Vol.2017-MBL-82 No.6
Vol.2017-UBI-53 No.6
2017/3/9

複数ディスプレイをシームレスに統合するウェブブラウザベース分散ウィンドウシステム

板倉 宏太[1,a)]　可児 潤也[1]　由良 淳一[1]　武 理一郎[1]

概要：本論文では，議論や協調作業における資料の共有・操作をするための，複数ディスプレイをシームレスに統合するウェブブラウザベースの分散ウィンドウシステムを提案する．複数ディスプレイを用いた従来のシステムは，各ディスプレイに接続された機器に専用のアプリケーションを導入する必要があり，プラットフォームが混在している場合はアプリケーションの開発や導入・更新が困難だった．また，複数の機器が通信して連携しているため，操作時の応答性低下が問題であった．提案するウィンドウシステムはウェブブラウザ上に構築され，ウェブコンテンツを複数ディスプレイにまたがって表示・操作可能にする．ウェブブラウザ上でシステムを構築することで，さまざまなプラットフォームで共通のコンテンツを動作させることが可能になる．また，各機器で分散してウィンドウを管理することで，通信による応答性低下を防ぐことができる．提案システムの実装を行い，評価実験によって本システムの有用性を確認した．

A Distributed Window System based on Web Browser for Connecting Displays Seamlessly

Kota Itakura[1,a)]　Junya Kani[1]　Junichi Yura[1]　Riichiro Take[1]

1. はじめに

人々が一か所に集まり議論や協調作業を行う場合，参加者同士で共通認識となる情報を共有することが重要となる．情報共有の手段としては大型のディスプレイやプロジェクタが用いられてきたが，近年では，電子黒板やテーブル型ディスプレイを用いて，資料の操作やメモの作成など，閲覧だけでなく情報とのインタラクションを可能にすることで，コミュニケーションをより円滑化する方法が提案されている [1], [2].

また，一つのディスプレイでなく，複数のディスプレイを用いる方法も提案されている．複数のディスプレイを用いることで，表示面を拡張して情報を大きく表示できるだけでなく，机で作業した内容を壁に転送して閲覧するといったように，さまざまな表示面を用途に応じて使い分けることが可能となる [3], [4].

以上のように，机や壁面などのあらゆる面が柔軟に連携できるようになってきており，将来的には，もはや参加者は機器を特に意識することなく，情報の閲覧・移動・操作を直感的に行えるようになっていくと考えられる．このような，入力可能な多数のディスプレイを柔軟に組み合わせて，あたかも部屋全体を一つのインタラクション空間として扱う仕組みを，本研究所では空間 UI と定義している [5]. **図 1** に空間 UI のイメージを示す.

空間 UI を実現するには，ディスプレイやプロジェクタに接続された機器（以下，ディスプレイ機器）をネットワーク上で連携させ，アプリケーションウィンドウの入出力を管理するウィンドウシステムが必要となる．しかし，空間 UI におけるウィンドウシステムの実現にあたっては以下の問題点が存在する.

問題点 1：プラットフォームの混在による開発困難性

多種多様な機器を連携する場合，各機器のプラットフォームが統一された環境であるとは限らない．そのため，ウィンドウシステムやその上で動くアプリケーションを各プラットフォームに向けて開発する必要があり，開発コスト

[1] 株式会社富士通研究所
[a)] itakura.kota@jp.fujitsu.com

図 1　空間 UI のイメージ

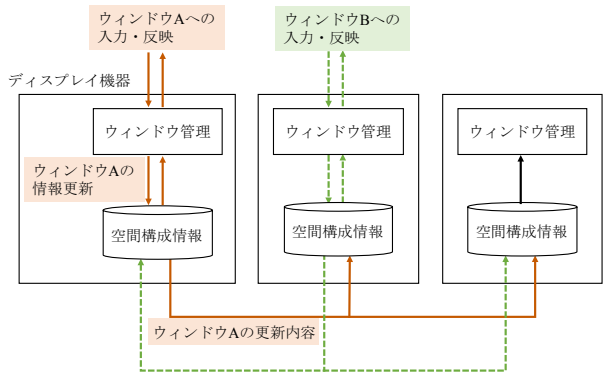

図 2　分散共有型ウィンドウ管理

が非常に大きい．また，ソフトウェアの導入や更新を全ての機器で行う必要があり，導入や運用にかかる手間も大きい．

問題点 2：高い応答性要求

使い勝手の良いシステムを実現するには，画面への入力からその反映までの時間を短くすること，すなわち応答性を高めることが不可欠である．しかし，ディスプレイ一台のみで利用する場合と異なり，複数のディスプレイ機器をネットワーク上で連携させる場合，通信に時間がかかるため，応答性を大きく損ねる．

以上の問題点を解決して空間 UI を実現するため，本論文ではウェブブラウザベースの分散ウィンドウシステムを提案する．提案するシステムは，動作基盤にウェブブラウザを用いることで，プラットフォームに依存しないウィンドウシステムとアプリケーションの動作を実現する．また，ウィンドウを一か所で管理するのでなく，各ウェブブラウザで分散管理を行うことで，通信による遅延の少ないスムーズな動作を実現する．

