情報処理学会論文誌 Vol.55 No.3 1186–1200 (Mar. 2014)

クラウドコンピューティング環境での認証連携における動的属性利用技術の提案と評価

下道 高志[1,a)] 佐々木 良一[1]

受付日 2013年3月25日, 採録日 2013年12月4日

概要：複数のネットワークドメイン上で複数のサイトを連携するサービスは，クラウドコンピューティングの環境下で増加すると予想される．そこで必要とされる技術は，単なるサイト間の認証連携だけでなく，分散された属性情報を利用するサービスのための技術である．扱われる属性情報は静的属性情報に加え，動的属性情報も今後増加すると考えられる．本稿ではアイデンティティ管理，サービス技術であるSAML/ID-WSF に注目し，クラウド上への適用の実際と問題点を考察したうえで ID-WSF を拡張し，離れた地点においてもプライバシを配慮しながら高速かつ安全に大量の動的属性を利用できる方式の提案を行うとともに，提案する方式が柔軟性，高速性，安全性の面で有効であることの評価も行った．

キーワード：アイデンティティ，クラウドコンピューティング，連携技術

A Proposal and an Evaluation of Technology on Federated Identity and Usage of Attributes in Cloud Computing

Takashi Shitamichi[1,a)] Ryoichi Sasaki[1]

Received: March 25, 2013, Accepted: December 4, 2013

Abstract: Federated services among multi-domain network are expected to be widely deployed onto cloud computing environment. Technology not only for federated identity but also services using distributed attributes, that are not only static but also dynamic, are required. Focusing on SAML and ID-WSF, that are technology for identity management and services with privacy, this paper discusses deployments and problem of them in the real world to extend ID-WSF, proposes adaptable, fast and safe technology which can safely use attributes, and evaluates the effectiveness of the proposed technology.

Keywords: identity, cloud computing, federation

1. はじめに

さまざまな消費者向けサービスや企業情報システムでは，同一ネットワークドメイン上のサイトで完結するサービスだけでなく，複数のサイトがドメイン間で連携するサービスが増えている．連携されたサービスをユーザが利用する場合，複数サイト間での認証連携が必要となり，SAML（Security Assertion Markup Language）[1]，OpenID [2]，OAuth [3] といった仕様が規定されている．連携サービスを提供する各サイトが保有する個人属性情報の集合をアイデンティティと呼び，認証連携を行うことによって連携される個人属性情報の集合を，連携アイデンティティ（Federated Identity）と呼び [4]，そのための技術を連携アイデンティティ技術と呼ぶ．

連携アイデンティティにおける個人の属性情報利用のためには，個人情報保護やプライバシを考慮したセキュアなプロトコルが必要であり，SAML や SAML を拡張した Shibboleth [5] を利用する属性転送方式や，SAML を利用しつつ属性利用の本人確認の仕様を備える Liberty Alliance の ID-WSF（Identity Web Service Framework）[6] 等が規定されている．SAML はサイト間のセキュアな認証連携の

[1] 東京電機大学
Tokyo Denki University, Adachi, Tokyo 120–8551, Japan
[a)] takashi.shitamichi@private.email.ne.jp

ために数多く研究され適用が進んでいる．

従来属性情報とは，氏名，住所，生年月日，性別といった静的属性（static attribute）を指してきた．その一方で時間とともに変化する個人のライフイベントに関連した情報，たとえば位置情報，体温，脈拍，血圧等の生体情報，摂取した食事や飲物等の情報もある．これらをライフログと呼ばれることもあるが，本人の嗜好や行動を示すアイデンティティであり属性である．ゆえに動的属性（dynamic attribute）と定義することができる [7]．動的属性は静的属性と違い，短い時間の間に同一の属性が変化したり，新たな属性が追加されたりすることが考えられる．さらに取得期間が長ければ，変化する属性値の総数は莫大になり，データ量も膨大となる場合も想定される．既存のサイトは静的属性と動的属性を一緒に扱ってきたが，動的属性の性質を考えると，静的属性とは分離され独立に管理・運用されるサイトすなわち動的属性情報プロバイダを考えることもできる．Facebook や Twitter は API を組み合わせることによって特定個人のイベントを抽出できるが，このイベントも整理すれば動的属性ととらえることもできる．またユーザにサービスを提供するサイトにとっては，静的属性および動的属性を扱うサイトと柔軟に連携し，独自のサービスを提供できるかが重要と考えられる．

過去，静的属性を扱う技術に関して，複数のサイト間における高速な認証連携を行うために，ユーザエージェント機能のローミングによる認証時間の短縮の提案 [8] や，モバイル端末のアクセスに対して，ロールベースの SSO プロキシを適用することによる認証時間の短縮を行う技術の提案 [9] は行われているが，LAN や特定のネットワーク等の狭域の環境における SSO の認証時間の短縮のみを目的としており，クラウド上のような広域でのサービスを想定していない．また属性情報管理については，安全性についての研究 [10] や異なる連携プロトコルでの実現方法の研究 [11] も行われている．しかし，静的属性と動的属性を分離し活用する技術は，狭域の環境を対象として過去，研究が行われているが [12]，Web サービスの観点で認証連携とあわせ，日々蓄積されていると推測される動的な属性情報を，速度と安全性を十分に配慮して扱う技術としては考案されていない．

属性情報を取り扱うサイトにおいては，ユーザのインタラクション（会話）時に属性転送を高速に行わなければ，ユーザエキスペリエンス（ユーザ体験）に影響を与える可能性がある．サイトがユーザとレイテンシ（遅延時間）が小さい地点に存在する場合には，属性転送時間は問題にならないと考えられるが，サイトがクラウド上のどこか，地球の反対側のような地点に存在している場合は遅延時間が大きくなると予想され，その結果，サービスを利用するユーザにとってのラウンドトリップ時間（RTT: Round Trip Time）は長くなり，ユーザ体験に大きな影響を与えると想定される．

そこで筆者は，認証連携および属性利用技術である SAML/ID-WSF を利用しつつ，静的な属性情報と動的な属性情報を高速かつ安全に扱うために，ID-WSF を拡張し，静的な属性情報と動的な属性情報を分離したうえで，動的な属性情報をサービス提供サイトに遅延時間が小さいサイトへローミングするアーキテクチャを考案した．Web サービスをはじめとしたアイデンティティ管理の手法や研究において，属性情報をローミングする技術の提案は，過去に行われていない．本稿では，2 章で既存の認証連携，属性利用技術を述べ，3 章で既存技術のクラウド適用の問題点を実験の結果から提起し，4 章で問題点を解決するための新方式の提案を行い，5 章で提案方式の評価を行い，6 章で提案方式のまとめを行う．

2. 既存の認証連携，属性利用技術

前章で連携アイデンティティを取り扱う技術として，SAML と ID-WSF が適用，研究されていることを述べた．SAML はさまざまなサービスにおける認証連携として，また ID-WSF は属性利用の技術として利用されている．本章ではアイデンティティ連携技術の面より SAML と，ID-WSF の考察を行う．

2.1 セキュリティ情報交換の技術としての SAML

SAML は連携シングルサインオン（SSO: Single Sign On）を行うための技術ととらえられることが多いが，元々はセキュリティ情報を，ネットワークを通して交換するためのフレームワークとして，2000 年代初頭に考案された．

SAML で定義されるアイデンティティ提供者（IdP: Identity Provider）とは，名前や年齢といったトークンの要素（クレーム）を作成する機能（オーソリティ）であり，セキュリティトークンを発行する機構（STS: Security Token Service）を運用する．またサービス提供者（SP: Service Provider）はクレームを利用することによってユーザを特定し，アプリケーション・サービスを提供する．SP は，クレームを利用するにあたり，その正当性について IdP から証明を受ける必要がある．この IdP からの応答を SAML では SAML アサーションと呼び，認証・認可・属性情報を XML で記述している．SAML による連携 SSO の特徴の 1 つは，サイト間の相互信頼（トラストサークル）を形成したうえで，ユーザが Web ブラウザ等の UA（User Agent）によって SSO を可能とすることである．トラストサークルを実現する技術的な実現方式として**図 1** に示すように，各サイトは個別にユーザのアカウントを保持し，個別のアカウントどうしを互いの仮名で紐づけ（リンク）を行う．この方法により各サイトは独自にユーザアイデンティティを保持し続ける一方，サイト間では共通する情報が存在しないため，実際のユーザ ID の流出や名寄せによ

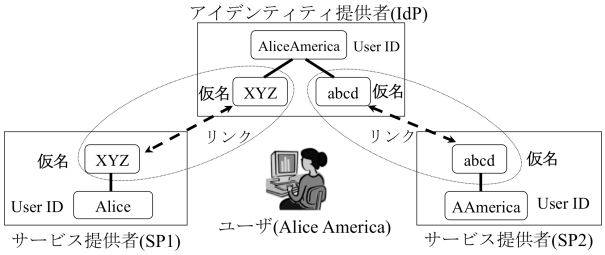


図 1 SAML によるアイデンティティ連携方式
Fig. 1 Identity federation using SAML.

