分散アプリケーション構築操作を複数種インタフェースから可能にする研究

岩井 将行[1], 中澤 仁[1], 徳田 英幸[1,2]

ユビキタスコンピューティング環境において，情報家電機器やセンサがネットワークに接続され，多様に遍在することになる．しかし誰が，それらの情報家電機器やセンサを抽象化するソフトウェアコンポーネント（部品）を組み合わせ，分散ユビキタスアプリケーションの構築を行うのかは差し迫った課題である．特に家庭内においては，ランニングコストや即時性の観点から，ユーザ自身が，それらのソフトウェアコンポーネント間を自由に組み合わせられる機構が不可欠である．本論文で，分散アプリケーションを複数種類のユーザインタフェースから構成できる独立型モデリングトポロジを提案し，それらを実現するミドルウエア uBlocks を提案する．

1 はじめに

近年，ネットワークに接続可能なセンサや情報家電を建物や部屋などに遍在させ人間生活を支援する，ユビキタスコンピューティング環境（以下ユビキタス環境と呼ぶ）が注目されている．しかし，ユビキタス環境の普及に伴い，ネットワークに接続されたセンサや情報家電が小型化し，それらの数が増加すると以下のような問題点が発生する．

第一の問題点として，ユーザ自身によって現状で動作しているデバイスの把握が困難になる．壁に埋め込まれるなどして日常生活で目にすることの無い組込デバイスの多くは，ユーザの記憶から消し去られてしまい，有効に活用できなくなる可能性がある．ユーザが存在を認識していなければ，そのデバイスに対して新しい役割を与えることさえできない．把握できない組込デバイス群は，組合せて利用することも困難となる．

第二の問題点は，開発者が初期に構築した組合せの内容をユーザが長期的に使い続けなければならなくなることである．ユーザの意識や嗜好が変化しても，システムをすぐさま適応させることはできなくなる．例えば,「ユーザ A が入室したら B という動作を行い，退出したら C という動作を行う」という設定を開発者に行ってもらってから，生活スタイルの変化や季節によって，ユーザの要求との間にズレが生じた場合には，仕方なく利用するか，その分散アプリケーションを停止させるしかない．また，開発者やシステムエンジニアなどに再度構築を依頼することは，経済的なコストが発生するため不適切と言える．こういった問題点が考えられる現状では，専門的な開発者によって構築された分散アプリケーションを一方的にしかユーザが利用できず，ユビキタス環境に対してユーザ側から新たな役割を与えることはできない．ユーザへのアプリケーションの一方的な**押しつけ**になっている状況は，本当の意味でユーザがユビキタス環境の恩恵を受けているとは言えない．

以上の状況を踏まえ著者らは，ユビキタス環境に存在するデバイスをユーザが視覚的に把握でき，さらにそれらの組合せをユーザ自身によって変更可能

[1] 慶應義塾大学大学院 政策・メディア研究科
[2] 慶應義塾大学 環境情報学部
本研究は情報処理振興協会平成 14 年度未踏事業として採択されました。
2003 年 10 月 17 日受付.

にしたミドルウエア「uBlocks」を提案する．図 1 に uBlocks のスクリーンショットを示す．

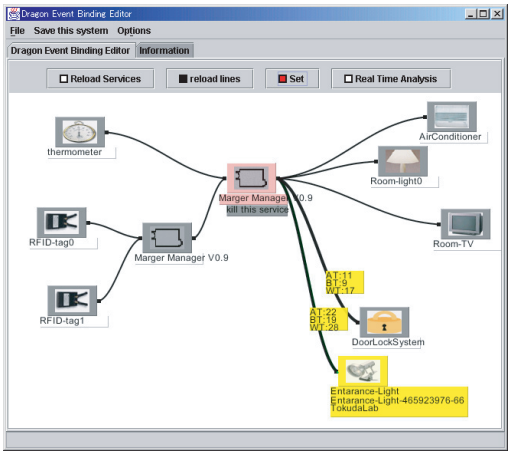

図 1 uBlock Browser により分散アプリケーションを構築しているスクリーンショット

uBlocks は，ユビキタス環境のためのソフトウェア開発者の負荷を軽減できるだけでなく，ユーザ自身による分散アプリケーション構築を可能にする．分散アプリケーションの構成を変更する際に，複数種類（マルチモーダル）の視覚的なユーザインタフェースから一貫性をもって操作可能な機構を備えている．本論文でのマルチモーダルという用語は，「複数の表示系を持つグラフィカルユーザインタフェースを同時に別々のユーザが，1 つのユビキタス環境に対して利用すること」として用いている．

以下に本論文の構成を示す．第 2 章では，本論文の想定環境の説明及び，ソフトウェアコンポーネントを用いた分散アプリケーション構築者について議論する．第 3 章では，ソフトウェアコンポーネント，モデラ，ユーザインタフェースの 3 つのプレイヤ要素が構成する独立型モデリングトポロジを説明し，複数個のインタフェース間で一貫性を保つ手順を述べる．第 4 章では，uBlocks のマルチモーダルなブラウザインタフェースについて説明する．第 5 章で，uBlocks を用いて構築できるアプリケーション例を述べ，第 6 章で関連する研究に対する uBlocks の優位点を述べる．最後に，今後の研究課題を示し，本論文をまとめる．

2 ユーザ自身による分散アプリケーションの構築要件

2.1 想定する分散アプリケーションの定義

現在のユビキタス環境は，組込機器や情報家電機器，センサなどを部屋の壁などへ埋め込み，分散配置されたものをネットワーク上で動作させる現実的な手法によって構築されている．こういった機器の OS 上で動作し，センサやハードウエアの情報をデータとして抽象化しているものを，**ソフトウェアコンポーネント**と呼ぶ．ユビキタス環境を想定したアプリケーションの多くは，ソフトウェアコンポーネントからのセンサ情報取得やそれを介した機器操作などを通じて実世界に関係した情報を共有し，ユーザ支援をすることが前提になっている．ユビキタス環境におけるアプリケーションは，「センサの情報を取得し」，「その情報を加工し」，「実世界に何らかの動作を行いユーザを支援する」というデータの流れに着目するデータフロープログラミングモデルとなっている．こういった分散されるソフトウェアコンポーネント（以下コンポーネント）を連携し，その中でデータの流れに着目してユーザを支援するアプリケーションを，本論文で対象とする**分散アプリケーション**とする．データの流れに着目するデータフロープログラミングは，ユビキタス環境にとどまらず，ウェブサービスプログラミングにおいても，ページ間でセッションデータを受け渡すことに着目した開発が行われる [4]．近年の複雑化するシステム開発において頻繁に使われるプログラミングモデルであり，本論文で想定するのに妥当である．

2.2 分散アプリケーション構築者についての議論

これまでの分散アプリケーション開発は，計算機間での通信のみを主に考慮すれば良かった．これに対して，ユビキタス環境における分散アプリケーションは，ユーザが生活している場面で利用される．そのため，ソフトウェアコンポーネント開発者，分散アプリケーション構築者，ユーザの役割が異なる．ソフトウェアコンポーネント開発者とは，ハードウエアメーカや研究機関などによって作成されるハードウエアに依存したコードを実装する人物である．コンポー