本論文の構成を以下に述べる．2 章では本システムについて，方針やシステム構成の詳細を述べる．3 章では本システムの実験，評価，および考察を行う．4 章では本システムと関連研究との比較を行う．5 章では本研究のまとめを述べる．

2. 設計

本章では，提案するブラウザベース分散ウィンドウシステムについて，まず 1 章で述べた課題の解決方針について検討し，次に提案システムのアーキテクチャと各機能の詳細を述べる．

2.1 方針

2.1.1 ウェブブラウザ上でのウィンドウシステム構築

マルチプラットフォームでの動作を実現するため，本システムは実行基盤としてウェブブラウザを利用する．

ウェブブラウザはさまざまなオペレーティングシステム上で提供されており，さまざまな環境で同一のウェブアプリケーションを動作させることができる．近年では，HTML5[6] や ECMAScript 2015[7] など，ウェブブラウザ仕様の提案や各ウェブブラウザでの仕様実装が進んでおり，高度な機能や表現力をさまざまなプラットフォームで共通に利用できるようになっている．

本システムではこのようなウェブブラウザのマルチプラットフォーム性に注目し，ウィンドウシステム自体をウェブアプリケーションとして作成することによって，ウィンドウシステムとその上で動くアプリケーションをマルチプラットフォームで動作可能にする．

ウェブブラウザを用いる利点としては他にも，システムの導入や更新にかかる負荷を大幅に削減できる点が挙げられる．ウェブブラウザは実行時に最新のアプリケーションを取得して実行するため，導入の際にウィンドウシステムのインストール作業を各ディスプレイ機器で行う必要はなく，システム更新の際にもウェブサーバ上のアプリケーションを更新するだけでよい．

2.1.2 分散共有型ウィンドウ管理

高い応答性を確保するために，本ウィンドウシステムは各ディスプレイ機器で分散共有型のウィンドウ管理を行う．

複数ディスプレイ機器でウィンドウシステムを構成する場合，通信時間による入力遅延が問題となる．全てのウィンドウをサーバで一元管理する場合，どのウィンドウを操作する際も毎回サーバに問い合わせる必要があり，必ず通信遅延が発生する．そこで本システムでは，なるべく通信せず入力を反映できるよう，全てのウィンドウを一か所で管理するのでなく，各ウィンドウをそれぞれ入力元のディスプレイ機器で管理するという分散ウィンドウ管理方式を採用した．**図 2** に分散共有型ウィンドウ管理の概要を示す．

分散共有型ウィンドウ管理では，ディスプレイの接続構成やウィンドウの位置など，ウィンドウ管理に必要となる全ての配置の構成情報（以下，空間構成情報）を各ディスプレイ機器が保持する．これにより，各ディスプレイ機器で通信せずともウィンドウが管理できるため，ウィンドウ移動などの操作入力を即座に反映することが可能である．

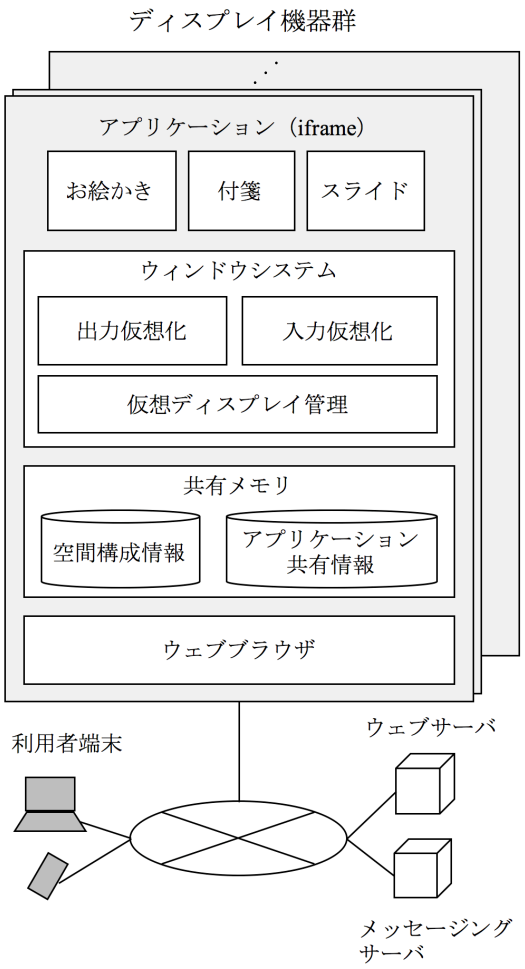

図 3　分散ウィンドウシステムのアーキテクチャ

空間構成情報の変更内容はその後他のディスプレイ機器に送信され，変更が部屋全体に反映される．

2.2　アーキテクチャ

2.1 節の方針を踏まえ，ブラウザベースの分散ウィンドウシステムを設計した．本システムのアーキテクチャを**図 3** に示す．

本システムはサーバとディスプレイ機器群，そして利用者端末から構成される．本システムではウィンドウ管理をすべてディスプレイ機器が行うため，サーバ側にはアプリケーションを配信するウェブサーバ機能と，ディスプレイ機器間で相互に通信するためのメッセージングサーバ機能のみが必要となる．

