情報処理学会論文誌 Vol. 52 No. 2 415–423 (Feb. 2011)

クラウドサービスにおける分散コンポーネントフレームワークの提案

周 劼[†1] 綾木良太[†2]
島田秀輝[†1] 佐藤健哉[†2]

近年，インターネット上に分散して存在する計算資源を利用してユーザに情報やアプリケーションを提供するクラウドコンピューティングの検討がさかんである．クラウドが提供するサービス（クラウドサービス）により，ユーザは多様なアプリケーションが利用可能となる．一方で，ユーザが利用可能な端末も多様化し，1 人が複数の端末を所有し同時に並行して利用する状況になりつつある．このようなユーザが複数の端末を用いてクラウドサービスを利用する際に，現在の構成では，柔軟なクラウドサービスの統合利用，複数端末の同時並行利用が実現困難という問題がある．本研究では，これらの問題を解決するために，新たにクラウドサービスにおける CSD（Context-information for Services and Data）アーキテクチャを用いた分散コンポーネントフレームワークを提案する．また，提案フレームワークに基づいたプロトタイプの実装・評価を行い，柔軟なクラウドサービスを提供するためのフレームワークの実現可能性を検討する．

A Proposal of Distributed Component Framework for Cloud Services

George Zhou,[†1] Ryota Ayaki,[†2] Hideki Shimada[†1] and Kenya Sato[†2]

Recently, studies on the cloud services are becoming popular to provide users with many kinds of information and applications by using computer resources distributed on the Internet. Under this environment, a user is apt to manipulate more than one computer terminal simultaneously towards high usability. When a user makes use of the cloud services with multiple terminals, there occur two issues; low flexibility of cloud service integration, and difficulty of accessing a single service with multiple terminals. In this research, we propose a distributed component framework using CSD (Context-information for Services and Data) architecture for cloud services to address these issues. In addition, we implement and evaluate a prototype base on the framework to confirm usefulness of our proposal.

1. はじめに

近年，インターネット上に分散して存在する計算資源を利用してユーザに情報やアプリケーションを提供するクラウドコンピューティング（あるいは単にクラウド）[1] の利用がさかんになりつつある．クラウドにより提供されるサービスの総称をクラウドサービスと呼び，SaaS（Software as a Service），PaaS（Platform as a Service），HaaS（Hardware as a Service）/IaaS（Infrastructure as a Service）に分類[2] されるのが一般的である．これらクラウドサービスにより，ユーザは Web ブラウザを利用することで，ワープロ，表計算に加えて，チャット，電話など多様なサービスが利用可能となる．たとえば，セールスフォース[3] の顧客管理サービス，Google Apps[4] における電子メール，チャット，ドキュメント作成などのアプリケーション提供，Amazon EC2[5] によるソフトウエア実行のためのプラットフォーム提供などがあげられる．一方，コンピュータやネットワーク技術の発展，低価格化により，ユーザが利用可能な端末も多様化し，1 人が複数の端末を所有し同時に並行して利用する状況になりつつある．

クラウドの技術によりインターネット上の計算資源を Web（通常は HTTP）を通して利用することで，端末ごとの仕様の差異を吸収し利用可能となるが，その反面，ユーザがコンピュータシステム自体を保有した場合に可能となるサービスのカスタマイズや運用の変更が困難となり，複数端末を利用したサービスの協調利用も容易には実現できない．また，基本的にはすべてのデータがクラウドに集約されるため，セキュリティや提供されるサービスの停止などの懸念がある．加えて，複数のクラウドサービスを連携して利用することも困難である．そこで本研究では，このような問題点の解決を目指すアプローチとして，新しくクラウドサービスにおける CSD（Context-information for Services and Data）アーキテクチャを用いた分散コンポーネントフレームワークを提案し，設計，実装，評価を通してその実現可能性を検討する．

本論文では，まず 2 章において，クラウドサービスの構成と問題点を述べる．その後，3 章では，本研究における提案方式の詳細を説明する．4 章では，本提案方式を用いたフ

†1 同志社大学理工学部情報システムデザイン学科
Department of Information Systems Design, Doshisha University
†2 同志社大学大学院工学研究科情報工学専攻
Graduate School of Engineering, Information and Computer Science, Doshisha University

