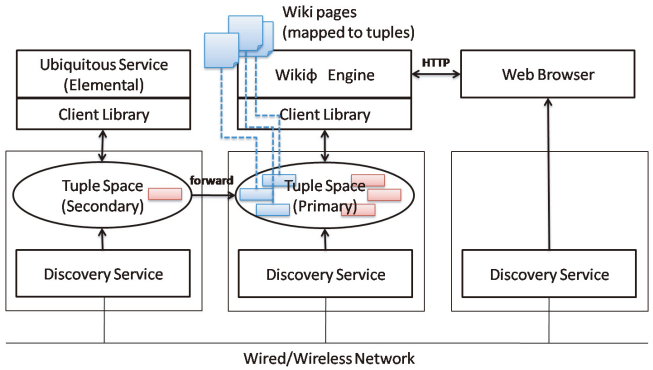

図 2　ヰキ$_\phi$ の実装概要
Fig. 2　Overview of the implementation of Wiki$_\phi$.

あるサービスエリア内には，複数のサーバ機器が存在しうるが，その中の 1 台で主タプルスペースサービス（Primary Tuple Space）が動作し，Wiki$_\phi$ エンジンをホスティングする．他のサーバ機器上では副タプルスペースサービス（Secondary Tuple Space）が動作し，要素サービスからのタプル操作のリクエストは，主タプルスペースサービスへと転送される．

開発者は Web ブラウザを通じて Wiki$_\phi$ エンジンにアクセスし，Wiki ページとして管理される要素サービスの設定情報や複合サービス定義を必要に応じて閲覧・編集する．

4.2 実装環境

ヰキ$_\phi$ プロトタイプの実装環境は次のとおりである．まず，Windows XP Professional SP2 の Cygwin 上で開発したが，特定のオペレーティングシステム環境には依存しない実装になっている．タプルスペースの実装としては，Rinda[35] を採用した．また，ディスカバリサービスには Bonjour を利用している．クライアントライブラリについて，現状では Ruby，Python，Action Script，Java 等のプログラミング言語に対応している．

図 3 は，ヰキ$_\phi$ の基本サービスの配置を示している．左側のサーバ A（Server A）上に主タプルスペースサービス（TS_P）が存在し，そこに基本サービスとしてディスカバリサービス（Discovery Service A），イベントディスパッチサービス（Event Dispatch Service），およびヰキ$_\phi$ エンジン（Wiki$_\phi$ Engine）が配置される．プロトタイプでは，ディスカバリサービスは Python，イベントディスパッチサービスは Ruby を用いて実装した．また，ヰキ$_\phi$ エンジンについては Python およびそのネットワークプログラミング用フレームワークである Twisted を利用して実装するとともに，Web ブラウザでの表示用トップページを HTML，CSS，JavaScript を用いて記述した．また，Ajax ライブラリとして prototype.js[⋆1] および Ext JS[⋆2] を使用した．

そのほか，要素サービスとしてビデオプレーヤ，ビデオクリッピング，プレゼンテーションツール（ビデオとスライドの再生）等を Adobe AIR で，プロジェクト制御サービスを Java で，iTunes 制御サービスを Ruby で実装している．

⋆1 http://www.prototypejs.org/
⋆2 http://extjs.com/

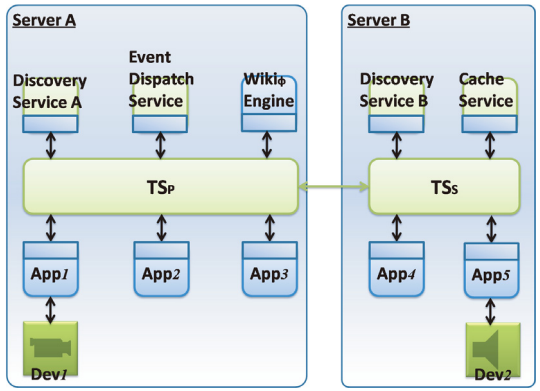

図 3　基本サービスの配置
Fig. 3　Layout of basic services.