一方で，それぞれのディスプレイ機器は，複数ディスプレイにまたがるウィンドウの入出力管理を実現する必要がある．本システムでは，ディスプレイ機器をまたがるウィンドウの位置を規定するため，ディスプレイ機器群を組み合わせた大きいディスプレイ空間として，仮想ディスプレイ座標空間を規定した．仮想ディスプレイとディスプレイ機器との関係を**図 4** に示す．

仮想ディスプレイを用いて，ディスプレイ機器でウィンドウ管理を行うにあたっては，1) 空間構成情報の共有機能，2) 空間における自分の表示位置を計算してウェブブラウザへ描画する出力仮想化機能，3) ウェブブラウザへの入力情報を他のディスプレイ機器と交換し，ディスプレイ横断的な入力やマルチタッチを実現する入力仮想化機能が必要となる．また，ウェブアプリケーションを複数のディスプレイや利用者端末間で同期して表示するための，4) アプリケーション同期機能も必要である．以下，それぞれについて詳細を述べる．

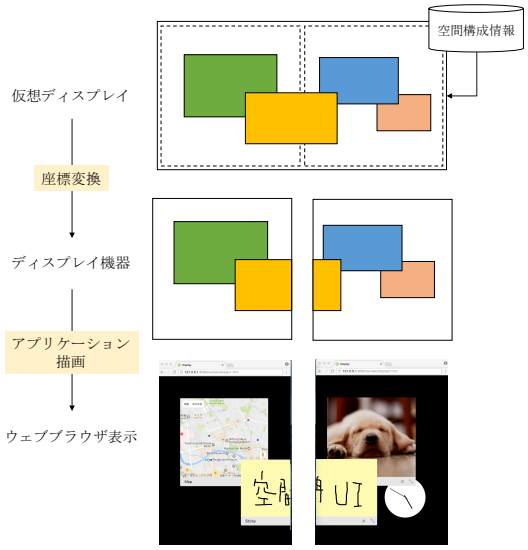

図 4　仮想ディスプレイによるまたがったアプリケーション配置

2.2.1　空間構成情報の共有

空間構成情報に含まれる情報を**表 1** に示す．空間構成情報は全てのディスプレイ機器が同じ情報を保持する．空間構成情報への変更が指示されたときは，ディスプレイ機器が内部に保持する空間構成情報はただちに変更され，また他のディスプレイ機器に変更内容が送信されることで同期を行う．

本システムでは，空間構成情報の共有を実現するため，本研究所が開発しているデータ共有ライブラリ [8] を採用した．空間構成情報が変更されるたびに他のディスプレイ機器への送信を行うと，メッセージの量が膨大となり処理負荷・通信負荷が増大するが，このデータ共有ライブラリは処理負荷や通信負荷に応じてメッセージの送受信量を最適化することで，負荷を抑えつつデータ共有を行う．

なお，空間構成情報には，仮想ディスプレイやディスプレイ機器，ウィンドウ情報の他に，ディスプレイ接続情報が含まれる．ディスプレイ接続情報は，机や壁など，空間的には連続していないディスプレイ面を接続するための設定であり，複数の仮想ディスプレイの辺同士の接続関係を規定する．ウィンドウが接続辺を超えた際は，アプリケーションは対応する接続辺に転送される．

2.2.2　出力の仮想化

各ディスプレイ機器は，空間構成情報に含まれる仮想

表 1　空間構成情報

種別	概要	データ内容
仮想ディスプレイ	複数のディスプレイを一つの面として表示するための，仮想的なディスプレイ設定	ID, サイズ
ディスプレイ機器	各ディスプレイ機器がそれぞれ仮想ディスプレイのどの部分を表示するか	ID, 属する仮想ディスプレイ ID, 位置, サイズ, 回転量
ディスプレイ接続	机と壁の接続などの，仮想ディスプレイ同士の接続関係	接続関係のディスプレイ ID 組, 接続されている辺の位置
ウィンドウ	ディスプレイ上で描画されるアプリケーション窓	ID, 仮想ディスプレイ ID, 位置, サイズ, 回転量, 表示アプリケーションの URL, z 方向位置指定（topmost 等）