ネント開発者は，近年，他のコンポーネントとの接続部分を考慮に入れ開発する必要に迫られている．分散アプリケーション構築者とは，メーカや作成者の異なるコンポーネント間を結び合わせ，アプリケーションを構築する人物であり，各コンポーネントやソフトウェア技術に関して深い知識を必要とする．ユーザとは，日常的な生活の中で分散アプリケーションの恩恵を受ける人物である．家庭内などを想定した場合は，居住する人物全てに相当する．

本章では，ソフトウェアコンポーネント開発者，分散アプリケーション構築者，ユーザの関係を明らかにし，本研究が目的とするユビキタス環境に適した 3 者の関係を示す．

コンポーネント開発者 = 分散アプリケーション構築者 ≠ ユーザ

既存の多くの分散システムは，コンポーネント開発者がコンポーネントの配置も行い，分散アプリケーションを稼働状態にする．大規模なシステム開発の場合は，コンポーネント間の組み合わせも同一のプログラマがあらかじめ設定してしまう．信頼性が必要なシステムで，テストやチューニングもコンポーネント開発者本人が行う方がよいが，構築者に要する人件費などのランニングコストが増加する．ユーザは，完成した分散アプリケーションを一方的に利用するだけである．ユビキタスアプリケーションの開発をシステム開発者にのみに全て任せてしまうと，アプリケーションが画一化されユーザにとって意義のあるアプリケーションを提供することが困難になる．

コンポーネント開発者 ≠ 分散アプリケーション構築者 ≠ ユーザ

開発者でもなく，ユーザでもない，専門の構築を行う人物（システムエンジニア）が，分散アプリケーションの構成を行う．この構築者は，分散されるソフトウェアコンポーネント間のつながりを記述する．記述とは例えば,「ホスト A にある B コンポーネントの C 機能を引数 D によってホスト F 上の G コンポーネントの H 機能と接続する」などである．この記述を基に分散アプリケーションを動的に構成し，実行される．記述形式が標準化され普及すれば，あるアプリケーションに関する記述を構築者が販売し，ユーザが購入するなどのビジネスモデルを有することが可能である．VNA [9] の VNA Markup Language(VML) や AMIDEN [18]，さらにエンタープライズのシステム開発のコンポーネント開発者と構築者を切り分ける EJB の Deployment Descriptor [17] など多くの記述形式が研究されている．しかし構築者は，高度なスキルとコンポーネントに関する詳細な知識を必要とする人物に限られることが多い．そのため家庭内などのユビキタスコンピューティング環境では，分散アプリケーションの構成を常に変更できない．

コンポーネント開発者 ≠ 分散アプリケーション構築者 = ユーザ

家庭内のレガシな AV 機器のケーブル配線は，ユーザ自身が行ってきた．同様に，ユビキタス環境がターゲットとするホームネットワークにおいても，ユーザ自身に分散アプリケーションを構築できることが望ましい．ユーザが分散コンポーネントの配置とアプリケーションの構築を行うメリットとしては，季節，時間帯，その場の状況に最適な分散アプリケーションに，時間や経済的コストをかけないで切り替えることができる点である．本論文では，この「コンポーネント開発者 ≠ 分散アプリケーション構築者 = ユーザ」の役割分担に着目して議論を進めていく．

2.3 ユーザ自身による分散アプリケーションの構築要件

ユーザ自身が分散アプリケーションの構築を行うには課題もある．ユーザが現在の分散コンポーネントの存在を知り，それらの関連を把握する必要がある．さらに，プログラミング習熟度のないユーザにも分散アプリケーションを構築可能とするため，多様な構築用ブラウザが必要になる．ユーザが複数存在する場合には，構築用のユーザインタフェースも複数個必要になり，それらのインタフェース間で一貫性を保って表示させることが必要である．

2.3.1 システムの複雑性と接続操作単純性

ユビキタスコンピューティング環境においては，目的や作成者の異なるコンポーネントを接続するためには，それぞれ個別の複雑なプロトコルに適応させる必要がある．しかしユーザにその設定を行わせるのは

困難がある．そのため，ユーザ側には接続と切断という非常に単純な概念のみを提示し，ミドルウェア内部でそれを隠蔽する必要がある．

2.3.2 コンポーネント間の接続モデルの一貫性

家庭内などでの分散アプリケーションの構築は，ユーザは複数人が存在することを想定しなければならない．複数人からの設定を行う場合には，分散コンポーネント間の接続モデルを，全てのユーザインタフェースで一貫性をもって管理できる必要がある．コンポーネント追加の反映，ソフトウェアコンポーネントの削除，コンポーネント間の接続状態の収集，コンポーネント間の接続状態変更の反映などである．

2.3.3 マルチモーダルなユーザインタフェース

ユーザのコンピュータに対するスキルは様々なレベルである．さらに，ユーザが日常生活で直面しているコンピュータの利用環境も，外出先や移動中など様々な場面がある．この様なユーザ独自の好みや環境に対応するために，文字型インタフェース，GUI インタフェース，さらに携帯端末向け Web-based インタフェースなど複数種類のインタフェースを用意し，アプリケーション構築の利便性を向上させる必要がある．

3 uBlocks の概要

本章では，第 2.3 節の機能要件を満たすための，uBlocks のメカニズムを説明する．uBlocks では，ソフトウェアコンポーネント間の通信ドルウエア Dragon と，ユーザに分散アプリケーションの構築を視覚的に行わせるインタフェース uBlock Browsers からなる．

3.1 複雑さを隠蔽するコンポーネント間通信機構

Dragon[5][6][12] は，コンポーネント間 Remote Method Invocation(RMI) を用いた通信ミドルウエアである．コンポーネント間インタフェースは，コンポーネントの機能の量に伴って複雑化する．我々は複雑化を避けるため，コンポーネント間の通信を，Java RMI のプログラミングインタフェース *notify(evt)* の引数として，イベントオブジェクトを通知することにより実現する．イベントオブジェクトの受け渡しのみに通信機構を限定することで，第 2.3.1 項に示したコンポーネント間通信の複雑化を避けている．イベントオブジェクトとしては，直列化したオブジェクトおよび遠隔呼び出し可能なオブジェクトの 2 種類が配送可能である．通信によって受け渡される内容をイベントオブジェクトという形で統一し，統一したイベントオブジェクトは，コンポーネント間で図 2 の様に受け渡される．各コンポーネントは，Event Filtering Module を内部に保持する．Event Filtering Module は，図 2 の様にイベントオブジェクトを変換したり，イベントオブジェクトの通過を制限したりするなどの機能があり，コンポーネント開発者が自由に拡張して開発できる部分である．Event Filtering Module によってコンポーネント間のイベントは，各機能に適宜マッピングされる．