© 2011 Information Processing Society of Japan

レームワークのプロトタイプ実装を行い，実現可能性検討のための評価を行う．5 章で関連研究について解説した後に，最後に 6 章で，まとめと今後の課題を記す．

2. クラウドサービス

2.1 クラウドサービス構成

現在，クラウドサービスのベンダによって，多数のプラットフォームが提供されているが，普及率，汎用性，拡張性を考慮するとプラットフォームの機能が Web ブラウザに集約される場合がほとんどである．クラウドサービスを実現するための一般的なシステム構成を**図 1** に示す．ユーザの端末においてユーザインタフェースである Web ブラウザが動作し，HTTP リクエストが端末からクラウド側にあるそれぞれの SaaS や HaaS などを実現するプラットフォーム上のリソースにアクセスを行う．そこで処理が行われその結果を HTTP のレスポンスとして XML 形式のデータが返送される．HTML および CSS を利用した形態で，ユーザインタフェースとなる Web ブラウザ上に結果が表示される．端末とクラウド上の計算資源とのやりとりは HTTP を利用しデータは XML 形式であるため，Web ブラウザのみの利用で仕様が異なる端末においても実現可能となる．さらに，JavaScript [8]，Ajax [9] をはじめ，HTML5 [10] や Comet [11] の技術の登場により，より高い利便性をユーザに提供している．

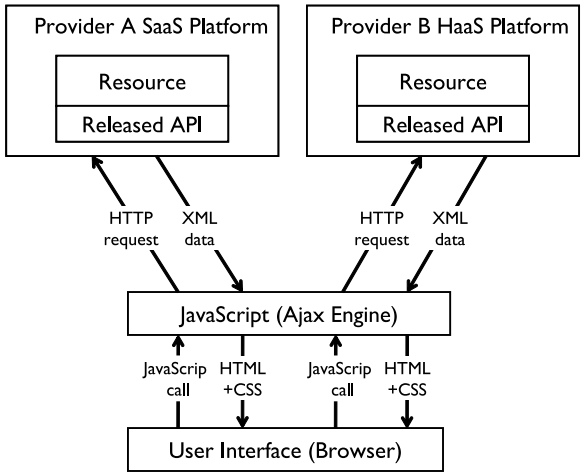

図 1　クラウドサービスのシステム構成
Fig. 1　System architecture for cloud services.

2.2 クラウドサービスの利用形態の問題点

現在のクラウドサービスの利用形態において次のような問題点がある．

- 柔軟なクラウドサービスの統合利用
  各社が提供するクラウドサービスにおいては，独自のユーザインタフェースやファイルフォーマットが存在し，機能が類似したサービス間へのスケールアウトができないという問題点がある．一例として，ドキュメント作成のサービスとして Web ブラウザ上で起動する Microsoft Office Web Apps [12] と Google Docs [13] がある．Office Web Apps で作成したファイルに対して，変換ツールを使用せずに Google Docs では利用できない．類似したサービスであっても，変換ツールや変換機能が存在しない場合は，サービス間の移行ができない．また，現在のクラウドサービスでは，特に SaaS において，そのほとんどが端末側で実行するアプリケーションソフトウェアをそのままサービスとしてクラウド側に移行したものがほとんどであり，ソフトウェアアーキテクチャもそのままクラウドに適用されている．そのため，サービスとデータが一体化されており，データもサービス提供側であるクラウドに配置する必要がある．つまり，サービスとデータが 1 対 1 の対応関係であり，特定のサービスのために作成されたデータを，種類の異なるサービスでは利用できず，複数のサービスにおいて複数のデータを利用するような多対多の柔軟な統合利用が実現不可能となる．
- 複数端末の並行利用
  現在のクラウドサービスにおいては，一般的に，Cookie [15] といったユーザ特定情報をもとに，ユーザの使用状態やユーザごとのアプリケーションの状況に関する設定や履歴などをクラウドサービス内に置くことにより，サービスの継続的利用を実現している．したがって，1 人のユーザが複数端末を所有し利用する状況を想定した場合，クラウドサービスはユーザが使用している端末側の情報を把握していないため，各端末の性能差，使用場所などに適したカスタマイズができない．また，設定や履歴などをすべてクラウドサービス内に保存・管理する際，クラウドサービスから個人情報が漏洩する可能性がありセキュリティの問題が発生する可能性がある．そのため，一部の機密性が高い情報をクラウドサービス内に保存せずに，ユーザ側の複数の端末で共有して保存するという要求に対して，現在のクラウドサービスではサービスとデータが明確に分離されて管理されていないため，実現することができない．このためには，クラウドおよび複数端末の並行利用の機能を実現する必要がある．ここでいう複数端末の並行利用とは，1 人のユーザがある時点である端末を利用し，また，別の時点で別の端末を利用するよ