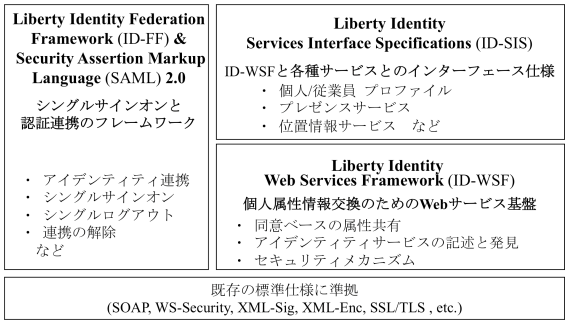


図 2 Liberty Alliance 仕様のフレームワーク
Fig. 2 Framework of Liberty Alliance specification.

るプライバシ情報漏洩を防止することが可能となる．

SAML 仕様書ではアサーション，プロトコル，特定のプロトコルメッセージの組合せ仕様であるバインディングに加え，SSO 等のユースケースをプロファイルとして規定している．Web ブラウザによる SSO のプロファイルとしては次の 3 種類を定義している [13].

(1) SP-initiated SSO: Redirect/POST Bindings
(2) SP-Initiated SSO: POST/Artifact Bindings
(3) IdP-Initiated SSO: POST Binding

SSL/TLS によるチャネルセキュリティの確保に加え，SAML アサーションを Web ブラウザのリダイレクト経由で直接引き渡さず，信頼されたサイト間を直接 SOAP（Simple Object Access Protocol）通信で SAML アサーションを引き渡すことによって，よりセキュリティを確保する (2) の方式が注目されている[*1].

2.2 属性利用の技術としての ID-WSF

ID-WSF は異なるサイト間で，ユーザの意思に基づき属性情報を安全に流通させるための Web サービス仕様である．Liberty Alliance 仕様として，SAML と ID-WSF は**図 2** のフレームワーク上に規定されている．

ID-WSF 仕様では安全性を保障するための仕組みとして通信プロトコルに SOAP を利用したうえで，セキュリティメカニズム仕様を規定している．通信の秘匿性とメッセージの完全性を組み合わせて定義し，セキュリティの種別を “セキュリティメカニズム ID” と呼ばれる識別子で規定している．組合せには多くの方法があるが，たとえばメッセージの完全性を確保するために，**図 3** の構造の SAML トークンを使うことにより，以下を実現できる [17].

- アサーションからユーザの特定
- 情報要求元である送信者の認証を XML 署名で検証
- 第三者機関（IdP）のアサーションに含まれる送信者の公開鍵によって送信者の承認

ID-WSF による属性利用のフローを**図 4** に示す．

通信プロトコルに SOAP を利用したうえ，セキュリティトークンの運用を考慮することにより，ID-WSF は安全性に優れた属性利用仕様を実現している．前述した OpenID でも属性交換のための仕様があるが REST（Representational State Transfer）ベースの仕様であり，SOAP のようにさまざまな WS-* 仕様で規定された追加のプロトコルによって，エンド・ツー・エンドのメッセージセキュリティをサ

[*1] SSL/TLS については脆弱性や公開鍵の共有の問題等が報告されている [14], [15]．セキュリティ対策として SSL/TLS だけでなく，改竄防止と完全性を保証するために署名を行った SOAP メッセージはより安全とされ，地域情報プラットフォーム等政府機関等で広く適用されている [16].

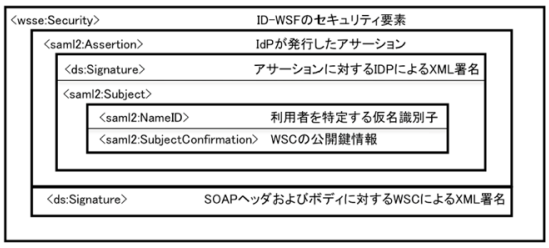
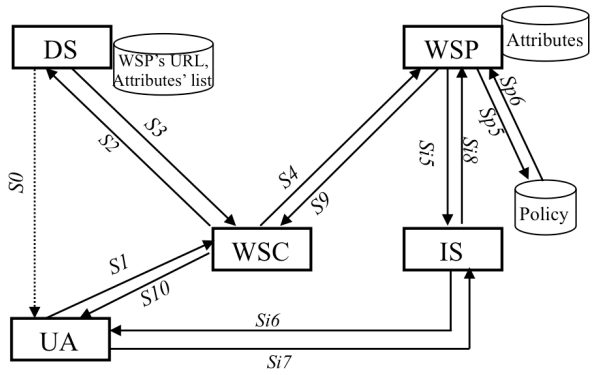


図 3 SAML トークンの構造
Fig. 3 Structure of SAML token.



S0: DSのbootstrapを含むSSOアサーションを取得
*S1:*アサーションを利用しWSCのサービスにアクセス
*S2:*WSPのロケーションを要求
*S3:*ロケーションを返却
*S4:*ユーザの属性を要求
(ポリシーによるアクセス許可判定の場合)
*Sp5:*属性へのアクセスポリシーを確認
*Sp6:*アクセス許可ポリシーの結果を返却
(インタラクション・サービスの場合)
*Si5:*ユーザに属性へのアクセス許可を依頼
*Si6:*ユーザ同意を得るためのインタラクション
*Si7:*インタラクションの結果を返却
*Si8:*アクセス許可問い合わせの結果を返却
*S9:*属性を返却
*S10:*WSCでの処理結果を返却

図 4 ID-WSF のメッセージフロー
Fig. 4 Message flow of ID-WSF.

ポートする追加仕様は存在しない．複数のクラウド間においてエンド・ツー・エンドのセキュリティを確保したうえでの属性交換は，SOAP ベースの仕様が望ましい．

ID-WSF では，特定の属性を提供するプロバイダである WSP（Web Service Provider）にユーザが事前に属性を登録し，WSP の URL を検索サイトである DS（Discovery Service）に登録する．ユーザがサービス提供サイトである WSC（Web Service Consumer）を利用するときに，DS の在り処を示すポインタが WSC に通知されることによって，WSC は WSP の URL に登録されている属性情報の項目リストを DS から入手する．

WSC は入手した項目リストにより属性を WSP に要求し，ユーザの属性情報を入手する．この際，WSP は WSC から属性要求があった場合，次の 2 通りの動作を行う．

- 事前に定めたポリシに従い属性を返す
- 属性の持ち主に可否を逐次問い合わせる（IS: Interaction Service（インタラクションサービス））

属性の管理/利用技術として，ID-WSF はさまざまな分野に適用されている．海外では ID-WSF 仕様策定後，いち早く AOL が Radio@AOL サービスを ID-WSF を用いて実現した [18]．また，Nokia では携帯端末のための ID-WSF を用いた SOA ベースのフレームワークを発表している [19], [20]．

国内では，コンテンツ視聴のための複数デバイス間の情報連携や，機微情報を扱う医療分野等，多くの研究と実績がある [21], [22], [23], [24], [25]．一方，今後国による整備が進む予定の社会保障・税番号制度においては，機微な情報を扱う医療分野で ID-WSF を適用することが検討されており，実証実験も行われている [26], [27]．

3. 既存技術のクラウド適用における問題点

連携アイデンティティ技術の中で SAML SSO はさまざまなサイトで適用可能となっている．しかし世界各地に広がるクラウド間で適用した場合，さまざまな問題が起きる可能性がある．そこで前章で述べた SAML SSO POST/Artifact Bindings を実装したサイトをクラウド上に導入し，サイト間およびシーケンスごとの経過時間を測定することにより，各サイトを分散配置した場合にどのような問題点が潜在する可能性があるかについて実験を通して考察した．

3.1 クラウド上での遅延時間の実測

最初にクラウドとして広くサービスを提供している AWS（Amazon Web Service）が世界で 7 つのリージョンで提供する IaaS サービスである Amazon EC2（Elastic Computing Cloud）において，遅延時間がどの位あるかを測定するために実験を行った．

表 1 クライアント PC，AWS EC2 リージョン間の RTT（msec, () 内は hop 数）

Table 1 RTT between client PC and each AWS EC2 regions (msec, (): number of hops).