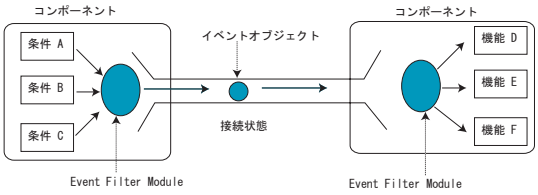

図 2 複雑なコンポーネント間通信の隠蔽

配送されるイベントオブジェクトやイベントオブジェクト到着後の挙動は，各コンポーネント開発者の実装依存であるが，開発者が実装する際に利用する最も基本的なイベントの型を示す．イベントとして伝えられるオブジェクトは，階層としてコンポーネント開発者に提供される．コンポーネント開発者は，リフレクションを用いて，伝達されたイベントの振り分けやイベント伝達後の制御を実装する．

以下に，Dragon で提供する代表的なイベントを説明する．

org.ublocks.event.SensorSourceEvent センサから発生するイベントオブジェクトのスーパクラスとして利用する．本クラスを継承して，位置情報に関するセンサや気象に関するクラスなどを作成する．

org.ublocks.event.ApplianceControlEvent 機器制御のイベントオブジェクトのスーパクラスとして利用する．本クラスを継承して，CD Player やラ

イトコントロールのイベントを作成する．

org.ublocks.event.rmitype.AsyncBack イベントオブジェクトが本クラスを継承していれば，到着後に送り元コンポーネントに対して非同期情報の通達が可能である．実行結果などを元のコンポーネントに伝達したい場合に有効である．例えば ON/OFF の命令が成功したかどうかの確認が必要な場合などに利用する．

org.ublocks.event.rmitype.Pull イベントオブジェクトが本クラスを継承していれば，到着後に送り元コンポーネントに対して最新の情報を pull させることが可能である．本イベントの到着先のコンポーネントから呼び出して最新情報を伝達したい場合に利用する．

3.2 コンポーネント間の接続と意味ある分散アプリケーション構築について

前節で述べたようにコンポーネント間のイベントは，汎用性を持たせてある．そのためコンポーネント間で見かけ上イベント通知が行われていても，互いのコンポーネントがそのイベント処理を正しく実装し，意味ある分散アプリケーションが実行できない場合がある．4 つのコンポーネント間の接続状態を列挙する．

I. 切断状態 コンポーネント A とコンポーネント B は接続していない．イベント通知も行われない．

II. 不完全な接続状態 コンポーネント A とコンポーネント B はイベントを受け渡しているが，イベントの意味ある処理が実装できていない．Event Filtering Module などを用いて処理可能なイベントに変換させなければならない．

III. 部分的な接続状態 コンポーネント A とコンポーネント B は接続し且つ，到着するイベントをスーパクラスでキャストすることにより，部分的に処理して，互いのコンポーネント間で協調する挙動を一部実行できている．

IV. 完全な接続状態 コンポーネント A とコンポーネント B は接続し且つ，完璧に，互いのコンポーネントのイベントを処理する意味のある分散アプリケーションを構築できている．

ユビキタスコンピューティング環境などにおいては，1 つの独立して動作するコンポーネントが様々な他のコンポーネントと通信を行う場面が想定できる．例えば，IV の「完全な接続」のみを目指した場合あるコンポーネントは，通信相手のコンポーネントを限定し，その内部構造を互いに完全に理解しなくてはならない．ユビキタス環境においては，メーカも目的も異なるソフトウェアコンポーネントが多数存在することは容易に想定でき,「完全な接続」を目指すことは開発に負担である．uBlocks は全体として III の状態をより多く作ることを目指し，ソフトコンポーネントやイベントクラスの設計及び実装を行った．

3.3 uBlocks のプレイヤとそのトポロジ

ユーザが構築する分散アプリケーションを次の 3 つのプレイヤに抽象化する．

ソフトウェアコンポーネント (S) ソフトウェアコンポーネントは，ハードウェアを意識することなく遍在するソフトウェアを示す．各ソフトウェアコンポーネントは，動作時は直接コンポーネント間で協調を行うが，常にどのコンポーネントと接続しているかという情報をモデラに通知可能でなければならない．モデラに送信されるデータは，接続先ベクトル $\overrightarrow{m_i}$ である．

モデラ (M) モデラは，分散されるソフトウェアコンポーネントから部分的に接続状態をを収集し，現時点での全てのコンポーネントの接続構成（モデル）を収集しキャッシュとして保持する．この全体モデルは，ユーザインタフェースに通知される．つまり $(\overrightarrow{m_1}, \overrightarrow{m_2}, \ldots \overrightarrow{m_n})$ の接続モデルをユーザインタフェースに通達する．モデラは，Jini [16] の Look up Service を改良する形で実装を行った．

ユーザインタフェース (G) ユーザインタフェースは，GUI などのユーザとの直接的なインタラクションを持つ部分である．全体モデル $(\overrightarrow{m_1}, \overrightarrow{m_2}, \ldots \overrightarrow{m_n})$ から，ユビキタス環境の情報をユーザに視覚的に提示する．ユーザは，この情報を基に分散アプリケーションの接続を行うことが

可能であり，接続命令を発行する．

著者らは，プレイヤのトポロジを，サービスサイドモデリングトポロジ (各ソフトウェアコンポーネントごとにモデラを保持する)，ユーザサイドモデリングトポロジ（各ユーザインタフェースごとにモデラを保持する），独立型モデリングトポロジ（モデラは独立してひとつである）の 3 つの可能性について数理的に検討している [8]．その結果組込機器や携帯端末などの処理資源を制限されている場合には，独立型モデリングトポロジがユーザインタフェースやコンポーネントに処理が集中しないものであると言えた．特にユーザインタフェースの数が複数ある場合でも，コンポーネント自身のトラフィックには影響を与え難い．図 3 の様に各プレイヤが別々に分散して存在している独立型モデリングトポロジである．

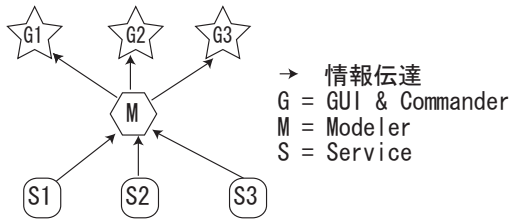

図 3 採用した独立型モデリングトポロジ

uBlocks においては，一環境にモデラ 1 つを前提にしている．もしあるモデラに障害が発生している場合は，待機していたセカンダリがモデラになるように設定できる．モデラは，コンポーネント間の通信に直接関わらないため，モデラの入れ替わりは，既に構築している分散アプリケーションには影響はない．しかし，モデラの耐故障性を上げるために同時に複数のモデラを起動した場合には，その和集合のモデリングがユーザインタフェース側で採用される．2 個以上のモデラの同時起動の是非に関する議論は，文献 [8] で扱っている．