© 2011 Information Processing Society of Japan

うに，時分割で複数端末の利用状況を変更する場合を想定している．端末間においては何らかの手法（たとえば，端末間のネットワーク接続による情報共有，ユーザが所有する USB メモリによる情報の共有・コピー，サービス提供とは別の外部のファイル共有など）により，ユーザが端末を変更した際に情報を共有（移動）させることを前提としているが，具体的な手段に関してはここでは規定しない．

3. 提案手法

3.1 概　　要

本研究では，2.2 節で述べた柔軟なクラウドサービスの統合利用，および，複数端末の同時並行利用という既存クラウドサービスの問題点を解決するため，新たに CSD アーキテクチャと呼ぶ方式を考え，それに基づいた分散コンポーネントフレームワークを提案する．**図 2** にフレームワークの動作手順の概略を示す．

本フレームワークにおいては，サービスを利用するタイミングでサービス利用開始，サービス利用中，サービス利用終了に分けることができる．あらかじめ登録されている複数のサービス候補からユーザが実際に使用するサービスを選定する．選定したサービスをデフォルトサービスとしてコンテキスト情報に登録し，データをサービスリソースにインポートした後は，従来通りサービスが Web ブラウザで立ち上がり，サービスリソース，およびデータリソースを登録し，端末リソースそれぞれに登録しているコンテキスト情報を確認し，変更がある場合，そのコンテキスト情報を更新する．

3.2 システム構成

提案する分散コンポーネントフレームワークでは，**図 3** に示すように端末（Terminal）とクラウドサービスの間に CSD アーキテクチャを構築し，1 階層設けることで，各クラウドサービスをユーザから隠蔽する．そのため，ユーザは，クラウドのサービス形態，データの保存先，利用する端末（PC，携帯など）を意識することなく柔軟にクラウドサービスの統合利用が可能となる．CSD アーキテクチャは，端末側においても，自らが提供者となってサーバ側でも配置が可能な構成であるが，今回の実装においては，端末側に配置した構成としている．提案する分散コンポーネントフレームワークは，CSD アーキテクチャを構成する各コンポーネントとユーザインタフェースから構成される．詳細を次に示す．

- コンポーネント S

  SaaS を抽象化したコンポーネントである．従来のクラウドサービスにおいて，ユーザが文書を作成したい場合には，Office 2010 Word や Google Docs などの各 SaaS を選択し，利用する必要があった．一方，CSD アーキテクチャでは，利用する各 SaaS をコンポーネント S が管理し，ユーザから SaaS を隠蔽する．そのため，ユーザは，文書を作成するコンポーネント S を選択し，利用すればよい．

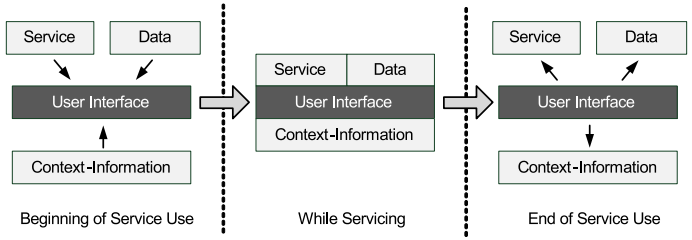

図 2　フレームワークの動作手順概要
Fig. 2　Outline of action sequence for proposed framework.

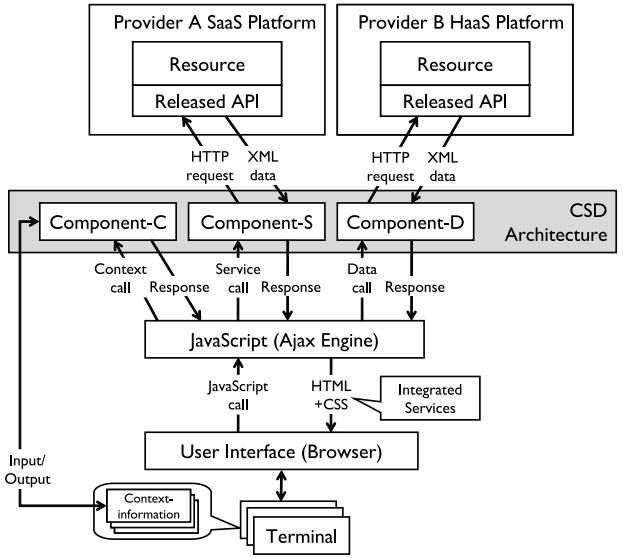

図 3　CSD アーキテクチャ
Fig. 3　CSD architecture.