4.3　ヰキ$_\phi$ エンジン

本節ではヰキ$_\phi$ エンジンの実装について，さらに詳述する．

4.3.1　データ形式の変換

ヰキ$_\phi$ エンジンは，タプルから Wiki ページへの変換，Wiki ページ内の Wiki コマンドの解釈による HTML 形式への変換という 2 段階の変換過程を経て Web ブラウザでの閲覧用 HTML を生成する．

Wiki ページを編集する際には，Web ブラウザ上で Wiki ページ部分をダブルクリックすることで編集画面に切り替わる．このとき，コンテンツがタプル形式から Wiki 形式に変換される．Wiki 形式のコンテンツの編集を行い，作業終了時に編集画面をダブルクリックすることで，コンテンツは Wiki 形式からタプル形式に再変換されてタプルスペース内に保存される．また，再度 Web ブラウザでの閲覧用に HTML の変換が実行される．

4.3.2　Web ブラウザへの非同期通知

ヰキ$_\phi$ の画面上には Wiki ページ一覧が種類別にフォルダ形式で表示される．現状では，wiki，handler，service タイプの Wiki ページ（すなわちタプル）に分類される．新規 Wiki ページの作成やユビキタスコンピューティング環境への新たな要素サービスの登録等によってタプルスペースに新たなタプルが追加されると，Wiki ページ一覧の情報が非同期に更新される．これは，タプルスペースからヰキ$_\phi$ エンジンに対してイベントタイプのタプルが送信され，さらに，それを受信したヰキ$_\phi$ エンジンが Web ブラウザに対して更新イベントを通知することで実現される．このヰキ$_\phi$ エンジンから Web ブラウザへの非同期イベント通知には，COMET [36] の手法を採り入れている．具体的には，Web ブラウザからの初回のコネクション開設時に，ヰキ$_\phi$ エンジンからタプルスペースに対してイベントタプルを取得する in 操作を実行することで処理をブロックし，後に更新イベントがタプルスペースに out されることでブロックが解除されることで実現される．

```
1: (#podcast\{([^}]*)\}),:podcast
2: (\[>(.*)\]),:tsout
3: (\[<(.*)\]),:tsin
4: (\[=(.*)\]),:tsrd
5: (#flvplayer\{([^}]*)\}),:flvplayer
6: (#ts_result),:tsresult
```

図 4　WikiCommand ページの例
Fig. 4　Example for a WikiCommand page.

```
<a href="#" onclick="ts_op('http://localhost:8084/out/$$1');">
            out $1
</a>
```

図 5　:tsout ページの内容
Fig. 5　Contents of :tsout wiki page.

4.3.3　Wiki コマンド定義による機能拡張

ヰキ$_\phi$ では，WikiCommand という名前の Wiki ページに，1 行 1 エントリで ＜ 正規表現による Wiki コマンドの形式 ＞ “,” ＜ 変換後の HTML を定義する Wiki ページ名 ＞ という形式の変換ルールを記述することで，新たな Wiki コマンドを定義して機能を拡張できる．**図 4** に，WikiCommand ページの例を示す（説明のために各行の先頭に行番号を記してある）．

たとえば，2 行目の変換ルールでは，Wiki ページ中に（`\[>(.*)\]`）の正規表現に一致する文字列があった場合に，その文字列を `:tsout` という Wiki ページに記述された内容で置換するよう定義している．**図 5** に `:tsout` ページの内容を示す．

Wiki ページ内で `[>`タプルの定義`]` という Wiki コマンドを記述しておくと，当該個所がリンクに変換される．Web ブラウザ上でこのリンクをクリックすることで，Wiki 経由で指定したタプルをタプルスペースに out できる．

4.4　クライアントライブラリ

クライアントライブラリの実装の詳細はプログラミング言語ごとに異なるが，基本的には

以下のようなライブラリ関数を要素サービスのプログラム内に挿入して使用する．

- `wf_app_init`：要素サービスの初期化コードに挿入．要素サービスが受けつける外部制御コマンド，要素サービスが生成するイベントおよびデフォルトのイベントハンドラ等の設定情報をタプルスペースに登録
- `wf_deliver_event`：要素サービス内でタプルスペースにイベントを送出する部分に挿入
- `wf_handle_event`：要素サービス内で外部からのイベントに対するハンドラの処理の記述に使用
- `wf_handle_command`：要素サービス内で外部からのコマンド実行要求の処理の記述に使用