この独立型モデリングトポロジを基に実装した uBlocks は，ソフトウェアコンポーネント発見，命令発行，コンポーネント間通信，コンポーネント変化検知を行う．

コンポーネント発見 分散されるソフトウェアコンポーネントを発見，収集し，ユーザに提示する．この際ユーザインタフェースが複数個あった場合でも，一貫してユーザに表示できなければならない．

命令発行 ユーザは，ユーザインタフェースの表示情報を基にコンポーネントに対して接続や切断を行う命令を発行する．同時に命令を行った場合の命令競合も考慮する．

コンポーネント間通信 その命令を受け取ったコンポーネントは，コンポーネント間で命令に従い，接続/切断を行う．

コンポーネント変化検知 コンポーネントの状態に変化があった場合はそれをユーザに通知する．

以下の節では，ユーザインタフェースを中心に uBlocks の起動時，命令時，競合命令時に区別して，一貫性をもった分散アプリケーションの構築手法を説明する．

3.4 uBlock Browser 起動時の一貫性

ユーザインタフェースである uBlock Browser 起動時の動作手順を図 4 に示す．uBlock Browser は，起動するとモデラを通じて，既に起動しているコンポーネントの結合状況を収集し，ユーザに現在の状態を通知する．図 4 では，ソフトウェアコンポーネント S_1 から S_1 と S_2 が接続されているという意味の $S_1 \rightsquigarrow S_2$ をモデラが受け取っている様子を示している．ここで $A \rightsquigarrow B$ は，A と B が接続されていることを意味している．S_2 と S_3 が接続されているソフトウェアコンポーネント S_2 からは，$S_2 \rightsquigarrow S_3$ という接続モデルを収集する．各 uBlock Browser でこれらのモデルを収集し，ユーザインタフェースに対して図 1 の画面の様に表示させる．

既にソフトウェアコンポーネントが動作している状態で，新たに uBlock Browser G_3 が起動した場合，モデラに対して，モデルに変化があった場合に自分自身に通知を行うように要請（モデルの伝達依頼）し，すぐさまモデラの保持するモデル $S_1 \rightsquigarrow S_2 \rightsquigarrow S_3$ を受け取る．モデルの伝達依頼はその後もモデラに保持され，コンポーネント側で変化があった場合は変化したモデルの伝達を受け取る．

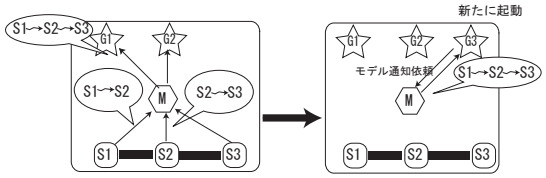

図 4　起動時の動作手順

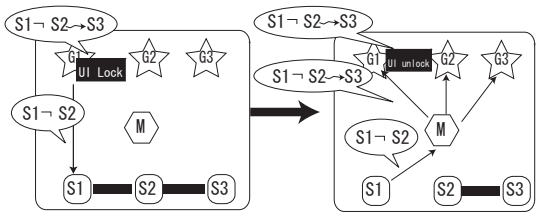

図 5　命令時の動作手順

3.5　uBlock Browser 命令時の一貫性

uBlock Browser 命令時の動作手順を図 5 に示す．uBlock Browser がユーザからの入力を受付けてシステムの再構築を行わせる際は，直接関係のあるコンポーネントにのみ伝達を行う．ユーザインタフェースの命令がモデラを経由しない理由は，モデラの負荷を下げると共に，モデラ依存度を下げるからである．同様に，コンポーネントは自分自身で動的に接続先を変更する可能性もある，この場合もモデラには結果のみを伝達する．

図 5 の場合は，$S_1 \rightsquigarrow S_2 \rightsquigarrow S_3$ と接続された分散アプリケーションを S_1 と S_2 の間を切断し，$S_1 \neg S_2 \rightsquigarrow S_3$ という新たなモデルを構成する様に命令した様子である．ここで $A \neg B$ は，A と B が切断されていることを表している．命令は，uBlock Browser の 1 つから発行され，すぐさま関係のあるコンポーネントに対してのみ命令を発行する．その際ユーザインタフェースは，ユーザから入力要求を一時的に遮断するため，ユーザインタフェース側で保持するモデルを変化させない措置をとる（図 5 中の UI Lock）．命令が発行され S_1 が S_2 との切り離しに成功すると，新たなモデルがモデラに通知され，$S_1 \neg S_2 \rightsquigarrow S_3$ というモデルに変化したことが uBlock Browser に通知される．この際に，uBlock Browser では，モデラから受け取ったモデルと，自らがロックしておいた命令直後のモデルを比較する．成功すれば UI Lock は解除する（図 5 中の UI unlock）．モデルを比較して一致しない場合，ユーザの入力を中止させ，ユーザにエラーを提示する．

現在 uBlocks での接続命令に関するメソッド一覧は以下である．

connectComponents(S_1,S_2)　通常のイベント配送を S_1 から S_2 に対して行うことを命令する．

disconnectComponents(S_1,S_2)　S_1 から S_2 へのイベント配送を停止する．

3.6　命令競合時の一貫性

複数 uBlock Browser がほぼ同時に，同一の分散アプリケーションに対して指示をした場合を説明する．図 6 では，ユーザインタフェース G_1 からは，$S_1 \neg S_2 \rightsquigarrow S_3$ という命令を発行した．次の瞬間ユーザインタフェース G_2 からは，$S_1 \rightsquigarrow S_2 \neg S_3$ という命令が発生していたとする．G_1 からは，$S_1 \neg S_2$ という命令が S_1 に発生し，G_2 からは，$S_2 \neg S_3$ という命令が S_2 に発生する．

S_1 は，S_1 と S_2 の切断を行い．その処理が完了したのを受けてモデラに $S_1 \neg S_2$ というモデルを送信する．モデラがその瞬間 $S_1 \neg S_2 \rightsquigarrow S_3$ を認識し，G_2 に対してモデルの変更を通達する．G_2 では，直前に命令したモデルと異なるため，非適合と判断し G_2 でユーザに変更を促すことができる．もしこの時点で，S_1 や S_2 が G_2 からの命令で変更されていれば，G_1 からの命令までに元に戻すか，現在の状態を保持するかをユーザに選択させることができる．

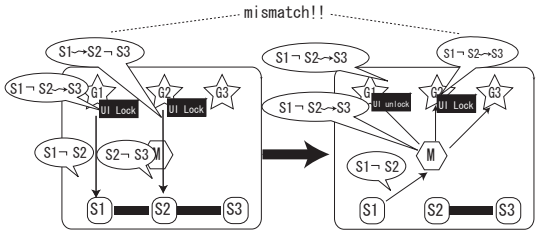

図 6　命令競合時の動作手順

本機構の競合時の動作は，命令競合時を完全な意味で回避するものではない．厳密に考えれば，G_1 か

らの S_1 への切断命令と，G_2 から S_2 への切断命令が同時に起こってしまった場合は，同時にモデラにアップデートが行われ G_1 にも G_2 にも命令時のロックしたモデルと異なったモデルが到着する可能性がある．しかし，この状態を避けるために命令中に全てのソフトウェアコンポーネントをロックしたり，全てのインタフェースをロックする手法は，結果としてコンポーネント間のトラフィックを増加させたり，ユーザビリティを下げてしまう場合がある．家庭内などの環境では，命令の頻度はさほど多いとは考えにくいため本論文では全てのコンポーネントへのロック機構は採用していない．