© 2011 Information Processing Society of Japan

- コンポーネント D
  HaaS を抽象化したコンポーネントである．コンポーネント D は，ユーザが利用しているデータの保存先情報を管理する．ユーザのデータが複数の HaaS 上に分散している場合でもコンポーネント D が統合的に管理しているため，ユーザ端末は，コンポーネント D に問合せを行うだけでよい．
- コンポーネント C
  利用するクラウドサービス（SaaS，HaaS）の情報，利用状況，利用者の状況を示すコンテキスト情報を管理する．コンポーネント C は，1 ユーザに対し，1 つのコンテキスト情報ファイルを用意し，管理している各情報を記録する．コンポーネント C は，コンテキスト情報ファイルに基づいて，コンポーネント S とコンポーネント D を連携させる．
- ユーザインタフェース
  提案する分散コンポーネントフレームワークでは，拡張性，汎用性，普及率の理由から既存の Web ブラウザを拡張したユーザインタフェースを使用する．

ここではユーザインタフェースとして Web ブラウザを想定している．これは，インターネットにおける Web ブラウザの利用率は高く，現在のクラウドサービスのほとんどが Web ブラウザを利用しているためである．しかし，ユーザインタフェースとして Web ブラウザの利用に限定しているわけではなく，何らかの Script 言語（本研究では現在主流である JavaScript を採用）が動作する環境において独自のユーザインタフェースで実現することも可能となる．実装において Firefox を利用しているのは，アドオンのプログラムを簡単に実装できるためであり，Web ブラウザにアドオンが利用できれば，Firefox である必要はない．また，本アーキテクチャで Ajax を利用した理由は，異なるドメインからリソースを取得する際にユーザの操作の負担が減り利便性が向上すると考えられるためであり，Ajax の利用を前提としているわけではなく非同期処理でなくても動作可能である．

ここでいうコンテキスト情報とは，既存のクラウドサービス（SaaS）の情報，データとその保存場所（HaaS）の情報，端末やネットワークの状況などを含めて利用者側の状況を示す情報の 3 つのタイプからなり，それぞれが ID や名前，位置情報などで構成される．サービスを具体化する際，サービスによりデータモデルの記入されていないパラメータの場合も想定し，ここで示す以外の情報も追加可能としている．

複数のクラウドサービスの結合に関して，たとえば，SaaS と HaaS を結合する際，ユーザがクラウドサービスの組合せ方法を指定する必要がある．具体例として，Google Docs を利用して Box.net にある文書データを編集する状況を考える．Google Docs，Box.net の情報を含むコンテキスト情報の構成要素の例を**表 1** に示す．サービスやデータの ID を利用し，service_id 1 と，data_id 1 を組み合わせて結合したサービスを利用する場合，2 つのサービスがコンポーネントに具現化され，フレームワーク上で結合されることになる．そして，コンテキスト情報をもとに，コンポーネントを具現化するために，表 1 に示すコンテキスト情報の構成要素の HTTP のリクエストを行う．**図 4** に SaaS のリクエスト例，**図 5** に HaaS のリクエスト例を示す．

表 1　コンテキスト情報構成要素
Table 1　Elements of context information.

Type	Item	Description	Example data
service	service_id	Service ID	1
	service_title	Service title	Google Docs
	service_extension	Service type	Document
	service_location	Location of service	schemas.google.com
data	data_id	Data ID	1
	data_title	Data title	test
	data_extension	Data type	Document file
	data_location	Location of data	www.box.net
user	user_id	User ID	1
	user_title	User's name	Zhou
	user_name	Name of user's terminal	Zhou-PC1
	user_location	Location of terminal	IP address

```
POST /feeds/documents/private/full HTTP/1.1
Content-Length: 287
Content-Type: application/atom+xml

<?xml version='1.0' encoding='UTF-8'?>
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:docs="http://schemas.google.com/docs/2007">
    <atom:category scheme="http://schemas.google.com/g/2005#kind"
      term="http://schemas.google.com/docs/2007#document"
      label="document"/>
    <atom:title>new document</atom:title>
    <docs:writersCanInvite value="false" />
</atom:entry>
```

図 4　SaaS リクエストの例
Fig. 4　Example of SaaS request.