Linda の Ruby 版実装である Rinda では，Ruby の分散オブジェクト実装である dRuby に依存しているため，他言語から直接呼び出せない．そのため，各ライブラリ関数の内部では，Rinda のタプルスペースとの通信に介在するプロクシモジュールとメッセージ送受信を行う被覆コードの呼び出しを行っている．被覆コードの内部ではクライアント Socket を生成・管理し，プロクシモジュールへのメッセージ送信およびプロクシモジュールから送出される非同期イベントの受信のための通信端点として利用する．

図 6 は，Python 用のクライアントライブラリを利用したサンプル要素サービスの記述例を示している．wflib モジュールをインポートして，ライブラリ関数を使用する．同様に，既存の Python アプリケーションについてもソースコード内でライブラリ関数を利用して，元々のプログラムに大きな改変をすることなく，ヰキ$_\phi$ のサービスとして登録・利用できる．

```
 1: import wflib
 2:
 3: def handle_command(dict):
 4:   wflib.deliver_event({"to":"destination",
 5:                         "type":"event",
 6:                         "name":"SampleEvent"})
 7:
 8: def handle_event(dict):
 9:   pass
10:
11: if __name__ == '__main__':
12:   wflib.app_init({"id":"sample.local",
13:                   "type":"service",
14:                   "name":"SampleService",
15:                   "cmd_handler":handler_command,
16:                   "event_handler":handle_event})
```

図 6　クライアントライブラリの利用例
Fig. 6　Example usage of a client library.

4.5　イベントハンドラの定義

ヰキ$_\phi$ のプロトタイプ実装では，図 3 で示したとおり，同一ネットワークセグメント内のユビキタスサービスが生成するイベントは，1 度大域的なイベントディスパッチサービスで受信し，そこに登録されたイベントハンドラを検索して実行する．イベントディスパッチサービスが参照するイベント，要素サービス名，およびイベントハンドラの対応表は，EventHandlers という名前の Wiki ページとして管理されており，ヰキ$_\phi$ を通じて編集可能である．また，要素サービスごとに登録されるイベントハンドラも Wiki ページとして管理されているため，ヰキ$_\phi$ を通じて編集可能である．**図 7** では，presenter サービスから CuePoint イベントを受信した際に呼び出されるイベントハンドラ Wiki ページの例を示している．この例では，イベントディスパッチサービスが，goto コマンドを引数 “slide8” でローカルセグメント上の全 presenter サービスに対して実行することになる．

4.6　要素サービス間通信

まず，要素サービス間の制御通信はタプルスペースを通じて行われる．Rinda タプルスペース内ではメッセージは Hash タプルとして表現され，フィールドには識別子（id），メッセージタイプ（type），名前（name），タイムスタンプ（timestamp），送信者（from），受信者（to），データ（data）が含まれる．

各要素サービスは必ずしも Ruby で実装されるとは限らないため，要素サービスと Rinda タプルスペースの間の通信では中間形式として JSON（JavaScript Object Notation）形式[37] の文字列を採用し，各言語の Hash オブジェクトや辞書型オブジェクトとの間で相互変換を行っている．

4.7　アクセス制御

タプルを生成する際に，フィールドにアクセスキーを設定することで，データフィールドを暗号化する機能を提供している．これによって，タプルの検索時に検索用テンプレートに確認用のアクセスキーを与え，タプルスペース側でアクセスキーの正当性を確認し，正しい

```
{"command":"goto", "args":"slide8"} presenter.*
```

図 7　イベントハンドラの定義例
Fig. 7　Example of an event handler.

と確認された場合にのみデータフィールドを復号したタプルを返すことでアクセス制御を実現している．

4.8　資源競合への対応

複合/要素サービスの作成者は，登録時にタプルの共有可能属性（sharable）のフィールドに閲覧 “r”，編集 “w”，実行 “x” の値の組合せを指定しておくことで，要素サービス検索時の検索対象に含めることの可否，Wiki を通じての公開情報（メタデータ）の編集の可否，さらに，複数の複合サービスからの同時利用の可否を指定できる．