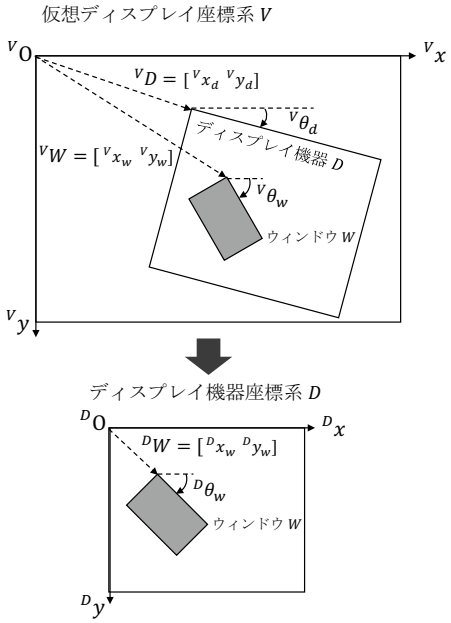

図 5　仮想ディスプレイ座標からの座標変換

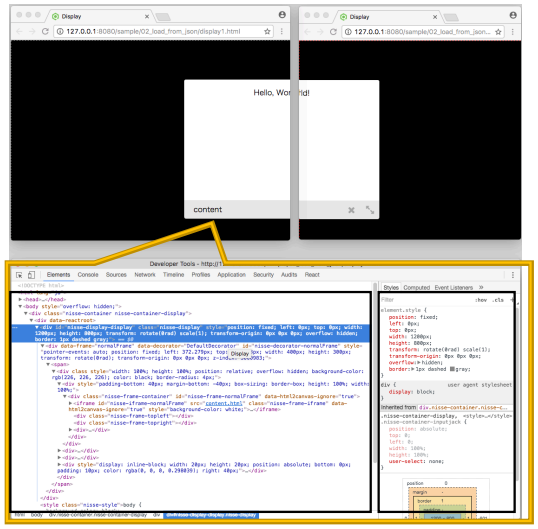

図 6　文書構造とスタイルシート

ディスプレイ情報を用いて，各々が担当するディスプレイ範囲へと座標変換を行ってアプリケーションを描画する．

例として，図 5 のようにディスプレイ機器とウィンドウが仮想ディスプレイ上に配置されている場合を考える．仮想ディスプレイ座標におけるディスプレイ機器領域の左上位置を ${}^{V}D = [\ {}^{V}x_d \quad {}^{V}y_d\ ]$，ディスプレイ機器領域の回転量を ${}^{V}\theta_d$ とするとき，仮想ディスプレイ座標における左上位置 ${}^{V}W = [\ {}^{V}x_w \quad {}^{V}y_w\ ]$，回転量 ${}^{V}\theta_w$ のウィンドウについて，ディスプレイ機器の座標系に変換したときの左上位置 ${}^{D}W$ および回転量 ${}^{D}\theta_w$ はそれぞれ以下の式により求められる．

$$\begin{cases} {}^{D}W = ({}^{V}W - {}^{V}D) \begin{bmatrix} \cos{}^{V}\theta_d & -\sin{}^{V}\theta_d \\ \sin{}^{V}\theta_d & \cos{}^{V}\theta_d \end{bmatrix} \\ {}^{D}\theta_w = {}^{V}\theta_w - {}^{V}\theta_d \end{cases} \quad (1)$$

多くの場合 ${}^{V}\theta_d = 0$ であり，さらに容易に算出できる．

$$\begin{cases} {}^{D}W = {}^{V}W - {}^{V}D \\ {}^{D}\theta_w = {}^{V}\theta_w \end{cases} \quad ({}^{V}\theta_d = 0 \text{ のとき}) \quad (2)$$

次に，算出した座標を元にアプリケーションの描画を行う．ウェブブラウザの文書構造を編集する API である DOM(Document Object Model)[9] を用いて，外部リソースを表示する iframe 要素を挿入することで，アプリケーションを画面に表示する．さらに，表示要素の見た目を制御するスタイルシートを設定することで，計算した表示位置通りにアプリケーションを表示する．表示された画面，およびその文書構造とスタイルシートを図 6 に示す．このように，各ディスプレイで滑らかにつながるように iframe の位置を調整することにより，複数ディスプレイにまたがるアプリケーション表示をウェブブラウザ上で実現できる．

なお，本システムでは，他のウィンドウシステムと同様，ウィンドウの移動やリサイズなどの機能を標準的に提供している．これにより，どのようなアプリケーションであっても，アプリケーション側に特別な設定なく，ウィンドウ移動やリサイズを行える．

2.2.3　入力の仮想化

出力の仮想化によって，アプリケーションを複数のディスプレイにまたがって表示することが可能である．しかし，電子ペンやタッチパネルなどを用いて画面への入力操作については，入力イベントがタッチされたディスプレイ機器にのみ通知されるため，複数ディスプレイをまたがるようなドラッグ操作やマルチタッチ操作は，それぞれ二つのド