To / From	アイルランド	サンパウロ	バージニア	東京	オレゴン	カリフォルニア	シンガポール
クライアントPC	262.257 (25)	315.358 (22)	176.226 (19)	11.294 (11)	124.768 (21)	116.191 (19)	80.260 (14)
アイルランド	0.595 (6)	209.370 (17)	94.720 (15)	284.699 (17)	174.144 (24)	157.955 (22)	458.114 (19)
サンパウロ	208.338 (17)	0.582 (10)	140.327 (18)	314.552 (21)	219.416 (21)	183.778 (20)	351.245 (21)
バージニア	95.516 (18)	150.102 (19)	0.963 (8)	189.889 (18)	98.864 (17)	83.758 (15)	260.546 (17)
東京	267.631 (21)	294.169 (18)	194.126 (19)	0.542 (6)	137.046 (20)	126.557 (18)	82.490 (14)
オレゴン	170.743 (22)	224.769 (19)	98.878 (19)	123.098 (21)	0.718 (10)	20.546 (14)	317.168 (23)
カリフォルニア	160.125 (19)	204.288 (17)	83.377 (14)	117.463 (18)	20.568 (13)	0.494 (6)	288.349 (18)
シンガポール	442.614 (20)	376.727 (20)	248.398 (18)	86.477 (16)	305.802 (19)	300.779 (16)	0.619 (6)

(1) 実験環境

利用した AWS EC2，測定に用いたクライアント PC 環境の仕様は次のとおりである．

［AWS EC2 上のサイト環境］

- リージョン：バージニア，オレゴン，カリフォルニア，アイルランド，サンパウロ，東京，シンガポール
- OS/Middleware：Fedora 15 32 bit/JDK1.6，Tomcat6，OpenAM9.5.3
- インスタンスタイプ：Small（1.7 GB memory，1 EC2 Compute Unit，160 GB storage）
- IP タイプ：14 インスタンス全部に Elastic IP を利用

［クライアント PC 環境］

- ロケーション：東京
- OS：Mac OS X 10.6.8
- Web ブラウザ：Firefox 9.0.1
- ネットワーク：Wired，100 Mbps

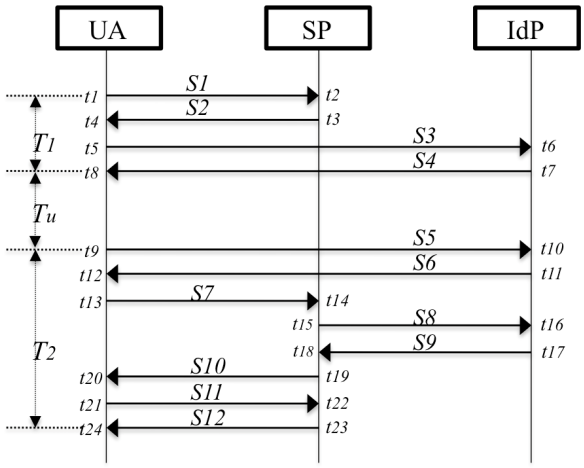
(2) 実験の方法

7 つのリージョンで IdP および SP を各々立ち上げ，計 14 個の仮想ノードを使って SAML SSO サイトを構築し，実際に SAML SSO を実行した．Web ブラウザ，IdP，SP 間の RTT を測定した．クライアント PC と 7 リージョン，および 7 リージョンどうしの計 56 通りの RTT を，traceroute コマンドを実行した．

(3) 実験の結果

測定した値を**表 1** に示す．実験の測定結果により以下が導かれた．

- クライアント PC から最も大きい RTT はサンパウロの 315.358 ms であり，東京の 11.294 ms の 30 倍近い値を示している．
- リージョン間で最も大きい RTT はアイルランド/シンガポールの 458.114 ms であり，リージョン間で最も小さい RTT を示しているオレゴン/カリフォルニアの 20.546 ms であり，20 倍以上の値を示している．



*S1:*リソースへアクセス
S2:<Authen Request>を伴いリダイレクト
S3:<Authen Request>を伴いHTTP GET
*S4:*クレデンシャル要求（ログイン画面）
*S5:*ユーザによるログイン
*S6:*SAML Artifactを伴いリダイレクト
*S7:*SAML Artifactを伴いHTTP GET
S8:<Artifact Resolve>を伴いSOAP通信
S9:<Artifact Response>を伴いSOAP通信
*S10:*リソースへリダイレクト
*S11:*リソースへアクセス
*S10:*リソースを提供

図 5 SAML SSO POST/Artifact Bindings のシーケンス
Fig. 5 Sequence of SAML SSO POST/Artifact Bindings.

- 同一リージョン内における RTT は 1 msec 未満である．

3.2 AWS EC2 上での SAML SSO における経過時間計測

次に，実際のシーケンスごとの経過時間を測定した．

(1) 実験環境

前節の環境を利用した．

(2) 実験の方法

SAML SP-Initiated SSO POST/Artifact Bindings に基づき，AWS EC2 7 リージョンに各々 IdP，SP を立ち上げた．各サイトにはシーケンス時間計測のためのツールとして tshark をインストールして利用した．シーケンスを**図 5** に示す．

実験における計測点を $t1$〜$t24$ の 24 カ所とした．ここで，

- T_1 = SP のサービスリソースにアクセスし IdP からのログイン画面が出力されるまでの時間
- T_2 = ログイン後 SP からサービスリソースが表示されるまでの時間
- T_u = ユーザがログイン入力する時間

とすると，最初に SP のサービスリソースにアクセスしてからリソース画面の表示までの時間 T は

$$T = T_1 + T_u + T_2$$

である．T_u は属人的な値であるため，当該シーケンスにおける処理時間 T_s は，$T_s = T_1 + T_2$ であり，T_s が実際のサービスのレスポンス時間となる．ゆえに T_s の大きさがユーザ体験に影響を与えると考えられる．

表 2 IdP/SP 間の経過時間
Table 2 Duration between IdP and SP.

From \ To		アイルランド	サンパウロ	バージニア	東京	オレゴン	カリフォルニア	シンガポール
アイルランド	T_1	874	989	838	674	715	426	448
	T_2	955	1,551	1,039	1,042	965	933	1,426
	T_s	1,829	2,540	1,877	1,716	1,680	1,359	1,874
サンパウロ	T_1	950	702	645	403	902	492	459
	T_2	1,459	1,114	1,267	1,190	1,096	1,121	1,353
	T_s	2,409	1,816	1,912	1,593	1,998	1,613	1,812
バージニア	T_1	655	569	407	229	447	548	539
	T_2	1,003	1,272	640	879	795	702	1,052
	T_s	1,658	1,841	1,047	1,108	1,242	1,250	1,591
東京	T_1	448	395	385	63	173	182	121
	T_2	1,466	1,498	869	199	1,763	683	412
	T_s	1,914	1,893	1,254	262	1,936	865	533
オレゴン	T_1	471	509	496	325	468	377	365
	T_2	1,229	1,291	826	560	600	534	1,072
	T_s	1,700	1,800	1,322	885	1,068	911	1,437
カリフォルニア	T_1	523	517	503	190	346	486	237
	T_2	1,066	1,325	799	440	674	660	995
	T_s	1,589	1,842	1,302	630	1,020	1,146	1,232
シンガポール	T_1	579	522	318	206	234	267	218
	T_2	1,929	1,595	1,057	442	1,009	1,035	405
	T_s	2,508	2,117	1,375	648	1,243	1,302	623

(3) 実験の結果

IdP/SP の実験による測定結果は**表 2** に示すとおり，以下の結果となった．

- T_1 が最大となるのは，アイルランド/サンパウロの 989 msec
- T_2 が最大となるのは，シンガポール/アイルランドの 1,929 msec
- T_s が最大となるのは，アイルランド/サンパウロの 2,540 msec
- T_1，T_2，T_s とも東京/東京が最小でそれぞれ 63 msec，199 msec，262 msec

T_s の最大値であるアイルランド/サンパウロの 2,540 msec は東京/東京の 262 msec の 10 倍近い値を示している．T_1 は最初にユーザがアクションを起こし，次の入力であるログインまで「待たされる」時間である．しかし最大値でも 1 秒程度である．T_2 はログイン後，サービス画面をブラウザに提供するまでの時間，つまり画面表示が始まる時間であり最大値の場合，ユーザは約 2 秒「待たされる」ことになる．そこで T_2 の最大値に注目して，アイルランド/シンガポールのセットについて東京/東京と比較し詳細に経過時間を示したのが**表 3** である．

ここで，図 5 で示される S_8 および S_9，すなわち IdP/SP 間の SOAP 通信に注目する．表 3 では，$t14$〜$t19$ が該当する．さらに Δ に注目すると，$\Delta t15 = 468$ msec，$\Delta t16 = 294$ msec，$\Delta t17 = 3$ msec となっている．TCP/IP 通信の詳細を調べると，$\Delta t15$ は [SYN] ⇒ [SYN, ACK] ⇒ [SYN] であり，TCP/IP 接続確立の時間である．また $\Delta t16$ は POST のあと [ACK] が返ってくる時間であり，両方とも大きな値を示している．それに対し，IdP が S8 のメッセージを受信し，Artifact Response を組み立て，S8 メッセージを送信するま

表 3 T1 経過時間と T2 経過時間
Table 3 Duration of T1 and T2.