ヰキ$_\phi$ プロトタイプでは，共有可能属性フィールドの情報に基づき，ヰキ$_\phi$ エンジンおよびイベントディスパッチサービスで Wiki ページの可視/不可視状態の変更，コマンドやイベントの送信の可否の判定を行うことで，資源競合に対する緩やかな対応を行っている．より形式的かつ厳密な，資源の競合検知および解決機能を必要とする場合には，検知器や解決器をタプルスペースに新たに追加する等の対応が必要となる．

4.9　基本性能評価

本節でヰキ$_\phi$ プロトタイプの基本性能を示す．測定環境は次のとおりである．

- CPU：Dual Core AMD Opteron 880（2.4 GHz）
- 主記憶：32 GB
- OS：Linux 2.6.5 SMP（x86_64）
- ネットワーク：Infiniband（10 Gbps）
- Ruby バージョン：ruby 1.9.0
- Python バージョン：python 2.5.1

2 台のサーバ中，一方で Rinda タプルスペースを動作させ，他方で測定用クライアントプログラムを実行し，タプルスペース上のタプルを rd（read）操作で取得する時間を測定した．タプル操作の呼び出しに (a) dRuby の遠隔メソッド呼び出し，(b) Socket 接続のプロクシによるメッセージパッシング，(c) XML-RPC の 3 種類の方式による実装を用い，100 回の呼び出しの相加平均を計算した．その結果，それぞれ (a) 2.5 msec，(b) 1.4 msec，(c) 16.2 msec となった．

測定に使用したサーバ機の仕様が，ユビキタスコンピューティング環境での使用が想定される組み込みサーバ機器と比べると（現時点では）かなり高性能なため実行時間の測定値自体については参考程度にとどまるが，3 者を比較すると XML-RPC の呼び出しコストが高いことが分かる．プロトタイプシステムの実装では，多言語で記述された要素サービス間での相互運用性を考慮して当初 (c) 方式を採用したが，GUI のボタンを備えた要素サービスでボタンを押下してからハンドラが実行されて別の要素サービス対するコマンド実行が行われるまでの遅延が無視できないほど大きいため，(b) 方式をとることで改善を図った．

次に，非同期イベント通知に要する時間の測定を行った．2 台のサーバ機の一方で Python で実装した要素サービスを起動し，`handle_command` ハンドラ内で `wflib.deliver_event` 関数の呼び出しを行い，自分宛てにサイズ約 126 Byte のイベント送信を行い，もう一方のサーバ上のヰキ$_\phi$ 基本サービス経由で送り返されたイベントによって `handle_event` 関数が呼び出されるまでの往復時間を測定した．同じく 100 回の呼び出しの相加平均を計算した．この結果は 42 msec となった．このとき往路では，`wflib.deliver_event` の呼び出し後にテキストデータの escape 処理，タプルの JSON 形式への変換処理，Socket 経由でのプロクシへの書き込み処理，Rinda タプルスペースへの書き込み処理の順に実行される．復路では，Rinda タプルスペースからプロクシへのイベント通知を経て，要素サービスが利用するライブラリの Socket にイベントメッセージの書き込みが行われ，JSON 形式からタプルへの変換処理，テキストデータの unescape 処理を行った後に，`handle_event` の呼び出しが順次実行される．

最後に，1 台のサーバ上で Rinda タプルスペースに異なる個数のタプルを蓄積した状態で，クライアントでタプルを検索・取得するまでの時間の測定を行った結果を**図 8** に示す．蓄積されるタプルの個数が増加するに従って，検索・取得に要する時間が指数関数的に増加

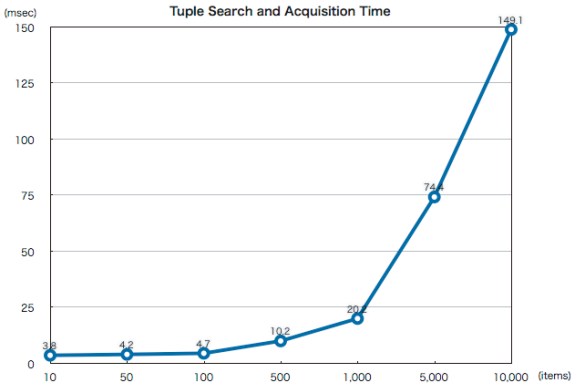

図 8　タプルの検索と取得時間
Fig. 8　Time of tuple search and acquisition.