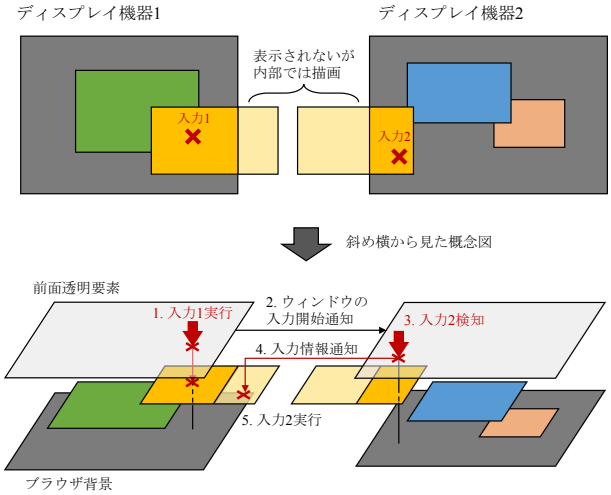

図 7 入力仮想化の概要

ラッグ操作や二つのシングルタッチ操作であると誤って判定される．そこで本システムでは，複数ディスプレイをまたがるような入力を実現するため，入力の仮想化を行う．

入力仮想化の概要を**図 7** に示す．本システムではまず，入力操作を直接アプリケーションに伝える代わりに，前面に配置した透明要素で入力イベントを全て受け止める．次に，入力イベントの対象であるウィンドウを計算し，そのウィンドウが別のディスプレイ機器で入力中であれば，入力イベントの内容を入力中のディスプレイ機器に転送する．最後に，アプリケーションに対して入力イベントを擬似的に再現することで入力を実行する．このとき，画面接続部付近で入力終了後すぐに隣接するディスプレイで入力が開始した場合は，一連のドラッグ入力とみなし入力を継続する．以上により，アプリケーションはディスプレイをまたがるドラッグやマルチタッチ入力を認識でき，対応する処理を実行することが可能となる．

ウェブブラウザにおける入力イベントの擬似的な再現を行う API は，Touch Events - Level 2[10] として仕様が提案されており，Google Chrome[11] などのウェブブラウザで既に実装が行われている．本システムではこの仕様を用いて，iframe 内のアプリケーションに入力情報を渡し，アプリケーション内で入力イベントを擬似発火することで入力操作を再現する．

なお，ウェブアプリケーション内で入力イベントを疑似発火するために，あらかじめアプリケーション側に本システム向けのライブラリを読み込む必要がある．このライブラリは入力情報を受け取って入力イベントの疑似発火を行うほか，次節で述べるアプリケーション同期用の API を提供する．

2.2.4 アプリケーション同期

入力仮想化，出力仮想化によって，ウェブアプリケーションを複数ディスプレイに渡って表示・操作できるようになる．しかし，アプリケーションは各ディスプレイ機器上でそれぞれ独立して動作しているため，あるディスプレイ上でアプリケーションが入力に応じ表示内容を変化させたとしても，入力されなかった他のディスプレイでは表示内容は変化しない．そのため，アプリケーションの表示を同期するための仕組みが必要となる．アプリケーション同期の仕組みを提供することで，ディスプレイ機器間で表示が同期できるようになるだけでなく，利用者のスマートフォンや PC で同じアプリケーションを開いて同期できるようになる．

本システムでは，表示するアプリケーションの性質に応じて，キーバリューストア，文書構造転送，リモートデスクトップの三種類のアプリケーション同期方式を提供する．アプリケーション開発者はいずれかの方式を選択し，対応するライブラリを導入することでアプリケーション画面の同期を実現する．それぞれについて以下に説明する．

キーバリューストア

キーバリューストア方式では，アプリケーション開発者に対し，共有情報の設定，取得，変更検知を行う API を提供する．この API を用いると，文字列をキーとして任意の情報を設定でき，設定された情報は他のディスプレイ機器上の同アプリケーションに通知されるため，開発者はこの API を利用してアプリケーションの同期を実現できる．情報の同期には空間構成情報と同じく分散共有メモリを用いる．また，同じアプリケーションを利用者のウェブブラウザで開き，分散共有メモリ上の同じ情報を参照することで，利用者端末と空間側の画面同期を実現できる．

文書構造転送

ウェブアプリケーションにおいては，画面表示に必要となる現在の文書構造やスタイルシート，フォーム入力内容といった情報をウェブブラウザ上のプログラムから抽出することできる．そこで，文書構造転送方式では，利用者端末で開いたウェブアプリケーションの表示内容を抽出し，各ディスプレイ機器に転送することで，ディスプレイ機器で同期した表示を実現する．**図 8** に本方式の概要図を示す．ディスプレイ機器上に入力があった場合，入力情報は一度利用者端末に転送され，入力の疑似発火を行ったのち，変更された表示情報を再度ディスプレイ機器に送信することで入力反映を行う．

リモートデスクトップ

リモートデスクトップ方式では，VNC[12] などの既存のリモートデスクトップ方式を利用して，利用者端末の画面内容を各ディスプレイ機器で表示する．ウェブアプリケーションとして作成したリモートデスクトップクライアントをウィンドウとして開き，利用者端末と接続することにより画面同期を実現する．