図 7 は，複数のインタフェースが一貫性をもった表示を行っている様子である．2 台のノート PC に合計 4 つの uBlock Browser を起動し，それぞれを独立して操作しても全ての uBlock Browser で同一の分散アプリケーションの構成を表示している．

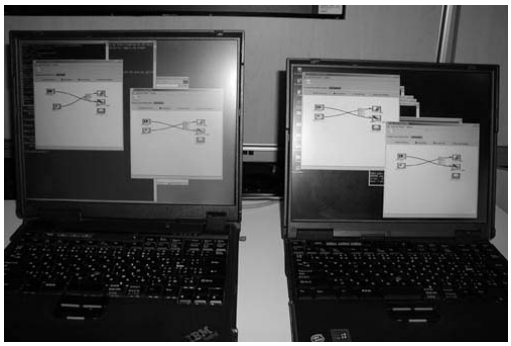

図 7　複数のインタフェースが分散アプリケーションの表示を同時に行っている様子

4　マルチモーダルユーザインタフェース

第 2.3.3 項で述べたように，ユーザ自身は，時に移動中であったり，異なる場所にいるなど，ユーザが分散アプリケーションを構築する場所やタイミングは選ばない．またユーザが使い勝手のよいインタフェースは，ユーザ自身の好みや習熟度による．そのためユーザ自身による分散アプリケーション構築を行わせるには，システム側でユーザの様々な環境に適応できる設定用のユーザインタフェースを提供する必要がある．uBlocks は，モデラからモデルを収集する役割と，各コンポーネント間の接続と切断の指示を出す役割を行うユーザインタフェース開発の共通モジュールとして **uBlock Base** を提供する．uBlock Base は，独立型モデリングトポロジによって実装されているためモデラに対して変更があったら通知するように依頼しておき，常に最新のコンポーネント間のモデルを受け取ることができる．

著者らは，図 1 で示した Java2D の uBlock Browser インタフェースに加え，3D のインタフェース，文字入力形式のインタフェース，i-mode などの WEB ブラウザからの操作などのマルチモーダルなインタフェースをサポートし，ユーザの利便性を上げることを目指した．図 8 で示す様に，シェルインタフェースや i-mode 用のインタフェースを uBlock Base を用いて実装した．uBlock Base は，利用するコンポーネントの持つ固有情報と，コンポーネント間の接続モデルを保持する．以下に固有情報一覧を示す．

- コンポーネント名
- ベンダー情報
- 住所
- 建物内位置情報
- GUI 上のアイコン情報
- コンポーネントの利用状況
- コンポーネントを制御するするダウンロード可能な GUI

4.1　Java2D Browser

図 1 にあるようにユーザは，マウスによってアイコンに抽象化されたコンポーネントを結びあわせる．個々のアイコンやコンポーネント情報の文字による説明は，コンポーネントが指定する URL からダウンロードして表示している．アイコンをクリックすると「接続」,「切断」の 2 つの動作を行える．

4.2　Java3D Browser

直感的にタッチパネルなどから指で操作を行うユーザインタフェースを開発した．図 10 は，部屋の様子を 3D としてモデリングし，機器の利用を定義した情報を,「紙を貼り付ける」というメタファを用いてユー

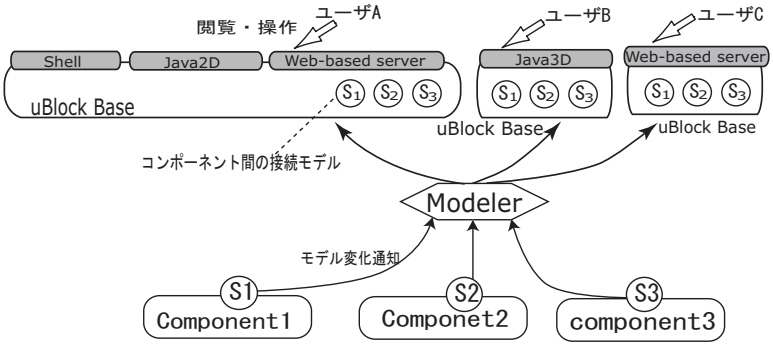

図 8　多様なユーザインタフェースの提供

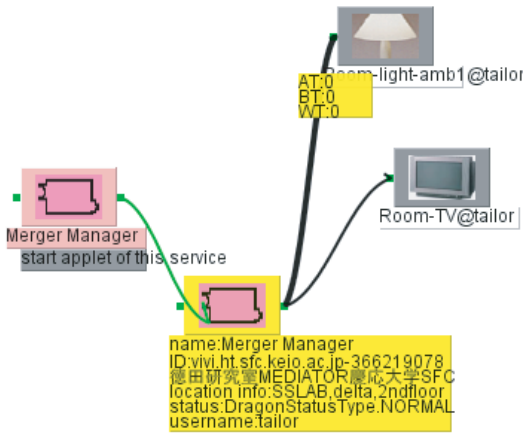

図 9　Java2D uBlock Browser の詳細表示

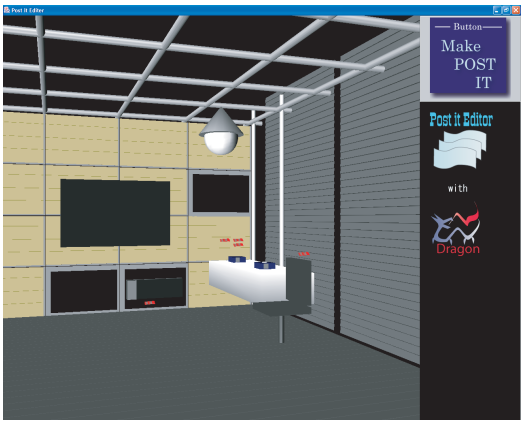

図 10　Java3D uBlock Browser

ザに理解し易いよう 3D にして提示した．

4.3　WEB-based uBlock Browser

ユビキタス環境においては，ユーザは常に移動する．その際も，ユーザが常に保持する携帯電話などの機器でも，分散アプリケーションを構築可能にする必要がある．しかし，携帯電話などの表示する資源の限られた端末には，全てのユビキタス環境の情報は通知できない．そのため必要な情報だけ選択して送る必要がある．図 11 のように，携帯電話からの操作を想定し，以下の 3 つのステップで分散アプリケーションが構築可能になっている．