するのが分かる[*1]．先述のタプル操作の呼び出しに XML-RPC を用いた測定結果と比較すると，タプルの個数が 500 から 1,000 の間になると，素早い応答時間を要求されるようなイベント処理に対応するのが難しいことが分かる．

5. 複合サービスの構成例

前章で説明した実装内容をふまえ，本章ではヰキ$_\phi$ プロトタイプ実装を利用した複合サービスの構成例について述べる．

5.1 ヰキ$_\phi$ でサポートする構成方式

ヰキ$_\phi$ を利用して要素サービス間の協調を定義するには，Wiki ページ化された設定情報の個別編集による方式と一括編集による方式がある．すでに見てきたとおり，ヰキ$_\phi$ に登録される要素サービスの設定情報は，システムレベルではタプルとして表現されており，ユーザレベルではそれに Wiki ページの閲覧・編集という操作を通じてアクセスすることになる．

個別編集方式　各要素サービスに対応づけられた Wiki ページにアクセスし，イベントハンドラページの編集や外部制御コマンドの呼び出しを通じて要素サービス間の相互作用を順次個別に記述する．

一括編集方式　複数の要素サービスの協調によって実現される複合サービスを，新たな Wiki ページとして定義する方式．要素サービスごとに作成される Wiki ページとして個別管理される設定情報を 1 つの Wiki ページに集約して管理できるという利点がある．

5.2 複合サービス定義用 Wiki コマンド

複合サービスを定義する際に使用する主な Wiki コマンドの例を**図 9** に示す．

```
// 初期化コード
#init_service{"ServiceName", "Destination"}

// サービスが受けつけるコマンドの登録
#export_command{"ServiceName", "CommandSpec", "HandlerName"}

// サービスが受けつけるコマンドの登録解除
#unexport_command{"ServiceName", "CommandSpec"}

// コマンドの送出
#exec_command{"Destination", "CommandSpec"}

// イベントの送出
#deliver_event{"Destination", "EventSpec"}

// イベントハンドラの登録
#register_handler{"Destination", "EventName", "HandlerName"}

// イベントハンドラの登録解除
#unregister_handler{"Destination", "EventName"}
```

図 9　主な複合サービス定義用 Wiki コマンド
Fig. 9　Major wiki commands to define composite services.

5.3 応 用 事 例

本節では，複合サービス定義の具体的な流れを示すための応用事例として，ビデオクリップメイルサービスの構成方法について述べる．さらに，複合サービスの構成可能性について検討する．

5.3.1 ビデオクリップメイルサービス（ClipMail）の定義例

まず，本複合サービスは，動画プレーヤ要素サービス，動画クリッピング要素サービス，メイル送信要素サービスとで構成され，動画プレーヤ要素サービス GUI の再生ボタンの押下から停止ボタンの押下までに再生された部分を動画クリッピング要素サービスが切り出して保存し，その URI をメイル送信要素サービスの機能によって通知するというサービスを提供する．

図 10 に関連する Wiki ページの内容を示すが，ClipMail 複合サービス自体の定義は，複合サービスが外部公開するコマンド（`send_mail`）とハンドラの対応を登録（3–4 行目）するコマンド定義部分と，動画プレーヤ要素サービスの再生ボタンの押下イベント（PlayButtonPressEvent）へのハンドラを登録（5–6 行目），動画プレーヤ要素サービスの停止ボタンの押下イベント（StopButtonPressEvent）へのハンドラを登録（7–8 行目），さらに，動画クリッピング要素サービスの切り出し終了イベント（ClipDoneEvent）へのハンドラを登録する（9–10 行目）イベントハンドラ登録部分とからなる．

複合サービスの定義の雛型を作成した段階で，各ハンドラの定義に入る．“[[...]]” で囲まれた部分が各 Wiki ページへのリンクになるので，適宜リンクをクリックしてハンドラの Wiki ページに移動し，ハンドラ呼び出し時に実行されるべき処理を Wiki ページの編集を通じて定義する．本事例では，Wiki コマンド `#exec_command` によって各要素サービスが外部公開しているコマンドを送出・実行している．この作業を行う際，どの要素サービスがどのようなコマンドを公開しているかについては Wiki ページの一覧から当該要素サービスの設定ページを選択・閲覧することで確認できる．