$T1$	シンガポール / アイルランド		東京 / 東京	
	経過時間	Δ	経過時間	Δ
$t1$	0	0	0	0
$t2$	150	150	7	7
$t3$	152	2	13	6
$t4$	301	149	20	7
$t5$	388	87	30	10
$t6$	430	42	38	8
$t7$	537	107	56	18
$t8$	579	42	63	7

$T2$	シンガポール / アイルランド		東京 / 東京	
	経過時間	Δ	経過時間	Δ
$t9$	0	0	0	0
$t10$	41	41	8	8
$t11$	58	17	23	15
$t12$	131	73	30	7
$t13$	136	5	34	4
$t14$	286	150	42	8
$t15$	754	468	89	47
$t16$	1,048	294	89	0
$t17$	1,051	3	91	2
$t18$	1,230	179	92	1
$t19$	1,467	237	121	29
$t20$	1,615	148	127	6
$t21$	1,629	14	140	13
$t22$	1,779	150	188	48
$t23$	1,780	1	191	3
$t24$	1,929	149	199	8

での Idp 滞留時間である $\Delta t17$ は 3 msec と非常に小さい．

処理が重く時間がかかるといわれている SOAP メッセージの取扱いも，遅延時間の問題に比べれば問題が小さいということが本節の実験で明らかになった．

3.3 クラウド適用における問題点の整理

SAML トークンの利用や本人同意の下での属性転送等の確立された技術によって，ID-WSF ではセキュリティやプライバシ面を十分考慮した属性情報の取扱いが可能となっている [28], [29]．しかしクラウド適用を想定した場合，前節の実験結果によれば次の問題点がある．

(1) 遅延時間の問題

さまざまなサービスがクラウド環境上に構築されると考えられるが，必ずしも国内に閉じた環境ではなく，世界のどこかのデータセンタで稼働すると考えるべきである．その場合，サイトが世界中に散在すると，遅延時間がサービス提供において問題となると考えられる．

たとえば，内容的にはまったく同じサービスが，在り処が違うサイトから提供されるとする．図 4 で ID-WSF のメッセージのフローを示したが，WSC が WSP からサービスを提供されることとなる．東京在住者が東京のサイトが提供するサービスを利用するのと，ロンドンのサイトが提供するものとでは，ユーザ体験に差が出てくるが，距離と中継設備による遅延時間が主な原因と考えられる．ユーザ体験上遅延時間は非常に重要と考えられている[*2]．

ID-WSF においてサービスを提供する WSC は，サービスアプリケーションの挙動によって随時条件を変えながら，WSP に（特に動的）属性情報を問い合わせることも考えられる．この場合，WSC と WSP 間の通信の遅延時間を小さくすることは，ユーザ体験向上に必要と考えられるが，ID-WSF では遅延時間を小さくする技術的対策は考慮されていない．

(2) 動的属性情報の取扱いの問題

企業でのデータの信頼境界とは，扱うデータを企業が自社で管理でき信頼できるか否かを仮定する境界線のことである．クラウド環境では信頼境界を越えて，個人のプライバシを含む情報が行き来することもあり，各国のプライバシや個人情報保護に関連した法制度面での問題に直面する可能性もある [30]．静的属性情報だけでなく，動的属性情報においては，個人の生活上でのプライバシに関連する情報もより数多いと考えられるため，可能な限り自分で自分の情報を信用のおけるサイトで管理したいと考えるユーザの潜在的要求も高いと考えられる．しかしながら ID-WSF は，静的属性情報の取扱いを中心として設計されており，刻々と蓄積される動的属性情報を扱うことは考慮されていない[*3]．

4. 提案

増加し続ける動的属性を ID-WSF で取り扱うには，前節に述べたような 2 つの大きな問題を解決する必要がある．そこで動的属性を十分な RTT で扱うために，ID-WSF を拡張した新たな技術方式である，動的属性分離ローミング方式を考案した．さらに高速にローミングを行うための改良方法である動的属性一括並列転送方式を考案した．本章では 2 つの方式について説明する．

4.1 動的属性分離ローミング方式

前章で述べたように，属性を利用するサービスにおいて静的属性と動的属性の取り扱い方法が違う．サービスを提供しようとするサイトに，静的属性，動的属性各々に最適な運営ポリシを持つ別々のサイトであったほうが，サービスを提供されるユーザが好む場合も考えられる．その一方で，ユーザは十分な速さの応答を望む．そこで，静的属性と動的属性を分離し，動的属性をサービス提供サイトである WSC に近いところへローミングすることにより，高速に動的属性を利用できる方式を考案した．

(1) アーキテクチャ概要

(a) 静的属性と動的属性の分離

常時増加する動的属性を考慮するために，属性を扱うサイトである WSP を，静的属性情報を扱うサイト WSPs と動的属性情報を扱うサイト WSPd とに分離する．

静的属性と動的属性を分離することによって，たとえば静的属性はセキュリティ的に非常に強固なサイト A が運用する一方，常時増加する動的属性は，スケーラビリティとレスポンスに優れたサイト B が運用するような形態をとることも可能であり，サービス提供するサイトや属性を提供

[*2] “Web を利用するユーザは，読み込みに 3 秒以上かかると苛立ちを感じ，47%のユーザが 2 秒以内の Web ページ読み込みを期待し，Web ページの読み込み時間に 3 秒以上かかるとユーザの 40%がそのサイトを去る” との調査結果が報告されている [31].

[*3] Liberty 仕様の中には，ある時点における位置情報だけの取得を目的とした Geo Location Service 仕様は存在する [32].

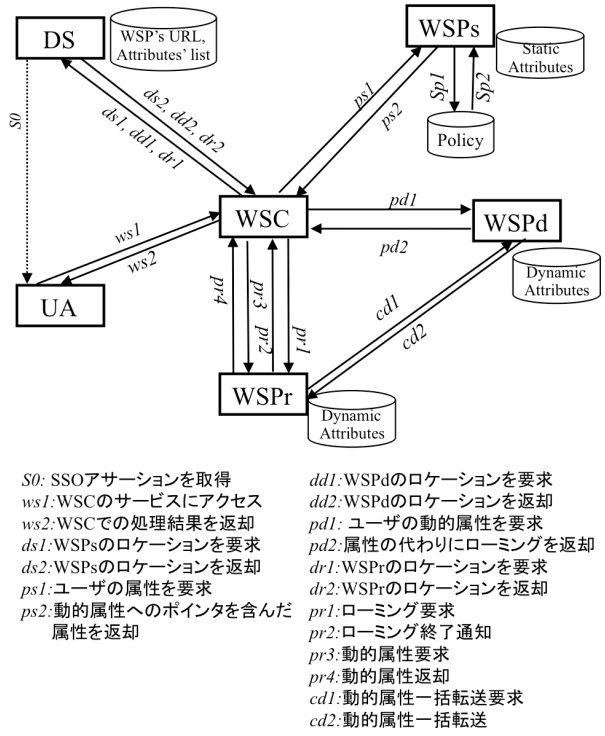


図 6 動的属性ローミング方式のフロー

Fig. 6 Message flow of dynamic attributes roaming.

するサイトにサービスアーキテクチャの柔軟性を与えることが可能となる．

本章ではモデルを単純化するために，ポリシ参照やインタラクションサービスの属性取得許可の機構についての記載は省く．

(b) 動的属性のローミング

分離した動的属性サイトである WSPd を，WSC と遅延時間の小さいサイト WSPr へローミングし，ネットワーク的に遠い場所にある場合に危惧される WSC と WSPd 間の遅延時間の問題を解決する．HTTP 通信の回数を減らすために，ローミングを行う際に動的属性を 1 つ 1 つ転送を行わず，当該属性を 1 つにまとめ一括して転送を行うことにより高速性を実現する．前項と同様，モデルを単純化するために，動的属性ローミング時におけるポリシ参照や，インタラクションサービスの属性取得許可の機構については記載を省く．

(a) の技術方式と (b) の技術方式をあわせることによって，動的属性を高速に扱うことが可能となる．アーキテクチャ概要とメッセージフローの概要および追加したメッセージフローを**図 6** に示す．動的属性呼び出しとローミングを実現するための具体的な技術方法を次に説明する．

(2) プロファイルの定義

動的属性サイトを WSC から呼び出すためにプロファイルを定義する．ID-WSF で規定している ID-SIS を拡張し，id-sis-wps（静的属性），id-sis-wpd（動的属性），id-sis-wpr（ローミング）の 3 つのプロファイルを規定する．id-sis-wps は ID-SIS Personal Profile である id-sis-pp の上位互換とする．要素 <DyamicAttributes> を id-sis-pp の XML スキーマに追加し，動的属性プロファイル名を次の形式で記述する．

```
<wps>
 <InformalName>taro</InformalName>
 <CommonName>
  <CN>Taro Tokyo</CN>
  <AnalyzedName nameScheme=" ">
   <PersonalTitle>Mr.</PersonalTitle>
   <FN>Taro</FN>
   <SN>Tokyo</SN>
  </AnalyzedName>
   ……..
 <DynamicAttributes>
  <DProfileName>urn:id-sis-wpd</DProfileName>
  ……
 </DynamicAttributes>
</wps>
```

(3) フローとメッセージ形式

WSC が動的属性を得て WSPr にローミングするまでのフローとメッセージの主なポイントは次のとおりである．図 6 におけるフローを () 内の識別子で示す．

① WSPd の在り処を得るための DS 呼び出し（*dd1*）

WSC から WSPs を呼び出し返却されたメッセージは id-sis-wps プロファイルの形式の構造となっている．このメッセージの中の要素 <DynamicAttributes> に id-sis-wpd が示されているので，WSC から DS に対し WSPd の在り処を次の形式で問い合わせる．

```
<disco:Query xmlns:disco="urn:liberty:disco:2005-11" id="discReq">
 <disco: ResourceID>https://idp-ds.com/ dca24a6f3f</disco:ResourceID>
 <disco:RequestedService>
  <disco:ServiceType>urn:id-sis-wpd:2012-06</disco:ServiceType>
  <disco:SecurityMechID>
   urn:liberty:security:2006-08:ClientTLS:SAMLV2
  </disco:SecurityMechID>
  <disco:Action>urn:id-sis-wpd:2012-06:GetAttributes</disco:Action>
 </disco:RequestedService>
</disco:Query>
```