© 2011 Information Processing Society of Japan

```
https://www.box.net/api/1.0/download/ <auth_token> / test
```

図 5　HaaS リクエストの例
Fig. 5　Example of HaaS request.

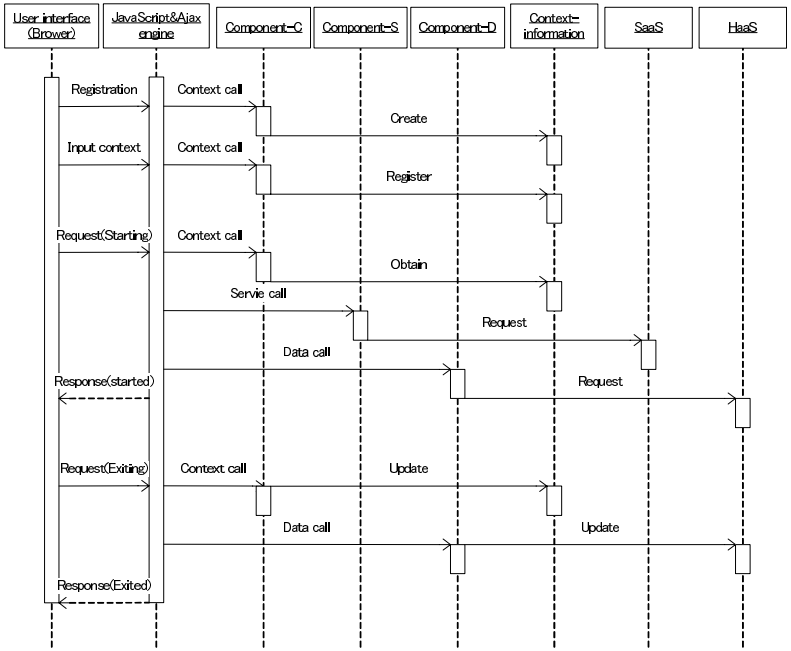

図 6　メッセージシーケンス
Fig. 6　Message sequence.

3.3 動　　作

提案する分散コンポーネントフレームワークのメッセージシーケンスを**図 6** に示す．また，下記に詳細を記載する．

(1)　ユーザ登録
ユーザが新規ユーザ登録を行うと，コンポーネント C は，コンテキスト情報ファイルを生成する．

表 2　評価環境
Table 2　Evaluation environment.

OS	Windows XP Professional SP2
CPU	Pentium 4, 3.00 GHz
RAM	1 GByte

表 3　実装仕様
Table 3　Implementation specification.

Web browser	Firefox3.5.6
Implementation style	add-on
SaaS	Google Docs
HaaS	Box.net

(2)　コンテキスト情報登録
ユーザは，すでに使用しているクラウドサービスがある場合，登録を行う．コンポーネント C は，コンテキスト情報ファイルにユーザの登録情報を記入し，更新する．

(3)　起動時
コンポーネント C のコンテキスト情報ファイルに基づき，コンポーネント S，コンポーネント D の管理している情報を取得し，分散したコンポーネントを組み合わせて，サービスを起動する．

(4)　終了時
コンテキスト情報ファイルと利用した HaaS 上のデータを更新し，終了する．

4. 実現可能性検討

4.1 プロトタイプ実装

提案する分散コンポーネントフレームワークの実現可能性を検討するために，プロトタイプ実装を行った．評価環境を**表 2** に示す．次に，**表 3** に実装仕様を示す．プロトタイプ実装では，Web ブラウザとして Firefox を用い，本提案を Firefox のアドオンとして実装した．また，SaaS として Google Docs，HaaS としてオンラインストレージサービスである Box.net [7] を用いた．

- ユーザインタフェース⇔コンポーネント S，D
ユーザインタフェースとクラウド上のリソースであるコンポーネント S，D との通信機能はリソースプロバイダが提供している Web API [14] を利用する．Web API は Web

© 2011 Information Processing Society of Japan

```
function getURL() {
  new Ajax.Request('uesr.json',
    {method:'get',onComplete:displayResult});
}

function displayResult(req) {
  var data = req.responseText.evalJSON();
  var html = '<table border="1">';
  for (var i=0; i<data.personals.length;i++) {
    html = html + data.URL[i].domain;
    html = html + data.URL[i].id;
    html = html + data.URL[i].name;
  }
  html = html + '</html>';
  ('result').innerHTML = html;
}
```

図 7　JSON ファイル読み込みの記述例
Fig. 7　Description example for reading JSON files.