アプリケーション同期方式の比較

本システムが提供する三つのアプリケーション同期方式の比較を**表 2** にまとめる．

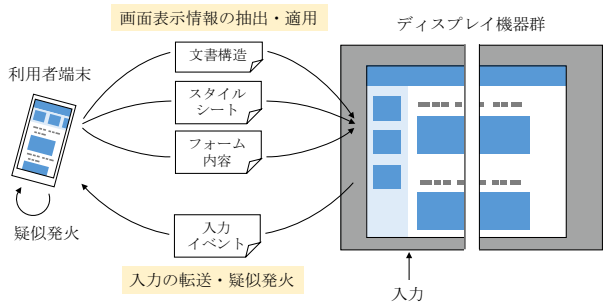

図 8　文書構造転送方式の概要

表 2　アプリケーション同期方式比較

	応答性	アプリの対応コスト	メディアコンテンツの同期	利用者端末の要否	利用者端末の導入コスト
キーバリューストア	○	×（専用構成）	○	○（不要）	○
文書構造転送	△	○（ウェブアプリ）	△（一部非対応）	×	○
リモートデスクトップ	△	◎（任意アプリ）	○	×	×（要インストール）

応答性の点では，ディスプレイ機器上で即座に応答できることや同期内容を最適化できることから，キーバリューストア方式が最も優れる．文書構造転送方式やリモートデスクトップ方式は，一度利用者の端末に入力情報を送信し，入力を実行したあとディスプレイ機器に表示内容を転送する必要があるため，応答性はキーバリューストア方式に劣る．

一方で，本システムへのアプリケーション対応コストについては，キーバリューストア方式は API を使う形で専用にアプリケーションを作る必要があるため，他の方式に比べ対応に手間がかかる．文書構造転送方式はウェブアプリケーションの内容によらず転送が行えるので，ライブラリを読み込むだけで対応できるものの，動画埋め込みなどを行う object タグや embed タグなど，プログラム上から抜き出せない一部の要素には対応できない．リモートデスクトップ方式では，アプリケーション側に一切対応の必要がないだけでなく，ウェブアプリケーションに限らず任意のアプリケーションの画面を転送することが可能である．

また，キーバリューストア方式以外は利用者端末に関して制約があり，転送元となるアプリケーションを利用者端末で開いている必要がある．リモートデスクトップ方式ではさらに，あらかじめ利用者端末に画面転送用のソフトウェアをインストールしておかなければならない．

以上のように，いずれの方式もそれぞれ長所や短所があり，アプリケーションによって適する方式が異なる．そのため，本システムはいずれの方式も提供し，開発者が任意で選択できるようにしている．

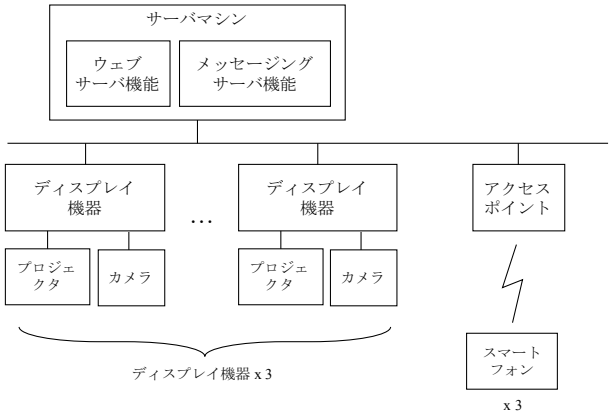

図 9　システム構成図

表 3　機器詳細

種別	プラットフォーム	実行環境
サーバ	Windows 8.1 Pro Intel Core i7-4765T 2.00GHz 8GB RAM	Node.js 6.2.2
ディスプレイ機器	Window 8.1 Pro Intel Core i7-4785T 2.20GHz 8GB RAM	Chromium 50
スマートフォン	Arrows F-02G Android 5.0.2	Android-WebView 43

3. 実装と評価

提案システムの有効性を確認するため，システムの実装を行い，アプリケーションの開発容易性や実行速度に関して評価実験を行った．実装の詳細，実験内容および考察を述べる．

3.1 実装

実装したシステムの構成を図 **9**，利用した機器の詳細を表 **3** に示す．本実装では画面出力にプロジェクタを用い，入力には赤外線カメラと電子ペンによる入力装置を用いた．また，各ディスプレイ機器で実行するアプリケーションとして，手書きで絵や文字を描いたり，付箋紙を貼り付けたりといった編集が可能な模造紙アプリケーションを作成した．本システムを動作させている様子を図 **10** に示す．

3.2 アプリケーション開発の評価

3.2.1 実験内容・結果

本システムの開発容易性を検証するために，本システムにウェブアプリケーションを対応させる際の，各アプリケーション同期方式におけるコード量を計測した．以下にそれぞれのコード変更量を述べる．