② DS のレスポンス（*dd2*）

WSC の問合せに対し以下を返す．

```
<disco:QueryResponse disco="urn:liberty:disco:2005-11" >
 <Status code="OK"/>
 <disco:ResourceOffering entryID="1">
  <disco:ResourceID>
     https://sp.wspd.com/dda825cfef</disco:ResourceID>
  <disco:ServiceInstance>
  <disco:ServiceType>
      urn:id-sis-wpd:2012-06</disco:ServiceType>
  <disco:ProviderID>https://www.wspd.com</disco:ProviderID>
  <disco:Endpoint>https://sp.wspd.com:443/soap</disco:ProviderID>
  <wsa:Metadata>
   <disco:SecurityContext>
   <disco:SecurityMechID>
      urn:liberty:security:2006-08:ClientTLS:SAMLV2
   </disco:SecurityMechID>
   <sec:Token xmlns:sec="urn:liberty:security:2006-08" usage=.. >
    <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion">
          ….Assertion data…</sa:Assertion>
   </sec:Token>
   </disco:SecurityContext>
  </wsa:Metadata>
 </disco:ServiceInstance>
</disco:ResoueceOffering>
</disco:QueryResponse>
```

③ WSPd の呼び出し（*pd1*）

得られた ResourceID の情報により，WSC は WSPd を次の形式で呼び出す．

```
<wpd:Query xmlns:wpd=”urn:id-sis-wpd:2012-08”>
<wpd:ResourceID>https://sp.wspd.com/dda825cfef</wspd:ResouceID>
 <wpd:QueryItem itemID=”activity”>
    <wpd:Select>/wpd:WPD/wpd:Activities</wpd:Select>
 </wpd:QueryItem>
</wpd:Query>
```

④ WSPd のレスポンス（*pd2*）

WSPd はこの呼び出しを受け，動的属性を返すか，もしくはローミングへ誘導する．ローミングへ誘導する場合は以下を返す．

```
<wpd:QueryResponse xmlns:wpd=”urn:id-sis-wpd:2012-08”>
 <wpd:Status code=”roam”/>
</wpd:QueryResponse>
```

⑤ WSPr の在り処を得るための DS 呼び出し（*dr1*）

WSPd から “roam” が返されることにより，WSC はローミングの準備を開始する．ServiceType を id-sis-wpr に指定し DS に WSPr の情報に関する問合せを行う．

```
<disco:Query xmlns:disco=”urn:liberty:disco:2005-11” id=”discReq”>
 <disco: ResourceID>https://idp-ds.com/dca24a6f3f</disco:ResourceID>
 <disco:RequestedService>
  <disco:ServiceType>urn:id-sis-wpr:2012-06</disco:ServiceType>
  <disco:SecurityMechID>
    urn:liberty:security:2006-08:ClientTLS:SAMLV2
  </disco:SecurityMechID>
  <disco:Action>urn:id-sis-wpr:2012-06:GetAttributes</disco:Action>
 </disco:RequestedService>
</disco:Query>
```

⑥ DS のレスポンス（*dr2*）

DS は WSPr の ResourceID の情報を返す．

```
<disco:QueryResponse  disco=”urn:liberty:disco:2005-11” >
<Status code=”OK”/>
<disco:ResourceOffering entryID=”1”>
 <disco:ResourceID>
    https://sp.wspr.com/ec23ab3faf</disco:ResourceID>
 <disco:ServiceInstance>
  <disco:ServiceType>
      urn:id-sis-wpr:2012-06</disco:ServiceType>
  <disco:ProviderID>https://www.wspr.com</disco:ProviderID>
  <disco:Endpoint>https://sp.wspr.com:443/soap</disco:ProviderID>
  <wsa:Metadata>
   <disco:SecurityContext>
    <disco:SecurityMechID>
      urn:liberty:security:2006-08:ClientTLS:SAMLV2
    </disco:SecurityMechID>
    <sec:Token xmlns:sec=”urn:liberty:security:2006-08” usage=.. >
     <sa:Assertion xmlns:sa=”urn:oasis:names:tc:SAML:2.0:assertion”>
          ….Assertion data…</sa:Assertion>
    </sec:Token>
   </disco:SecurityContext>
  </wsa:Metadata>
 </disco:ServiceInstance>
</disco:ResoueceOffering>
</disco:QueryResponse>
```

⑦ WSPr を呼び出しローミングを開始（*pr1*）

WSPd に WSPr へのローミングを依頼するために WSC は WSPr に対して次の形式で呼び出しを行う．

```
<wpr:Query xmlns:wpd=”urn:id-sis-wpr:2012-08”>
 <wpr:ResourceID>https://sp.wspr.com/dfa8c5djaf</wspr:ResouceID>
 <wpr:QueryItem itemID=”roam”>
    <wpr:Select>/wpr:WPR/wpr:Roam</wpr:Select>
 </wpr:QueryItem>
</wpr:Query>
```

⑧ WSPr による WSPd の呼び出し（*cd1*）

WSPr は BulkRequest を指定し，WSPd を次の形式で呼び出す．

```
<wpd:Query xmlns:wpd=”urn:id-sis-wpd:2012-08”>
 <wpd:ResourceID>http://sp.wspd.com/dda825cfef</wspr:ResouceID>
 <wpd:BulkRequest>
  <wpd:QueryItem itemGroup=”activity”>
    <wpr:Select>/wpr:WPR/wpr:Activities</wpr:Select>
 </wpd:QueryItem>
 </wpd:BulkRequest>
</wpd:Query>
```

⑨ WSPd から WSPr への一括転送（*cd2*）

ローミングは WSPd から WSPr へ以下の形式での一括データ転送によって行う．

```
<wpd:QueryResponse xmlns:wpd=”urn:id-sis-wpd:2012-08”>
<Status code=”OK”/>
 <wpd:BulkRequestResponse>
  <wpd:QueryItem itemGroup=”activity”>
    <wpd:GroupEntity id=1>
     <wpd:timestamp>2012-08-12T23:21:09Z</wpd:timestamp>
     <wpd:vital_signs>
       <wpd:blood_pressure>132:82</wpd:blood_pressure>
       ……..
     </wpd:vital_signs>
     ……..
    </wpd:GroupEntity id=1>
    <wpd:GroupEntity id=2>
    ……
    </wpd:GroupEntity id=n>
  </wpd:QueryItem>
  </wpd:BulkRequestResponse>
</wpd:QueryResponse>
```

⑩ 終了処理（*pr1*）

ローミング終了時に WSC へ成功のステータスを返すことにより，WSC は WSPr を WSPd のプロキシとして認識する．その後，UA からのリクエストに対しては，WSPr はあたかも WSPd と同様に振る舞うことが可能となる．

(4) シーケンス

動的属性ローミング方式のシーケンスを**図 7** に示す．図の右に示しているフロー識別子は図 6 を参照のこと．

4.2 動的属性一括並列転送方式

前節で述べたようにネットワーク的に遠距離では遅延時間が発生する．そこで大きなデータを転送すればさらに遅延時間が大きくなることは明らかである．これは前項で提案したローミング方式においてもデータ送信に時間がかかることを意味し，SOAP 通信の 1 対 1 の送受信プログラムである場合，RTT を短くするにはプログラム単体の改良やチューニングでは難しいと考えられる．

そこで大きな動的属性をローミングする際，適度な大きさのデータブロックに分割し，複数のプロセスでデータを並列送信し，並列受信したデータブロックを再度組み立てる方式により，RTT を短くする方式を考案した．

(1) アーキテクチャ概要

呼び出し側 WSPr 側のプロセスは n 個の子プロセスを生成する．呼ばれる側の WSPd 側のプロセスも同様に n 個

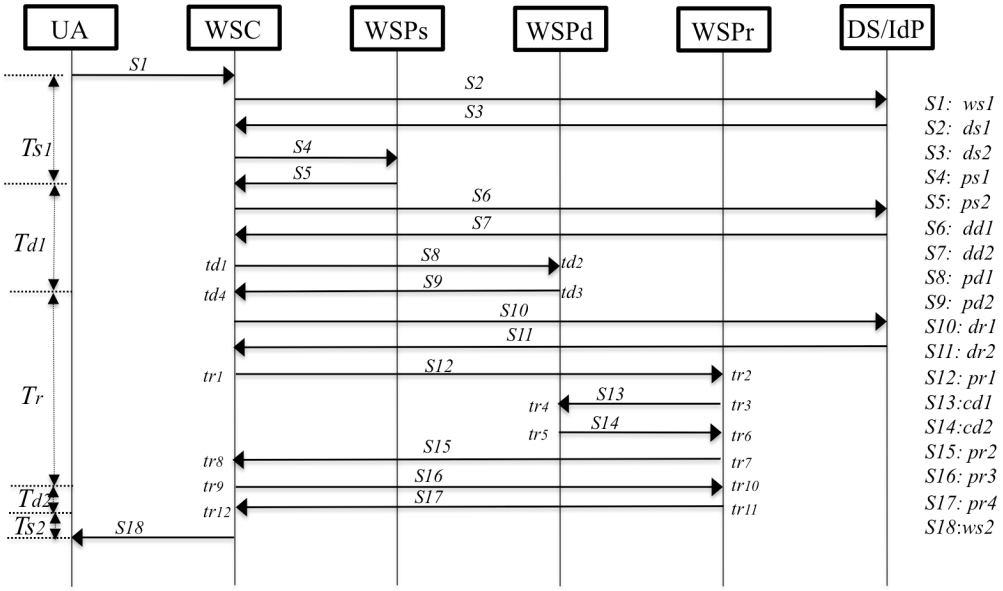


図 7 動的属性ローミング方式のシーケンス
Fig. 7 Sequence of dynamic attributes roaming method.