サイトなどの開発を効率的に行うための技術である．Web サイトなどの高機能なコンテンツをより短期間・低コストで開発できるという利点がある．また，アプリケーションがほかのアプリケーションに HTTP でアクセスすることで処理を依頼できる．提供する機能は URL で指定できるようになっている．すなわち，目的の機能の URL にアクセスしてパラメータを渡せば，処理結果が XML データで返ってくる．

- ユーザインタフェース⇔コンポーネント C
  コンポーネント C として JSON[16] を利用するため，JSON 形式であるコンテキスト情報に入力や出力が必要となる．**図 7** に示すように，URL 情報とデータファイルの名前を取得する場合，JSON ファイルからそのまま抽出して組み合わせるだけで実現可能とする．

4.2 実　行　例

実装したシステムの実行画面の表示例を**図 8** に示す．CSD を無効にした場合が左図であり，CSD を有効にした場合が右図である．CSD を利用したフレームワークは，Web ブラウザのアドオンとしての拡張であるため，既存の Web ブラウザとレイアウトが同じようにメニューやボタンを作成することが可能である．また，ユーザが Web ブラウザ上にクリックやドロップなどのようなイベントを発生させることにより，コンテキスト情報の読み込み，

図 8　CSD 無効時（左）および CSD 有効時（右）のファイル読み込み画面
Fig. 8　Screenshot for reading a file without CSD (left) and with CSD (right).

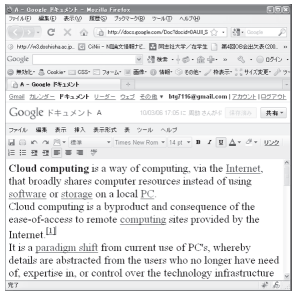

図 9　サービス組合せ実行画面
Fig. 9　Screenshot for service combination.

コンポーネントの具現化，サービスの結合がバックグラウンドで実行される．そして，**図 9** に示すように，フレームワークを経由して結合したサービスが Web ブラウザ上に立ち上がる．ユーザは初回のみサービスとデータのバインディング（組合せ方）を指定する作業が必要であるが，それ以降，初回のバインディングの状態が保持され，フレームワークがバックグランドで動作しバインディングが自動的に行われるので，画面を占有することはない．

4.3 評　　価

本フレームワークを用いたクラウドサービスの実現可能性検討のため，従来のクラウド

© 2011 Information Processing Society of Japan

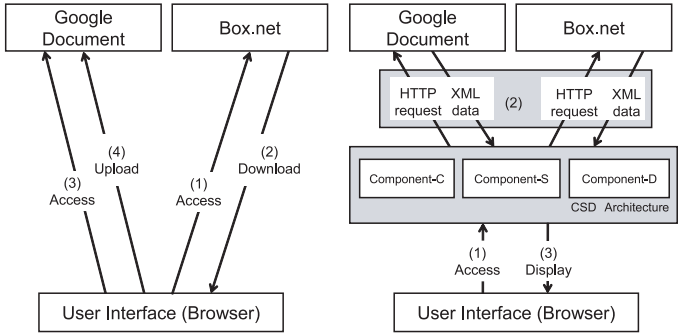

図 10　従来方式（左）および提案方式（右）の評価手順
Fig. 10　Evaluation sequence without CSD (left) and with CSD (right).