キーバリューストア

キーバリューストア方式への対応にあたっては，アプリケーションの内部に手を加える必要がある．ただし，アプ

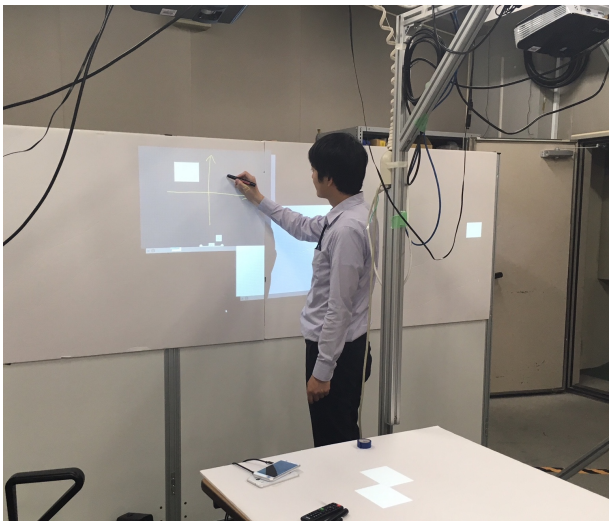

図 10　本システムの動作風景

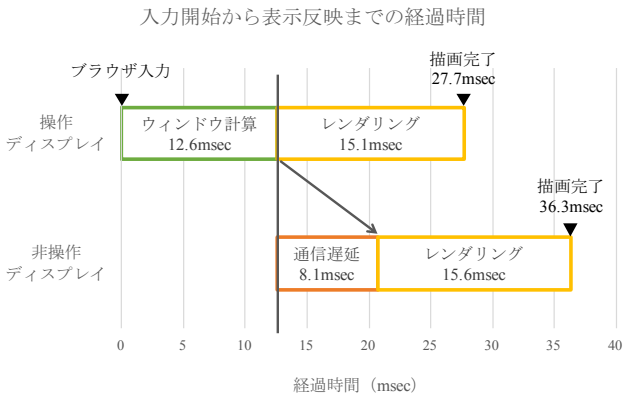

図 11　入力から表示反映までの経過時間

リケーションが，MVC[13] や Flux[14] などのアプリケーション設計方針に沿って内部データをくくりだして一括で管理するよう構成されている場合は，内部データをキーバリューストアと連動させるコードを記述するだけで本システムに対応でき，コードの改変量が抑えられる．模造紙アプリケーションも MVC パターンに沿って作られているため，20 行程度のソースコードの変更で本システムに対応することができた．

文書構造転送

文書構造転送方式については，ライブラリが自動的に画面構成を抽出して転送するため，アプリケーションには文書構造転送用のライブラリを読込む記述を一行追加するだけでよい．しかし動作の面では，模造紙アプリケーションに手書きの文字などを表現するためにビットマップ画像を扱う canvas 要素が含まれていたために，その canvas 要素の画像化・復元に若干の遅延が見られた．

リモートデスクトップ

リモートデスクトップ方式を用いる場合，リモートデスクトップのサーバアプリケーションを利用者端末に導入し，利用者端末のウェブブラウザでアプリケーションを開くだけで良いので，模造紙アプリケーションの変更は不要である．

3.2.2　考察

いずれの方式についても本システムへのアプリケーション適用は容易であったものの，文書構造転送での遅延のように適さない方式も存在するため，開発者はどの方式を採用するかを注意深く選ぶ必要がある．しかし，どの方式が適するかどうかを選ぶのは必ずしも容易ではないため，今後の課題として，利用している表示要素やデータ管理構造から適する自動的にアプリケーション転送方式を選択するといった，同期方式の自動選択機能が求められる．

3.3　応答速度の評価

3.3.1　実験内容・結果

応答性を検証するため，ウィンドウ移動操作の入力から画面反映までの時間を，入力されたディスプレイ機器とその他のディスプレイで 10 回ずつ計測して平均値を算出した．**図 11** に，ウィンドウの移動操作を行い画面に反映されるまでの時間の測定結果を示す．

3.3.2　考察

本グラフから，入力があったディスプレイ機器の応答時間は 28 ミリ秒ほどと，高い応答性を確保できていることがわかる．操作にも違和感はなく，概ね実用的な範囲内であるといえる．

ウィンドウ管理を分散管理でなく，サーバで一括して行う場合，単純には往復通信分の 16 ミリ秒が余分にかかり，入力反映に 44 ミリ秒ほどかかる計算になる．また，今回は有線でのネットワークであったが，無線などの遅延の大きい環境ではさらに応答性が低下する．本システムは入力中の画面については通信遅延の影響を受けないため，通信遅延の影響が大きい環境ではより有効であると考えられる．

4.　関連研究

本研究のような複数機器が連携するウィンドウシステムは，リモートデスクトップなどの単一の連携の延長として提案されているものや，複数機器を用いた協調作業支援システムの汎用化として提案されているものがある．それぞれの例として MetaVNC[15] と ReticularSpaces[4] を取り上げ，本研究との違いを説明する．

4.1　MetaVNC

MetaVNC は，VNC プロトコルを拡張し，複数リモートデスクトップ画面を単一のデスクトップ環境に合成して扱えるようにしたシステムである．ローカルのデスクトップとの融合を目指しており，ローカルのアプリとシームレスに共存できることが利点である．