1: コンポーネントの種類を選択させ，現在利用可能なコンポーネントの一覧を取得する．

2: 選択した 1 つのコンポーネントから接続可能なコンポーネント一覧を取得する．

3: 現在とは別の接続状態をクリックする．

4: 前の画面に戻り変更を確認することもできる．

本ユーザインタフェースは，バックエンドで HTTP サーバを uBlock Base と連動させて動作させている．図 12 のように他のユーザインタフェースと同時利用しても，前章の機構により一貫性が保たれている．

4.4　Shell 形式ユーザインタフェース

図 13 に示す文字型のインタフェースは，JXTA shell [14] からヒントを得て実装しているデバッグ向けのインタフェースである．現在は「list（情報表示）」,「connect（接続）」,「next（接続状態変化）」,「disconnect（切断）」,「kill（コンポーネント終了）」のコマンドから操作可能である．

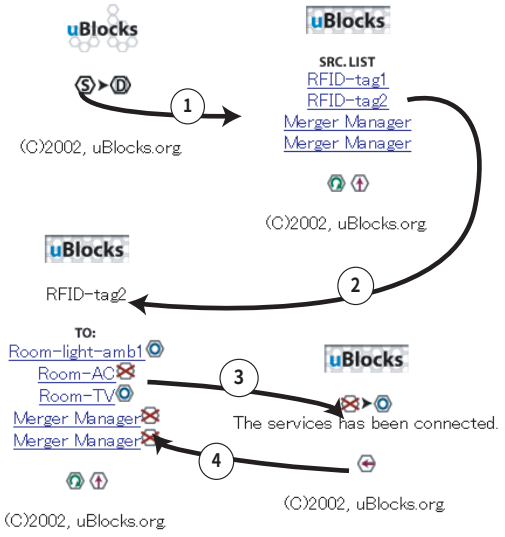

図 11 Web-based uBlock Browser の画面の遷移

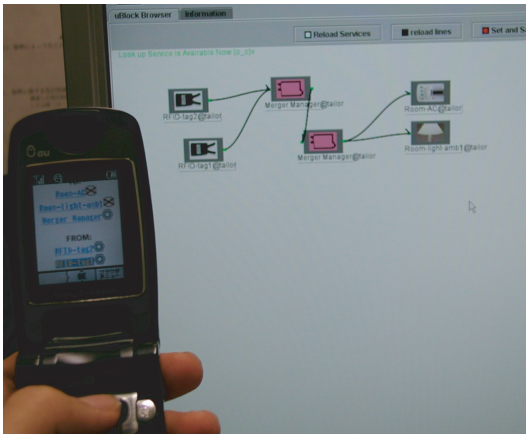

図 12 携帯電話から WEB-based uBlock Browser を操作し，背面の Java2D Browser と連動させる様子

5 性能測定と従来研究との比較

本章では，複数ユーザインタフェース間でのモデル伝達に関する基本的性能を測定し，安定してモデル伝達が行われることを測定する．また，従来研究と比較した本研究の優位点を述べる．

5.1 複数インタフェースへのモデル変化同期時間

ある 1 つのユーザインタフェースから，ネットワーク上に存在する 2 つのコンポーネントに対して，接続及び切断命令を下す．それらのコンポーネント間のモデル更新が全てのユーザインタフェースへ伝達終了までに要する時間を 30 回測定し，平均値，最大値，最小値をプロットした．ユーザインタフェースの総数は，1 個から 13 個まで変化させる．全てのインタフェースは，同一ホスト (Pentium IV 3GHz Windows XP) 上で動作させ，モデラは，別ホスト (Duron 800M FreeBSD 4.4) で動作している．図 14 に結果を示す．全てのユーザインタフェースへのモデルの通達終了するミリ時間 t は，一次式で近似すると $t = 13.477g + 75.207(g <= 13)$ であった．g は，ユーザインタフェースの数である．g が 13 以内という数字は，1 人に 1 台という家庭内などの携帯電話や PC の普及の現状から考えて現実的数字と言える．uBlocks のモデル変化の伝達時間 t は，$O(g)$ であり，予測可能である．このことから，uBlocks は安定していると言える．

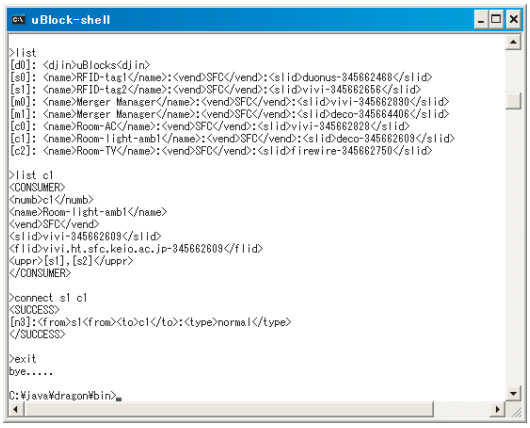

図 13 文字コマンドを用いた Shell uBlock Browser

5.2 インタフェース間のモデル通知時間の測定

モデルを変化させるユーザインタフェースとそのモデルの変化を同期するユーザインタフェースの対応の組み合わせをそれぞれ変え，モデル通知完了までの時間を各 50 回測定した．測定マシンの構成は，モデラおよびコンポーネントは同一マシン (Pentium IV 2.4G) で起動し，インタフェースは，同一マシン (Pentium IV 3G) で動作させている．最小時間，平均時間，最大時間を算出すると表 1 に示すようになっ

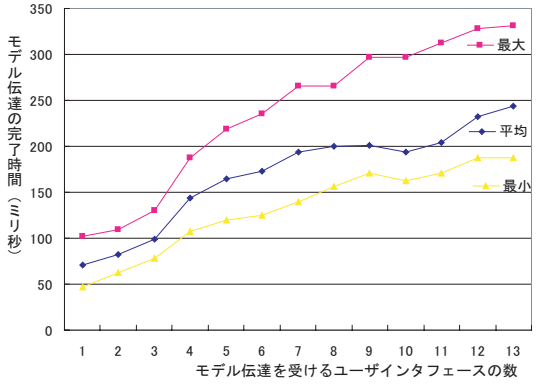

図 14　複数のユーザインタフェースへのモデル更新の伝達時間

た．Web-Based の場合は，サーバ側でモデルの更新が行われる時間である．

この測定から，Java2D の処理が他のインタフェースよりも劣っていることが分かるが，いずれの組み合わせも平均で 100ms 以内の実行速度を出せている．マルチモーダルなインタフェース間のモデル同期が安定した速度であると言える．

表 1　マルチモーダルなユーザインタフェース間のモデル同期時間 (ミリ秒)

モデル変更側	同期側	MIN	AVG	MAX
Web-based	Java2D	62	93.7	172
Web-based	shell	46	65.9	94
Java2D	Web-based	62	74.0	93
Java2D	shell	47	75.2	110

5.3　従来研究との比較

本節では，uBlocks と関連のある研究や製品との比較を行い，uBlocks の利点を確認する．

コンポーネント間のイベント接続を行うミドルウエアは，JMS[13]，CORBA Event Service[10]，JEcho[19]，JavaSpaces[15] など多く開発されてきた．特に JMS や CORBA Event Service は，同期通信，非同期通信，非接続状態のコンポーネントへの通知などが考慮された信頼性のあるミドルウェアである．しかし，それらの製品では，主に開発者向けのツールが多く，uBlocks のように分散アプリケーション構築をユーザが行う点が考慮されていない．