*1 文献 35) で，タプル群を線形に探索するのでスケーラビリティに問題がある旨，作者が言及している．

```
 1: // ClipMail 複合サービスの定義
 2: #init_service{[[ClipMail]], "clipmail.local"}
 3: #export_command{[[ClipMail]], "send_mail",
 4:                 [[ClipMailHandler]]}
 5: #register_handler{"player.snufkin", "PlayButtonPressEvent",
 6:                   [[PlayButtonPressHandler]]}
 7: #register_handler{"player.snufkin", "StopButtonPressEvent",
 8:                   [[StopButtonPressHandler]]}
 9: #register_handler{"clipper.snufkin", "ClipDoneEvent",
10:                   [[ClipDoneHandler]]}
```

```
// ClipMailHandler ページの内容
#exec_command{"sendmail.moomin",
              {"command":"send",
               "args":{"to":$1, "subject":$2, "body":$3}}}
```

```
// PlayButtonPressHandler ページの内容
#exec_command{"clipper.snufkin", {"command":"clip_start"}}
```

```
// StopButtonPressHandler ページの内容
#exec_command{"clipper.snufkin", {"command":"clip_stop"}}
```

```
// ClipDoneHandler ページの内容
#exec_command{"sendmail.*",
              {"command":"send",
               "args":{"to":$1, "subject":$2, "body":$3}}}
```

図 10　ClipMail 複合サービスの定義例
Fig. 10　Example of the definition of ClipMail composite service.

5.3.2　複合サービスの構成可能性

ビデオクリップメイル複合サービスの定義例で見たように，本研究では基本的には単純なコマンド・アンド・レスポンス型の動作をする要素コンポーネントを想定している．このような要素コンポーネントをパイプライン式に接続する複合サービス，あるいは図 1 (a) で示したように，GUI を構成するボタン等の個々の部品に要素コンポーネントを対応づけて実現される複合コンポーネントの構成が可能である．また，複合コンポーネントを要素コンポーネントとして階層的に複合コンポーネントを構成することも可能である．要素コンポーネント間の通信は，メッセージ通信はタプルスペースを介した間接通信によって実現され，メッセージの内容は data フィールドのサイズにも依存するが，数十 Byte から数 KByte 程度のテキストデータである．データ通信については，3 章で述べたように，メッセージ通信によって URI 等の識別子の情報を交換した後に，要素コンポーネント間で独自にネゴシエーションを行い適宜データの送受信を行う．その際，データ送信開始，データ受信完了等のイベントについてはメッセージ通信の枠組みで取り扱える．

Wiki の枠組みに従うという制約下でのラピッドプロトタイピング支援で，どこまで有用な複合サービスを構成できるかは今後利用経験を重ねる必要がある．文献 33) では，Wiki を Java プログラム開発のフロントエンドとして活用し，パッケージやクラスの定義を Wiki ページに対応づける手法について提案している．Wiki 内で通常のプログラミング言語による開発を支援する手法と，本研究で示した Wiki コマンドの組合せでユビキタス複合サービスを記述する手法とはアプローチは異なるが，Wiki によるプログラミング支援のあり方について今後さらに研究を進める必要がある．

さらに，要素サービス間のより複雑なインタラクションを実現するためには，文献 23), 38), 39) に見られるように，ビジネスプロセスを記述するために BPEL，OWL-S，あるいはそのほかの Web サービスや SemanticWeb 関連技術を導入することも今後選択肢として考えられる．その際，我々のアプローチとしては，Wiki エンジンが Wiki 記法を HTML に変換するように，一般開発者が用いる何らかの簡便な記法を XML の記述に変換するようなしくみを導入することを検討している．

6. ま と め

本論文では，ユビキタスコンピューティング環境に存在するさまざまな要素サービスを組み合わせて複合サービスを実現する際に，サービス開発者が予備知識のない状態でも，開発作業を進めるために必要な情報への一元的なアクセスを可能にし，簡便な操作によって新サービスのプロトタイピングを支援する新たな方式を提案した．本方式では，開発支援ツールのベースとして Wiki を採用することで，要素サービスの設定情報の確認や変更作業，複合サービスの定義作業について，Wiki ページの閲覧や編集作業に対応づけて行えるようになった．また，Wiki の特徴を踏襲しているため，少人数の開発者による複合サービスの共同開発や開発ノウハウの蓄積・共有にも効果的である．

今後の課題としては，本研究を通じて Wiki の基本機能の組合せで複合サービスを独立した Wiki ページとして定義でき，他の Wiki ページと合わせて一元管理できることを確認することで新しい開発の枠組みを提示できたものの，今後は開発工程をさらに簡素にすべく研究を進める予定である．

参 考 文 献

1) Weiser, M.: The Computer for the 21st century, *Scientific American*, Vol.265, No.3, pp.94–104 (1991).