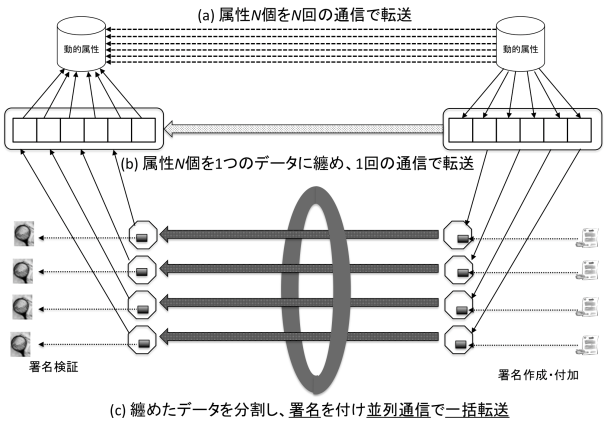


図 8 他の転送方式との比較
Fig. 8 Comparison with transfer methods.

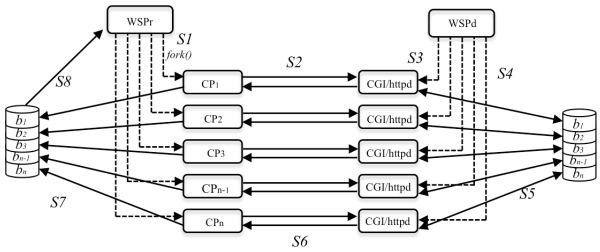


図 9 動的属性一括並列転送方式のフロー
Fig. 9 System flow of parallel transfer of batched dynamic attributes.

の子プロセスを生成する．各々の子プロセスは n 分割されたデータの指定ブロックを送受信する．通信プロトコルは HTTP を利用した SOAP 通信とする．つまり，各子プロセスが n 個に分割したデータブロックごとの SOAP メッセージの送受信を行う．

(2) セキュリティの確保

大量の動的属性の転送はセキュアにすべきである．SOAP 通信の際，以下の方法をあわせることにより，セキュアなデータ送受信を可能とする．

- SSL 通信を利用し通信経路上でのデータの暗号化
- SOAP メッセージでの XML データを XML 暗号化
- SOAP メッセージに XML 署名を行い改竄検知

(3) 他の転送方式との比較した方式概要

図 8 で動的属性一括並列転送を本稿での他の転送方式と比較している．

(a) 動的属性ローミング方式において一括転送を用いない方式．動的属性の要素ごとに HTTP GET のレスポンスメッセージに属性を載せて転送．

(b) 動的属性ローミング方式において一括転送を用いる方式．前項において説明したように要求される全要素を HTTP GET のレスポンスメッセージに一括した属性データを載せて転送．

(c) 一括並列転送方式．分割したブロックごとに署名を行い並列に転送する．転送先においては署名を検証し，受信したブロックが改竄されてないことを確認する．そして分割されたブロックを 1 つに再構成する．

(4) フロー

C 言語で実装する場合，フロー全体は以下のとおりである．**図 9** に全体図を示す．

S1：WSPr 側は子プロセス CP を n 個 fork() によって生成する．

S2：CP は HTTP GET で WSPd に n 番目の属性ブロック b_n をリクエスト．

S3：WSPd 側は httpd CGI によりプロセスを実行．

S4：別の CP から次の HTTP GET リクエストが到着した場合，別 httpd CGI を実行．n 個のプロセスが並列実

行される．

S5：各プロセスは n 番目の属性ブロックを取得．

S6：各プロセスは HTTP SOAP レスポンスにデータを埋め込み，署名を行い呼び出し元の CP へ送信．

S7：CP は受信した SOAP メッセージの署名検証後，データブロック b_n を取り出し，一時格納域に保管．

S8：すべての CP の動作が終了したら WSPr は一時領域にアクセス可能となる．

5. 評価

前章で提案したアーキテクチャの有効性を評価するために，実験を行った．3 章で行った計測の結果をふまえて，提案する 2 つの技術方式が有効であるか，さらにセキュリティ確保のための XML 署名の性能が十分実用的かについて，以下の (a)，(b) を評価した．

(a) 動的属性分離ローミング方式
ローミング機能の効果によって，分離された動的属性情報を WSC が取得する際の遅延時間が実際に小さくなるかの確認を行った．

(b) 動的属性一括並列転送方式
WSPd，WSPr が生成した n 個の子プロセスが，n 個のブロックに分割したローミングデータを n 個並列に転送する．一括並列転送により，(a) のローミング時間を短縮できることを確認した．

5.1 動的属性分離ローミング方式の実験と結果

(1) 実験のシナリオ

提案したアーキテクチャのシーケンス図 7 において，シーケンスの経過時間を属性情報全般，動的属性情報固有，ローミング固有に 3 分類し，各々 $(Ts1, Ts2)$，$(Td1, Td2)$，(Tr) としている．ローミングを行わない場合，WSC が必要とする動的属性を全部得るために S8，S9 が n 回実行されることとなる．

ここで S8，S9 の開始時間/終了時間を $td1/td2$，$td3/td4$，内部処理時間を tdp，1 回の動的属性取得時間を $Td1$，総経過時間である RTT を Tdt とすると，

$$Tdt = \sum_{i=1}^{n}(Td1)i = \sum_{i=1}^{n}(\Delta S8 + \Delta S9 + tdp)i$$
$$= \sum_{i=1}^{n}((td2 - td1) + (td4 - td3) + (td3 - td2))i$$

となる．一方，ローミングの RTT を Trt とすると，

$$Trt = \sum_{i=12}^{17}\Delta Si + \sum_{i=1}^{5}(trp)i$$
$$= (tr2 - tr1) + (tr3 - tr2) + \cdots + (tr12 - tr11)$$

である．この Tdt と Trt を実測する実験用プログラムを作成し，AWS EC2 上で実行した．

表 4 各方式の RTT 比較とローミング有効率（単位：sec，有効率：非ローミング/ローミング）

Table 4 Comparison of RTT in each method and effectiveness (msec, effectiveness: non-roaming/roaming).

	シンガポール内	シンガポール/東京			シンガポール/アイルランド		
N	非ローミング	非ローミング	ローミング	有効率	非ローミング	ローミング	有効率
1	0.011	0.426	0.835	0.51	1.067	2.513	0.42
2	0.016	0.815	0.986	0.83	2.098	2.858	0.73
3	0.025	1.211	1.003	1.21	3.192	2.874	1.11
4	0.028	1.660	1.018	1.63	4.256	2.977	1.43
5	0.040	2.066	1.077	1.92	5.335	3.166	1.69
6	0.046	2.515	1.084	2.32	6.463	3.264	1.98
7	0.054	2.768	1.144	2.42	7.442	3.436	2.17
8	0.060	3.215	1.134	2.84	8.476	3.961	2.14
9	0.071	3.614	1.172	3.08	9.514	4.203	2.26
10	0.078	3.996	1.184	3.38	10.595	4.356	2.43
20	0.144	8.150	1.492	5.46	21.969	6.600	3.33
50	0.358	20.493	1.941	10.56	54.021	6.644	8.13
100	0.723	40.733	2.749	14.82	107.388	8.274	12.98
200	1.412	N/A	3.724	N/A	N/A	10.569	N/A
500	3.899	N/A	7.522	N/A	N/A	19.447	N/A
1000	8.260	N/A	16.563	N/A	N/A	42.480	N/A

(2) 実験環境

[AWS EC2 上にサイトを設置]

以下のとおり設置した．

- サービス提供サイト WSC：シンガポール
- 属性情報提供サイト WSPd：アイルランド，東京，シンガポール
- ローミング・サイト WSPr：シンガポール
- ディスカバリサイト DS/IdP：東京

実験システムは C 言語で記述した．稼働条件は以下のとおりである．

[AWS EC2 インスタンス]

1 GB mem，CentOS5.7，Apache 2.2，MySQL5.1

[動的属性情報]

レコード長 1,024，2,048，8,192，16,384 byte

[実験プログラム]

C 言語で記述

(3) 実験方法

複数のイベントや測定時間における複数の動的属性情報を，定型長のデータレコードに入れることとした．前項に示した 4 種類の定型長レコードごとにレコード数 N を $N = 1$〜10, 20, 50, 100, 200, 500, 1,000 と変化させ，非ローミング方式の RTT である Trt とローミング方式の RTT である Tdt を計測した．WSC はシンガポール固定とし，WSPd をシンガポール，東京，アイルランド各々について 10 回実行し，平均値を取得した．