Jini の Distributed Event Model に対する設定用 GUI ツールを提供している Carp@[3] は，バイトコードジェネレータを用いて，動的に Jini のサービスをラップさせるオブジェクトを生成する．このラッパーオブジェクトにより，GUI から操作可能な Carp@Bean と呼ばれるサービスとなる．*port* と *connector* を用いて Jini の詳細なメカニズムを認識することなくコンポーネント間のつながりを即興的に開発者が設定できる．しかし既存の Jini のサービスをラッパするため，イベントの仲介を行うサービスは生成されず，やはり既知の Jini サービス間同士の通信になってしまう．そのため多様な分散アプリケーションを構築することは難しい．uBlocks では，入力と出力を同時に有するイベントを仲介するコンポーネントを提供し，内部を通るイベントオブジェクトを自由に変更可能であるので，Jini を基盤としないコンポーネントを含めた分散アプリケーションを構築でき，さらに本論文で述べた様に複数のインタフェースからの操作が可能である．また，uBlocks では，Carp@では提供されていない一対多の時間制約の設定が可能であり，配送の信頼性に厳密さを要するアプリケーションにも応用できる．

ICrafter[11] は，分散するコンポーネントから別々のユーザインターフェース定義 XML をダウンロードし，手元の PDA などでユーザインタフェースを動的に生成することが可能な研究である．uBlocks とは異なり，モデラは設置せずに全てコンポーネントの取捨選択と構成をユーザインタフェース上で行い，コンポーネントの一括で制御可能な GUI を構成する．ICrafter は，複雑な一括制御を行うリモコンなどは構成できるが，コンポーネント間の接続は考えられておらず，各コンポーネントは常にユーザインタフェースの状態を把握しなければならない．また各ユーザインタフェースは，自分に関係するコンポーネントの状態を常に監視しなければならず．組込機器や PDA に適したモデルとはならない．uBlocks は，コンポーネントの状態の監視や，ユーザインタフェースへのモデルの生成は全てモデラに一括で任せてしまうため，

リソースが限られるユーザインタフェースやコンポーネントにとっては処理を行う内容が少ない．

SIENA [2] は，コンポーネントをインターネットスケールで構造化できる．SIENA は厳密に型付けされたイベントと pattern と呼ばれるイベントのマッチングルールにより，複数の仲介者に渡ってイベントのルーティングを動的に行うことが可能である．pattern は，SIENA のネットワーク内を効率よく配布され，効率がよい．しかし，複数のユーザインタフェースを用意して全体を把握することは逆に困難になるため，本論文の主旨としたユーザによる構成には適していない．また，pattern には，株式購入などの厳格な型のある情報は記述することができるが，ユビキタス環境の様に多様なコンポーネントの存在には対応できない．

6 uBlocks を用いたアプリケーション例

分散するコンポーネント間の接続に関するアプリケーションは，著者らが実証実験を行ったものから適宜ライブラリとして提供している．本章では，uBlocks を用いて実現されているインターカム画像転送アプリケーション及び一括情報家電制御アプリケーションの 2 つについて述べ，uBlocks が，ユーザ自身によってアプリケーションが構築可能で，ある程度の複雑なアプリケーションも構築可能であることを示す．

6.1 インターカム画像転送アプリケーション

本アプリケーションは，ドアなどに設置しているカメラでキャプチャした画像を別の複数のコンポーネントに転送可能にするものである．はじめに，インターカムの画像をキャプチャ可能なソフトウェアコンポーネントとディスプレイに画像を閲覧させるコンポーネント，携帯電話のメールアドレスにメールを転送するコンポーネントの 3 つを起動させる．まず，ユーザが uBlock Browser を用いて，頻繁に利用する PC に転送するように接続しておく．この状態で，インターカムのボタンがおされキャプチャが行われるごとに，設定にしたがって PC 上のコンポーネントに対して，インターカムの画像を表示させるイベントが転送される．もし，ユーザが外出する際などは uBlock Browser によって携帯電話に転送するように接続を切り替えておく．この状態では，インターカムによってキャプチャされた画像は，携帯電話向けにサイズを縮小変換し，その変換後の画像の URL をメールによって携帯電話に転送する．ユーザは，到着したメールの URL を獲得すれば，図 15 の様に来訪者の画像を閲覧できる．

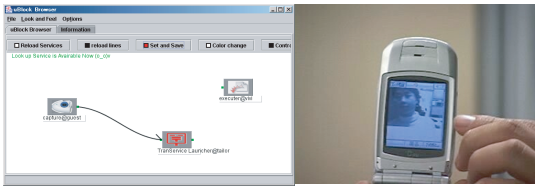

図 15 ユーザの接続命令にしたがって，キャプチャされた画像が携帯電話に配信される様子

6.2 一括情報家電制御アプリケーション

本アプリケーションは，1 度の命令で複数の情報家電を制御するアプリケーションである．携帯電話のブラウザからの 1 回の命令でユーザによって接続された情報家電機器に対して命令を発行できる．図 16 に示すように，ネットワークに接続され制御可能な情報家電機器には，赤外線リモコン経由で制御可能な MD プレイヤ及び，室内ライト，電源の ON/OFF 可能な制御機器がある．ユーザは，モードコンポーネントと呼ばれるコンポーネントと制御したい機器を接続する．この状態で携帯電話からユーザが望むモードコンポーネントを選択すると，該当するモードコンポーネントから接続されているコンポーネントへ操作イベントが命令され，一括して実行される．図 16 では，起床モードと就寝モードを設けて，それぞれに ON/OFF 操作したいコンポーネントを接続しておくだけである．

6.3 構成可能なアプリケーションに関する議論

ユーザには，コンポーネント間の切断/接続という操作に限ったことで，コンポーネント内部のイベントの発生条件や閾値，処理内容などについてユーザの手を加えることは不可能である．そのため，第 6.1 節に