2) Johanson, B., Fox, A. and Winograd, T.: The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, *IEEE Pervasive Computing*, Vol.1, No.2, pp.67–74 (2002).
3) Romaán, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K.: Gaia: A middleware platform for active spaces, *SIGMOBILE Mob. Comput. Commun. Rev.*, Vol.6, No.4, pp.65–67 (2002).
4) Kawahara, Y., Minami, M., Saruwatari, S., Morikawa, H. and Aoyama, T.: Challenges and Lessons Learned in Building a Practical Smart Space, *MobiQuitous*, pp.213–222, IEEE Computer Society (2004).
5) 徳田英幸，中澤　仁，岩井将行，由良淳一，村瀬正名：ユビキタスコンピューティングの世界を実現する革新的ネットワーク技術：2. ユビキタス空間を融合するネットワーク技術への課題，情報処理，Vol.43, No.6, pp.623–630, 社団法人情報処理学会 (2002).
6) 榎堀　優，西尾信彦：管理主体の異なるユビキタスシステム間の連携，*The 8th JSSST SIGSYS Workshop on Systems for Programming and Applicatoins* (*SPA2005*), pp.234–242, 日本ソフトウェア科学会 (2005).
7) Hodes, T.D., Katz, R.H., Servan-Schreiber, E. and Rowe, L.: Composable Ad-hoc Mobile Services for Universal Interaction, *Proc. MOBICOM'97*, pp.1–12 (1997).
8) Hodes, T.D. and Katz, R.H.: A Document-based Framework for Internet Application Control, *The 2nd USENIX Symposium on Internet Technologies and Systems* (*USITS '99*), pp.59–70 (1999).
9) Johanson, B. and Fox, A.: The Event Heap: A Coordination Infrastructure for Interactive Workspaces, *WMCSA '02*, *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, Washington, DC, USA, p.83, IEEE Computer Society (2002).
10) Grimm, R.: One.world: Experiences with a Pervasive Computing Architecture, *IEEE Pervasive Computing*, Vol.3, No.3, pp.22–30 (2004).
11) Picco, G.P., Murphy, A.L. and Roman, G.-C.: LIME: Linda meets mobility, *ICSE '99*, *Proc. 21st International Conference on Software Engineering*, Los Alamitos, CA, USA, pp.368–377, IEEE Computer Society Press (1999).
12) Davies, N., Friday, A., Wade, S.P. and Blair, G.S.: L^{2}imbo: A distributed systems platform for mobile computing, *Mobile Networks and Applications*, Vol.3, No.2, pp.143–156 (1998).
13) Gelernter, D.: Generative communication in Linda, *ACM Trans. Prog. Lang. Syst.*, Vol.7, No.1, pp.80–112 (1985).
14) Spiteri, M. and Bates, J.: An architecture to support storage and retrieval of events, *Proc. MIDDLEWARE 1998, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing* (1998).
15) Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O. and Spiteri, M.: Generic Support for Distributed Applications, *Computer*, Vol.33, No.3, pp.68–76 (2000).
16) Edwards, W.K., Newman, M.W., Sedivy, J. and Izadi, S.: Challenge: Recombinant computing and the speakeasy approach, *MobiCom '02*, *Proc. 8th Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, pp.279–286, ACM Press (2002).
17) Ichiro, S.: Dynamic federation of partitioned applications in ubiquitous computing environments, *Pervasive Computing and Communications, 2004, PerCom 2004*, *Proc. 2nd IEEE Annual Conference*, pp.356–360 (2004).
18) Chinnici, R., Gudgin, M., Moreau, J.-J. and Weerawarana, S.: Web Services Description Language (WSDL) Version 1.2, Technical Report, W3C (2002).
19) Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn., N., Nielsen, H.F., Thatte, S. and Winer, D.: Simple Object Access Protocol (SOAP) 1.1, W3c note, W3C (2000).
20) UDDI Spec Technical Committee Draft 3.0.2, Oasis committee draft (2004).
21) Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. and Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1 (2003). http://www.ibm.com/developerworks/library/specification/ws-bpel/
22) Burstein, M., Hobbs, J., Lassila, O., Mcdermott, D., Mcilraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. and Sycara, K.: OWL-S: Semantic Markup for Web Services, Website (2004).
23) 山登庸次，中辻　真，須永　宏：ユビキタス環境で動的にサービス実現するためのサービス合成技術，情報処理学会論文誌，Vol.48, No.2, pp.562–577 (2007).
24) Gregorio, J. and de hOra, B.: The Atom Publishing Protocol, RFC 5023, Internet Engineering Task Force, Network Working Group (2007).
25) Leuf, B. and Cunningham, W.: *The Wiki way: Quick collaboration on the Web*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001).
26) Dahlbäck, N., Jönsson, A. and Ahrenberg, L.: Wizard of Oz studies: Why and how, *IUI '93*, *Proc. 1st International Conference on Intelligent User Interfaces*, New York, NY, USA, ACM Press, pp.193–200 (1993).
27) Li, Y., Hong, J.I. and Landay, J.A.: Design Challenges and Principles for Wizard of Oz Testing of Location-Enhanced Applications, *IEEE Pervasive Computing*, Vol.6, No.2, pp.70–75 (2007).
28) Ballagas, R., Szybalski, A. and Fox, A.: Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments, *PerCom 2004*, *2nd IEEE International Conference on Pervasive Computing and Communications*, Orlando, Florida, USA (2004).

29) Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.-M.: The many faces of publish/subscribe, *ACM Comput. Surv.*, Vol.35, No.2, pp.114–131 (2003).
30) Munson, M., Hodes, T., Fischer, T., Lee, K.H., Lehman, T. and Zhao, B.: Flexible Internetworking of Devices and Controls, *Proc. IECON* (1999).
31) 江渡浩一郎：Wiki の起源と進化，情報処理学会研究報告，Vol.2007-HCI-123, No.41, pp.11–18 (2007).
32) Mochizuki, M.: An Experimental CALL System Enhanced with Wiki, *7th International Conference on Advanced Learning Technologies* (*ICALT*), pp.210–214 (2007).
33) Xiao, W., Chi, C. and Yang, M.: On-line collaborative software development via wiki, *WikiSym '07*, *Proc. 2007 International Symposium on Wikis*, New York, NY, USA, pp.177–183, ACM (2007).
34) Nakazawa, J., Tokuda, H., Edwards, W.K. and Ramachandran, U.: A Bridging Framework for Universal Interoperability in Pervasive Systems, *26th IEEE International Conference on Distributed Computing Systems* (*ICDCS2006*) (2006).
35) 関　将俊：dRuby による分散 Web プログラミング，オーム社 (2005).
36) Russell, A.: COMET - the next stage of AJAX (2006). http://www.irishdev.com/NewsArticle.aspx?id=2166
37) Crockford, D.: The application/json Media Type for JavaScript Object Notation, RFC 4627, Internet Engineering Task Force, Network working group (2006).
38) Jung, J.-Y., Park, J., Han, S.-K. and Lee, K.: An ECA-based framework for decentralized coordination of ubiquitous web services, *Inf. Softw. Technol.*, Vol.49, No.11-12, pp.1141–1161 (2007).
39) Chakraborty, D. and Lei, H.: Pervasive Enablement of Business Processes, *PERCOM '04*, *Proc. 2nd IEEE International Conference on Pervasive Computing and Communications* (*PerCom'04*), Washington, DC, USA, p.87, IEEE Computer Society (2004).

(平成 19 年 9 月 7 日受付)
(平成 20 年 2 月 5 日採録)

望月　祐洋（正会員）

2000 年慶應義塾大学大学院政策・メディア研究科博士課程修了，博士（政策・メディア）．2008 年現在，東京工業大学学術国際情報センター准教授．ユビキタスコンピューティング，分散コンポーネント，ミドルウェア等の研究に従事．IEEE-CS，ACM 各会員．