(4) 実験結果

レコード長 8,192 byte での測定結果を**表 4**, **図 10** に示す．

シンガポール内，シンガポール/東京のローミングと非ローミング，シンガポール/アイルランドのローミングと

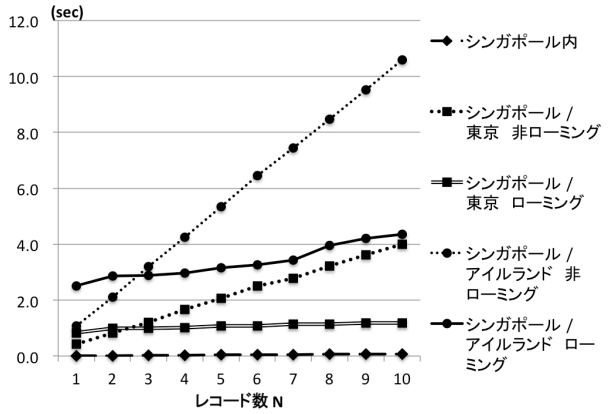


図 10 $N = 1$〜10 における各方式の RTT 比較
Fig. 10 Comparison of RTT with each methoed from $N = 1$ to 10.

非ローミングの RTT とローミングの有効率（非ローミングの RTT をローミングの RTT で除した数値）を示している．

主なポイントは次のとおりである．

- 非ローミング方式で WSC，WSPd が両方ともにシンガポールの場合，動的属性取得に要する RTT は $N = 1{,}000$ まで直線的に増加している．これは作成したプログラムが正しく稼働していることを示す．
- 非ローミング方式で WSC をシンガポール，WSPd を東京，アイルランドと変えた場合の RTT も，N が増えるに従って $N = 100$ までほぼ直線的に増加している（$N > 100$ 以上の場合，計測回ごとに大きくばらつきがあった．特に遠隔地であるアイルランドとの間ではコリジョンの影響を受けた可能性が高い）．
- 次にローミング方式を測定した．東京，アイルランドともに $N = 3$ の時点で，有効率 > 1 となり，ローミング方式が非ローミング方式よりも高速となった．また $N = 20$ では，ローミング式のシンガポール/アイルランドは非ローミング式のシンガポール/東京より高速となっている．
- ローミング方式は一括転送のため，非ローミング方式と比較すると転送データをひとまとめにするオーバヘッドがあるが，1 回の転送だけで行われるので，N が大きくなればその影響は軽微であった．
- 一括転送では N が大きい場合，1 度の送信で大きなデータを送る．$N = 1{,}000$ の場合には，$8{,}192 \times 1{,}000 = 8$ MB のデータを一括して転送したが，RTT は $N = 10$ のほぼ 10 倍であり，実験で利用した範囲のデータ量では問題は起こらなかった．

(5) 考察

本実験では，シンガポール/東京間において 100 件のレコードでの RTT は 2.79 秒であり，このサイト間でのローミング方式適用は十分に実用的であることを実証できた．一方，シンガポール/アイルランド間については，十分に速い RTT とはいえないが，100 件のレコードについて非ローミングと比較して 13 倍の RTT が得られており，高速化が機能している．実験の結果，ローミング方式は，非常に有効であることが確認された．

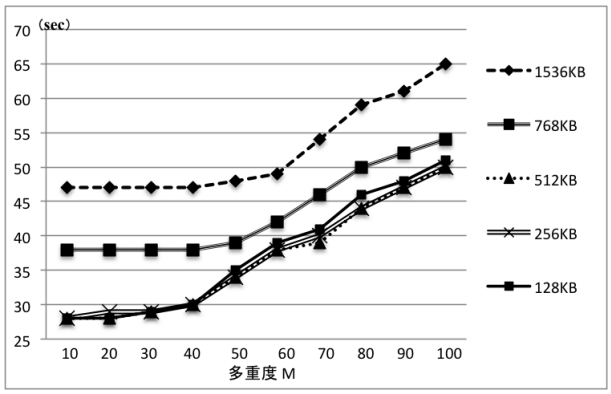


図 11 多重度 M における転送時間の比較
Fig. 11 Comparison of transfer time at multiplicity M.

5.2 動的属性一括並列転送のための実験と結果

(1) シナリオ

提案した方式では，1 つにまとめられた動的属性がある程度の大きさのデータになった場合，WSPd は複数のデータブロックに分割し SOAP メッセージを組み立て，WSPr にデータ転送する．したがって，クラウド環境である AWS EC2 上でデータの並列転送が時間短縮に有効に機能するか，実験プログラムにより確認を行った．

(2) 実験環境

分割並列転送と署名検証の 2 つの実験環境を準備した．

［AWS EC2 上のサイト］

以下のとおり設置した．

- WSPd 側を想定：東京
- WSPr 側を想定：シンガポール

稼働条件は以下のとおりである．

［AWS EC2 インスタンス］

1.7 GB mem (High-CPU Medium Instance), CentOS6.3, Apache 2.2, MySQL5.5

［転送対象のデータ］

次の大きさのデータブロックを準備

128 KB，256 KB，512 KB，768 KB，1,536 KB

(3) 実験方法

WSPr および WSPd は各々 M 個の子プロセスを生成し，子プロセス間で HTTP による多重度 M でデータブロックの並列転送を行う．実験プログラムは 4.2 節で述べたフローに基づき，C 言語で作成した．実験プログラムを各々について 10 回実行し，平均値を取得した．

(4) 実験結果

ブロックサイズごとに多重度 M で並列転送を行った．**図 11** は所要時間を示し，**図 12** は単位時間あたりの転送

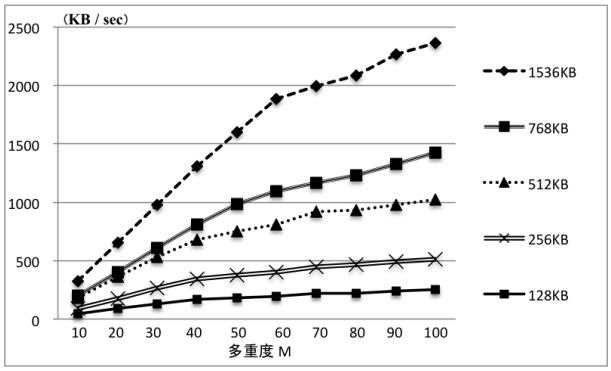


図 12 多重度 M における転送量の比較
Fig. 12 Comparison of transfer bandwidth at multiplicity M.

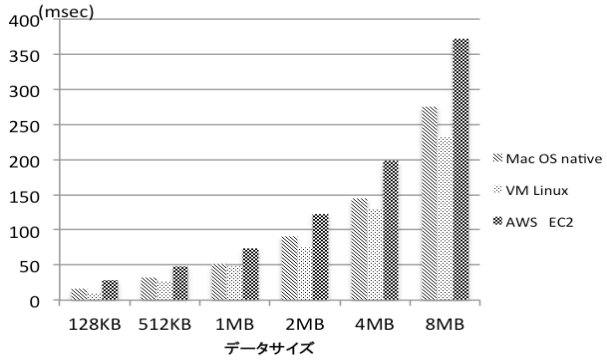


図 13 XML 署名の処理時間比較
Fig. 13 Comparison of processing time to attach XML signature.

量を示している．

同時に vmstat 等の OS コマンドでサーバのリソースの消費量を確認した．呼び出しを行う側の WSPr はほとんどリソースを消費せず，$M = 100$ においても CPU Idle 率は平均 90%を超えていた．

一方，呼び出される側で送信を行う WSPd では，$M = 50$ において，CPU idle 率は 70%程度であった．$M = 100$ においては 30%程度となっていたが，メモリや CPU 利用率等はまだ余裕があったといえる．

(5) 考察

データブロックの大きさごとに多少ばらつきはあるが，実験結果より次が導かれた．

- 多重度 40 程度までは，各ブロックにおいて所要時間に大きな変化はない．
- ブロックの大きさに応じて単位時間あたりの転送量が増加している．多重度 40 まではほぼ直線的に転送量が増加している．
- 多重度 1 から 40 程度までは，並列転送が直線的に非常に有効に機能している．

実験により多重度 40 までは並列転送が有効に機能していることが明らかになったが，これは実験環境で利用した AWS EC2 は 40 多重までの並列転送において，帯域の余裕があることを示している．

40 多重を超えたあたりから転送時間から見ると転送の効率性が落ち始めている．これは AWS がユーザごとに割り当てた帯域の上限近くまで本実行プログラムが使っていて帯域に余裕がなくなってきていると想定される．

その一方で，実験範囲のデータ量においては，転送時間はあくまで直線的に増えており，急激な悪化は見られない．AWS の環境では帯域の割当が非常に効率的に行われていると想定できる．