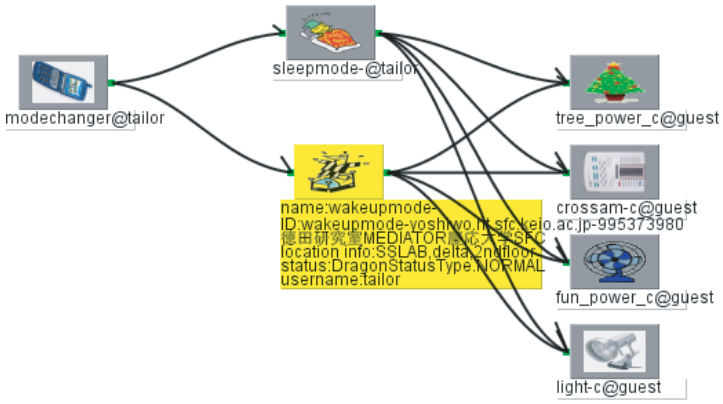

図 16　一括でユビキタスな部屋の情報家電機器を変化させた接続の様子

あるアプリケーションの例であるインターカムの画像のキャプチャコンポーネントは，ボタンが押されるごとにキャプチャされたというイベントを発生させてしまう.「2 回連続でボタンが押されても画像を一度しかキャプチャしない」という設定をユーザがしたければ，以下の様な準備を開発者がしておく必要がある．あらかじめキャプチャコンポーネントの開発者がユーザ用にダウンロード可能な（第 4 章の uBlock Base で述べた）「コンポーネントを制御するダウンロード可能な GUI 」を準備しておく．こういった，コンポーネント内部の振る舞いをユーザがどこまで設定可能にするかは，使い易さとのトレードオフとなる．

第 6.2 節にあるアプリケーションでは，携帯電話からのイベントがモードコンポーネントを介して，各情報家電の制御を行えるコンポーネントに伝達される．各コンポーネントでは，そのイベントオブジェクトの解釈を独自に行って処理する．そのためライトを 10 段階で調光できる新型ライトコンポーネントを導入しても携帯電話からの命令受付を行うコンポーネントにその新型のライトコントロールイベントを実装しなければ，限られた機能に限定されてしまう．新しいコンポーネントの登場後は，(1) コンポーネント側で古くからある命令をオーバライドして新機能を動作させる，(2) 仲介コンポーネントを設置する，(3) イベントを発行するコンポーネントを最新にアップデートするなど選択肢がある．

7　おわりに

ユビキタスコンピューティングの分野において，多くの従来型のシステムでは，ソフトウェアコンポーネント構成や接続をあらかじめ固定した分散アプリケーションが構築されてしまっている．ユーザに対して，分散アプリケーションの押し付けをする可能性がある既存のシステムは，必ずしもユビキタスコンピューティングの恩恵をユーザに与えているとは言えない．

本論文において著者らは，遍在するソフトウェアコンポーネントの組み合わせを，ユーザ自身により，視覚的に変更可能にする手法を検討し，複数個のインタフェースから一貫性のある分散アプリケーションを構築する際の手法を示した．uBlocks がマルチモーダルなユーザインタフェースをサポートできることを個々に説明しながら，実装したブラウザ群を示した．評価においては，複数個のインタフェースへモデルが伝達するまでにかかる時間と，マルチモーダルなインタフェース間での同期の時間を測定し，実効性と安定性を検証した．

uBlocks は，文献 [7] から最新のバイナリ，ソースコードがダウンロード可能となっている．今後は，VNA，UPnP，JXTA などで実装されたコンポーネントも透過的に接続可能にする仕組みについて検討する．また，本ソフトウェアの応用範囲をビジネス分野にも広げる為，コンポーネント間の通信には，SOAP [1]，JMS などの多様なミドルウエアとの連携を検討

している．またインタフェースに関しては，特に携帯電話などの表現リソースが限られた環境下においての，絞り込みのコンポーネント検索と表示について研究を進め，音声インタフェース，触覚インタフェースへの応用も将来課題とする．

謝辞

東京大学大学院　情報理工学系研究科 萩谷昌己教授には，情報処理振興事業協会平成 14 年度未踏ソフトウェア創造事業において，本ソフトウェア開発のプロジェクトマネージャをして頂き，貴重なご助言をくださいました．慶應大学政策・メディア研究科修士課程古市悠氏には，Jav3D uBlock Browser の開発を手伝っていただきました．ソフトウェア科学会論文誌査読委員の方々には，的確な参照事項と熱心なコメントを頂き，論文の大幅な改善を行うことができました．ここに，みなさまに深く感謝いたします．

参考文献

[1] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D.: .

[2] Carzaniga, A., D.S.Rosenblum, and A.L.Wolf: Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service, *Nineteenth ACM Symposium on Principles of Distributed Computing*, July 2000.

[3] Fahrmair, M., Salzmann, C., and Schoenmakers, M.: A Reflection Based Tool for Observing Jini Services, *Reflection and Software Engineering*, June 2000, pp. 209–227.

[4] Frank Leymann, IBM Software Group: Web Services Flow Language 1.0, may 2001.

[5] Iwai, M., Nakazawa, J., and Tokuda, H.: Dragon: Soft Real-Time Event Delivering Architecture for Networked Sensors and Appliances, *The 7th International Conference on Real-Time Computing System and Applications*, December 2000, pp. 425–432.

[6] Iwai, M., Nakazawa, J., and Tokuda, H.: Flexible Distributed Event-Driven Programming Framework for Networked Appliances and Sensors, *The 3rd International Symposium on Distributed Objects and Applications Short Papers Proceedings*, September 2001, pp. 61–68.

[7] Iwai, M., Yura, J., and Mochizuki, M.: http://www.uBlocks.org.

[8] Masayuki, I., Jin, N., and Hideyuki, T.: uBlocks: Enabling User-side Composition of Distributed Ubiquitous Application, **情報処理学会情報家電コンピューティング研究会（*IAC-4*）**, Vol. 4(2002), pp. 29–36.

[9] Nakazawa, J., Tobe, Y., and Tokuda, H.: On Dynamic Service Integration in VNA Architecture, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 7,No. E84-A(2001), pp. 1610–1623.

[10] Object Management Group: The Common Object Request Broker Architecture and Specification 2.2ed CORBA Event Service, February 1998.

[11] Ponnekanti, S. R., Lee, B., Fox, A., Hanrahan, P., and Winograd, T.: ICrafter: A Service Framework for Ubiquitous Computing Environments, *UBICOMP 2001, LNCS 2201*, October 2001.

[12] 岩井将行, 中澤仁, 西尾信彦, and 徳田英幸: 分散コンポーネントによる即興的アプリケーション構成機構の実現, **情報処理学会論文誌**, Vol. 43,No. 6(2002), pp. 1664–1676.

[13] Sun Microsystems, Inc.: Java Message Service Documentation Version 1.0.2b.

[14] Sun Microsystems, Inc.: JXTA v1.0 Protocols Specification. http://www.jxta.org.

[15] Sun Microsystems Inc.: JavaSpaces Service Specification, version 1.1, October 2000. http://www.sun.com/jini/specs/js1_1.pdf.

[16] Sun Microsystems Inc.: *Jini Architecture Specification*, October 2000.

[17] Sun Microsystems, Inc.: Enterprise Java Beans, 2001. http://java.sun.com/products/ejb/.

[18] Tajika, Y., Ajitomi, D., Minoh, M., and Kamae, T.: Service Structure and its Description in Peer-to-peer Network Appliance Architecture AMIDEN, *IWNA4*, January 2002.

[19] Zhou, D., Schwan, K., Eisenhauer, G., and Chen, Y.: JECho–Interactive High Performance Computing with Java Event Channels, *International Parallel and Distributed Processing Symposium*, April 2001.