サービス（従来方式）と CSD アーキテクチャを利用した方式（提案方式）を比較するため，ここでは，特定のユーザが Google Docs を利用して，Box.net に保存されている文書ファイルを開くことを想定した手順を評価する．従来方式の場合，ユーザが (1) Box.net にアクセス，(2) 文書ファイルをダウンロード，(3) Google Docs にアクセス，(4) 文書ファイルのアップロードを行う手順となる．提案方式の場合，ユーザが (1) Web ブラウザ上のボタンをクリック（コンテキスト情報を格納），(2) 統合サービスが起動（API 経由で Google Docs と Box.net それぞれ同時にアクセス），(3) 統合サービスを表示（XML データの結果を統合して表示）という手順となる．これらの手順を**図 10** に示す．

時間の観点からそれぞれの手順を比較すると，従来方式では，複数のサービスを統合利用するとき，ユーザがアクセス→ダウンロード→アクセス→アップロードの手順どおりに実施する必要がある．実際には，ユーザが行う操作のための時間も必要となる．一方，提案方式では，コンテキスト情報をもとに，異なるドメインにあるリソース（サービス，文書ファイル）に対して同時にリクエスト可能となる．また，レスポンス結果を XML 形式で統一しているので，フォーマットを変更する必要がなく，そのまま組み合わせて表示することができる．

Box.net に格納されているドキュメントデータのサイズを変化させた際の評価結果を**表 4** に示す．評価結果は 10 回測定を行った際の平均値である．評価結果が示すように，本提案を利用した場合，利用しない場合と比較して遅延時間がいずれも半分近く短縮された．

従来方式は，ユーザが行う操作のための時間を除いて，アクセス→ダウンロード→アクセス→アップロードにおける通信時間となる．一方，提案方式では，リソース（サービス，文書ファイル）に対して同時にリクエスト可能なので，従来方式のようにダウンロードした後にアップロードするという手順をとる必要がない．XML へのデータフォーマットの置き換え，組合せ処理に時間が必要となるが，リソースの転送にともなう通信時間と比較してかなり小さいため，合計として従来方式より提案方式の遅延時間が短縮される結果となった．

表 4　評価結果
Table 4　Evaluation results.

	100 kB	200 kB	300 kB	400 kB
Current Method	5.82 sec	6.95 sec	8.03 sec	8.62 sec
Proposed Method	3.31 sec	4.34 sec	4.88 sec	5.03 sec

5. 関 連 研 究

クラウドサービスの統合利用を着目した SaaS and Integration Best Practices [17] では，複数のコネクタ機能を提供する Integration as a Service Provider を利用し，SaaS の統合利用を実現している．しかし，本構成では，統合して利用できるのは，Integration as a Service Provider がサポートしている特定の SaaS に限定されるという条件がある．

Cloud Federation Manager [18] では，クラウドサービスの機能に着目し，複数のクラウドサービス間でスケールアウトやディスカバリを実現している．本研究では，クラウドサービスの統合利用の着眼点をユーザ側に置いているため，我々の研究とは着眼点が異なる，また，複数端末が用いられる場合が取り上げられていないという問題点もある．

本研究の CSD アーキテクチャは，クラウドサービスの統合利用において，利用者側の状況を考慮したコンテキスト情報を導入し，ユーザ自身でクラウドサービスの組合せ方を決定することが可能となる．また，コンテキスト情報の構成要素に従って，端末側のデータファイルも取り込むことができる．すなわち，同一のサービスであっても，利用者のニーズに応じて，外部に保持したくないデータに対しても外部のサービスが利用可能となる．今回の実装においては，Ajax や JavaScript などのリッチクライアント技術を利用して端末の Web ブラウザをアドオンにより拡張する方式を採用した．これにより，サービスを提要する側と利用者との間のみで情報のやりとりを行うことができる．また，ユーザインタフェースの構成が通常の Web ブラウザと変わらないので，ユーザが新たなユーザインタフェースを学習する必要もなく，負担も少なくなるという特徴がある．

© 2011 Information Processing Society of Japan

6. おわりに

一般に，ユーザが複数の端末を用いてクラウドサービスを利用する際に，(1) 柔軟なクラウドサービスの統合利用ができない，および，(2) 複数端末の並行利用を考慮していないという問題点がある．本研究では，クラウドサービスにおける CSD（Context-information for Services and Data）アーキテクチャを用いた分散コンポーネントフレームワークを提案することで問題の解決を図った．そして，提案する分散コンポーネントフレームワークの実現可能性の検討するために，提案に基づいた実装をし評価を行った．評価結果において，提案アーキテクチャの設計に基づいて実装したプロトタイプが正常に動作することを確認した．また，複数のデータサイズの処理時間が，提案手法は従来手法と比較して 2 倍近く速くなっていることを確認した．したがって，提案方式の実現性および有用性が確認できた．本研究は，柔軟なクラウドサービスを実現するためのフレームワークの提案と実現可能性の検討であり，今後の課題としてより幅広い実装環境においてフレームワークの有効性を確認する予定である．