本システムのリモートデスクトップ方式と構成は似てい

るが，入力の仮想化を行っていないため，複数ディスプレイ上でのまたがった入力を扱うことができない点が本システムと異なる．

4.2 ReticularSpaces

ReticularSpaces は壁面や机型のディスプレイ，および利用者端末を接続して協調作業を行うためのシステムである．ReticularSpaces は，アプリケーションの動作をアクティビティとして規定し，各ディスプレイ機器に導入された ReticUI と呼ばれるウィンドウシステム上で画面を構築する．アクティビティには参加者や場所，タスク進捗状態などのコンテクスト情報を持たせておくことができ，作業の再開や参加者の招集などに活用できる．

しかし，アクティビティとして独自のアプリケーション形式を採っているため，アプリケーション開発には ReticularSpaces が規定する形式に沿わなければならない．そのため，ウェブ開発知識をそのまま使って開発したり，もしくは既存のウェブアプリケーションをそのまま移行したりがいったことが可能な本システムに比べ，開発にかかる負荷が大きい．

5. まとめ

本論文では，議論や協調作業における資料共有・操作を行うための，複数ディスプレイをシームレスに統合するウェブブラウザベースの分散ウィンドウシステムを提案した．

従来の複数機器によるウィンドウシステムは，プラットフォームの混在によりアプリケーション開発が困難であり，また機器間の通信遅延によって応答性が損なわれる問題があった．本システムは，ウェブブラウザを用いた基盤構築によりマルチプラットフォーム性を確保し，分散ウィンドウ管理によって通信遅延を防ぎ応答性を高める．また，ウェブアプリケーションを本システムに容易に適用可能にするために，キーバリューストア方式，文書構造転送方式，リモートデスクトップ方式の三つのアプリケーション同期方式を提供する．評価実験によって，いずれの同期方式を用いてもウェブアプリケーションを少ない修正で本システムに適用できることを示し，本システムの開発容易性を確認した．また，28 ミリ秒という高い応答性を確保できていることを示し，本システムの有用性を確認した．

今後の課題としては，アプリケーション同期方式の自動選択が挙げられる．本システムでは，アプリケーションを作成する際，開発者が適するアプリケーション同期方式を選択しなければならない．よりアプリケーションを開発しやすくするためには，アプリケーションの内容に応じて自動的に同期方式を選択するなど，開発者が同期方式を意識することなく利用できる仕組みが必要である．

参考文献

[1] Lischke, L., Grüninger, J., Klouche, K., Schmidt, A., Slusallek, P. and Jacucci, G.: Interaction Techniques for Wall-Sized Screens, *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*, ITS '15, New York, NY, USA, ACM, pp. 501–504 (online), DOI: 10.1145/2817721.2835071 (2015).

[2] Bellucci, A., Malizia, A. and Aedo, I.: Light on Horizontal Interactive Surfaces: Input Space for Tabletop Computing, *ACM Comput. Surv.*, Vol. 46, No. 3, pp. 32:1–32:42 (online), DOI: 10.1145/2500467 (2014).

[3] Wilson, A. D. and Benko, H.: Combining Multiple Depth Cameras and Projectors for Interactions on, Above and Between Surfaces, *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, New York, NY, USA, ACM, pp. 273–282 (online), DOI: 10.1145/1866029.1866073 (2010).

[4] Bardram, J., Houben, S., Nielsen, S. and Gueddana, S.: The Design and Architecture of Reticularspaces: An Activity-based Computing Framework for Distributed and Collaborative Smartspaces, *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '12, New York, NY, USA, ACM, pp. 269–274 (online), DOI: 10.1145/2305484.2305529 (2012).

[5] 富士通研究所：部屋全体をまるごとデジタル化する UI 技術を開発し、ICT による共創支援の実証実験を開始，http://pr.fujitsu.com/jp/news/2015/07/27.html (2015).

[6] W3C Recommendation: HTML5, https://www.w3.org/TR/html5/ (2014).

[7] Ecma International: ECMAScript 2015 Language Specification, http://www.ecma-international.org/ecma-262/6.0/ (2015).

[8] 可児 潤也, 板倉 宏太, 今井 岳, 木原 英人, 植木 美和, 由良 淳一, 武理一郎：通信負荷にロバストな共同編集システムの評価，情報処理学会研究報告 (2017).

[9] W3C DOM Interest Group: W3C Document Object Model, https://www.w3.org/DOM.

[10] W3C Community Group: Touch Events - Level 2, https://w3c.github.io/touch-events/ (2016).

[11] Google: Google Chrome, https://www.google.co.jp/chrome.

[12] AT&T Laboratories Cambridge: VNC: Virtual Network Computing, https://www.cl.cam.ac.uk/research/dtg/attarchive/vnc/.

[13] Krasner, G. E. and Pope, S. T.: A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80, *J. Object Oriented Program.*, Vol. 1, No. 3, pp. 26–49 (online), available from ⟨http://dl.acm.org/citation.cfm?id=50757.50759⟩ (1988).

[14] Facebook Inc.: Flux Application Architecture for Building User Interfaces, http://facebook.github.io/flux/.

[15] Uchino, S.: MetaVNC - a window aware VNC, https://www.cl.cam.ac.uk/research/dtg/attarchive/vnc/.