謝辞　本研究は科研費（21500084）の助成を受けたものである．

参考文献

1) Vouk, M.: Cloud Computing – Issues, Research and Implementations, *CIT Journal of Computing and Information Technology*, Vol.16, No.4, pp.235–246 (2008).
2) Rimal, B.P., Choi, E. and Lumb, I.: A Taxonomy and Survey of Cloud Computing Systems, *Proc. 5th International Joint Conference on INC, IMS and IDC*, pp.44–51 (2009).
3) Salesforce.com: CRM (Customer Secure Login), http://www.salesforce.com/jp/ (accessed 2010-01-30).
4) Google: Google Apps, http://www.google.co.jp/apps/intl/ja/business/index.html (accessed 2010-01-30).
5) Amazon: Amazon Elastic Computing Cloud (Amazon EC2), http://aws.amazon.com/ec2/ (accessed 2010-01-30).
6) Microsoft: Windows Azure Platform, http://www.microsoft.com/windowsazure/ (accessed 2010-01-30).
7) Box.net: Simple Online Collaboration, http://box.net/ (accessed 2010-01-30).
8) Mozilla: Core JavaScript 1.5 Guide, JavaScript Overview, https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/JavaScript_Overview (accessed 2010-01-30).
9) W3C: XMLHttpRequest, W3C Working Draft 19 Nobember 2009, http://www.w3.org/TR/XMLHttpRequest/ (accessed 2010-01-30).
10) W3C: A Vocabulary and Associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010. http://www.w3.org/TR/html5/ (accessed 2010-03-10).
11) Arcand, J.-F. and Goddard, T.: Asynchronous Ajax for Revolutionary Web Applications, JavaOne Online Technical Sessions, TS-5250 (2008). http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-5250&yr=2008&track=nextweb (accessed 2010-01-30).
12) Microsoft: Office Web Apps, http://www.microsoft.com/japan/office/2010/webapps/default.mspx (accessed 2010-01-30).
13) Google: Google Docs – What's new?, http://www.google.com/google-d-s/whatsnew.html (accessed 2010-01-30).
14) W3C: XMLHttpRequest Level2, W3C Working Draft 20 August 2009, http://dev.w3.org/2006/webapi/XMLHttpRequest-2/ (accessed 2010-01-30).
15) Kristol, D. and Montulli, L.: HTTP State Management Mechanism, Request for Comments (RFC) 2965, The Internet Society (2000).
16) JSON: JSON, http://www.json.org/json-ja.html (accessed 2010-01-30).
17) Hai, H. and Sakoda, S.: SaaS and Integration Best Practices, *FUJITSU Sci. Tech. J.*, Vol.45, No.3, pp.257–264 (2009).
18) Takeda, K., Itoh, M., Yamanaka, K. and Murakami, A.: The design and implementation of "Cloud Federation Manager", which enables scale-out and disaster recovery between cloud systems, Information Processing Society of Japan, No.3, pp.35–36 (2010).

(平成 22 年 5 月 31 日受付)
(平成 22 年 11 月 5 日採録)

周　劼

2010 年同志社大学工学部情報システムデザイン学科卒業．現在，京都大学大学院情報学研究科社会情報学専攻博士課程在学中．

© 2011 Information Processing Society of Japan

綾木　良太

2009 年同志社大学工学部情報システムデザイン学科卒業．現在，同志社大学大学院工学研究科情報工学専攻博士課程在学中．

島田　秀輝（正会員）

同志社大学理工学部研究員．2001 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2004 年同研究科博士後期課程修了後，特任助教を経て，2009 年より現職．博士（工学）．

佐藤　健哉（正会員）

同志社大学理工学部教授．1986 年大阪大学大学院工学研究科電子工学専攻修士課程修了．同年住友電気工業情報電子研究所入社．1991〜1994 年スタンフォード大学計算機科学科客員研究員．2000 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．米国 AMI-C, Inc. Chief Technologist を経て，2004 年より現職．2008 年から名古屋大学大学院組込みシステム研究センター特任教授兼務．博士（工学）．ACM，IEEE-CS，自動車技術会各会員．

© 2011 Information Processing Society of Japan