AWS 環境下において，動的属性のデータ量が大きくなった場合においても，提案方式である一括並列転送方式は有効に機能すると考えられる．

5.3 署名処理の実験と測定結果

(1) 実験のシナリオ

分割並列転送する際には，セキュリティ確保のため SSL によるチャネルセキュリティ確保だけでなく，メッセージを暗号化し署名を行うことが望ましい．そこで AWS EC2 の上で処理性能が十分に確保できるかの実験を行うこととし，暗号化処理と署名処理のうち，今回は署名処理を実験対象とし，署名作成，検証の性能がオンプレミスでの性能と比較して遜色ないか検証を行った．さまざまな OS 環境で利用できる XML 署名モジュールとして，XML Security Library を利用した．

(2) 実験環境

AWS EC2 シンガポール上の環境と 2 つのローカル PC 環境の計 3 種類の実験環境を準備した．

［AWS EC2 上のサイト］

1.7 GB mem （High-CPU Medium Instance），CentOS6.3，Apache 2.2，MySQL5.5

［ローカル PC 環境］

Native：Core i7 2.2 GHz (4 core)，16 GB mem，MacOS 10.7

VM（Virtual Box4.2.4）：4 GB mem，CentOS6.3

［XML Security Library］

xmlsec1 1.2.18（SHA256，RSA2048 で利用）

［XML データ］

100 KB，500 KB，1 MB，2 MB，4 MB，8 MB

(3) 実験方法

XML 署名に関し，AWS EC2 上の環境と 2 つのオンプレミス環境の計 3 種類の環境で，大きさの違う XML データに関し，署名および署名の検証の性能測定を行った．

(4) 実験結果

図 13 が XML 署名，**図 14** が XML 署名検証の処理の時間である．主なポイントは次のとおりである．

- AWS EC2 のインスタンス種別の中でも比較的に高速なタイプ（High-CPU Medium Instance）を利用したが，ローカル PC と比較すると 20～30%程度低速である．

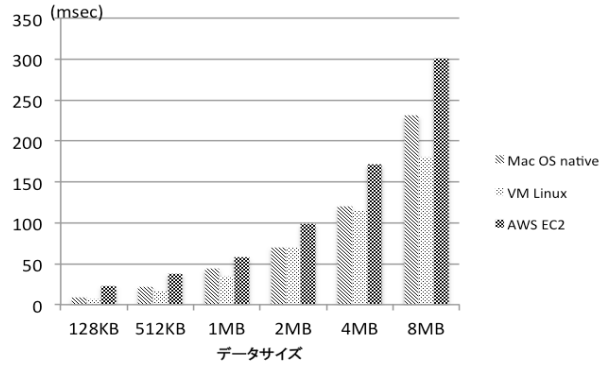


図 14 XML 署名検証の処理時間
Fig. 14 Comparison of processing time to verify XML signature.

表 5 各方式による効果
Table 5 Effectiveness by each method.

評価・提案した方式と技術	得られる効果
静的属性と動的属性の分離	柔軟性
動的属性ローミング方式	高速性（サービスへの応答）
並列一括転送方式	高速性（ローミングデータ）
暗号化，XML 署名	安全性

- 1 MB の XML 署名は 74 msec，署名検証は 58 msec となっている．2 MB の場合は，XML 署名が 123 msec，署名検証が 99 msec となっている．十分に実用的な処理時間と考えられる．
- ローカル環境では，仮想環境上で稼働しているゲスト OS Linux インスタンスの方が，ホストの MacOS 上の処理より高速である．CPU インテンシブな処理であるため，xmlsec は MacOS より Linux の方が高速性を発揮できる実装であると判断できる．

(5) 考察

マルチテナント方式のクラウド上でのプログラム実行は，物理環境を明示的に占有できない場合，自分以外のユーザがどんな処理をしているかを予測できない場合が多く，しばしば処理の実行が遅くなることもある．

しかしながら，AWS EC2 上で CPU 能力を必要とする本実験において，ローカル PC と遜色のない十分な処理速度を得ることができた．動的属性一括並列転送の SOAP 通信において，XML 署名は実用的であるといえる．

6. 評価と提案方式のまとめ

前章で提案する動的属性ローミング方式，高速化の手法である並列一括転送方式，クラウド上での署名処理性能の評価を行った．各項において考察したとおり，いずれも有効に機能し実用的であると考えられる．各方式で実現している技術がサービスに与える効果は，**表 5** にまとめられる．

- 静的属性と動的属性の分離：イベントごとに追加される動的属性は，静的属性と比較して大量になる場合があることが予測され，両属性を分離することにより大量処理も可能とする柔軟性を確保できた．
- 動的属性ローミング方式：ネットワーク的に遠方に動的属性提供サイトがある場合，サービス提供サイトにネットワーク的に近いサイトに動的属性をローミングすることにより，サービス提供サイトはユーザへの応答を高速に行うことが可能となる．
- 並列一括転送方式：動的属性が大量の場合，動的属性をブロックに分割し並列に一括転送することにより，AWS の帯域を有効に利用し，高速なローミングが可能となる．
- 暗号化，XML 署名：メッセージレベルでの安全性を確保するための暗号化，XML 署名を行う．AWS 環境での実験で，XML 署名処理を高速に行うことが可能であった．

今後増加すると考えられる属性を利用するサービスは，世界中のさまざまなクラウドで実装されると考えられる．その一方で，機器の技術革新でネットワークが高速化されたとしても，通信速度は光の速さ以下であり，ネットワークの遠さに影響による遅延時間の問題は避けて通れない．柔軟性，高速性，安全性を兼ね備えた属性利用サービスを実現するために，本稿で提案した方式をあわせて適用することは有用である．

7. おわりに

クラウド環境上でのサービスにおいては，静的属性だけでなく動的属性を扱うサービスが今後増えると考えられるが，大量の動的属性情報を司るサービスでは，サービス提供サイトと動的属性提供サイトの間の遅延時間が非常に重要となると考えられる．本稿ではクラウド環境を想定し，SAML/ID-WSF を拡張した新たな技術方式を提案した．静的属性と動的属性を分離することにより，サービス提供サイトに柔軟性を持たせた．動的属性情報をサービス提供サイトとの間の遅延時間の低いサイトにローミングし，RTT を短くすることにより高速性を実現し，ユーザ体験を向上させることを可能にした．また，ローミング時にデータを分割並列転送することにより，大量の属性情報であっても高速にローミングを行い，さらに転送時の SOAP メッセージを暗号化，署名を行うことにより安全性も兼ね備える方式とした．実験の結果から得た定量的データのより深い分析，さらなる実験と検証を進め，本方式を改良しながらさらなる実装に反映していく．

参考文献

[1] OASIS Security Services (SAML) TC, available from ⟨http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security⟩.
[2] OpenID Authentication 2.0 – Final, available from ⟨http://openid.net/specs/openid-authentication-2_0.html⟩.

[3] The OAuth 1.0 Protocol, available from ⟨http://tools.ietf.org/html/rfc5849⟩.
[4] Liberty Alliance Project White Paper, available from ⟨http://www.projectliberty.org/liberty/content/download/387/2720/file/Liberty_Federated_Social_Identity.pdf⟩.
[5] Internet2 Shibboleth Project, available from ⟨http://www.internet2.edu/shibboleth⟩.
[6] Liberty Alliance ID-WSF 2.0 Specifications including Errata v1.0 Updates, available from ⟨http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_wsf_2_0_specifications_including_errata_v1_0_updates/⟩.
[7] 下江達二：アイデンティティ管理技術の進展と変遷（〈特集〉Web アイデンティティと AI），人工知能学会誌，Vol.24, No.4, pp.504–511, 社団法人人工知能学会.
[8] Takeda, Y., Kondo, S., Kitayama, Y., Torato, M. and Motegi, T.: Avoidance of Performance Bottlenecks Caused by HTTP Redirect in Identity Management Protocols, *DIM '06*, *Proc. 2nd ACM Workshop on Digital Identity Management*, ACM (2006).
[9] Tran, T. and Wietfeld, C.: Approaches for Optimizing the Performance of a Mobile SAML-based Emergency Response System, *Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, 13th*, IEEE (2009).
[10] 千葉昌幸，漆島賢二，前田陽二：属性情報プロバイダ：安全な個人属性の活用基盤の提言，情報処理学会論文誌，Vol.47, No.3, pp.676–685 (2006).
[11] 畠山　誠：異なる連携プロトコルを仲介するプロキシ型属性情報管理システム，情報処理学会創立 50 周年記念（第 72 回）全国大会，5F-1 (2010).
[12] Cuddy, S., Katchabaw, M. and Lutfiyya, H.: Context-Aware Service Selection Based on Dynamic and Static Service Attributes, Wireless And Mobile Computing, *IEEE International Conference on Networking And Communications* (*WiMob 2005*), Vol.4, IEEE (2005).
[13] Security Assertion Markup Language (SAML) V2.0 Technical Overview, OASIS (25 Mar. 2008).
[14] 須賀祐治：SSL/TLS renegotiation 機能の脆弱性に伴なう移行における問題点, IPSJ SIG Notes 2010-CSEC-50(12), 1-4, 2010-06-24 (2010).
[15] Heninger, N., Durumeric, Z., Wustrow, E. and Halderman, J.A.: Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices, *USENIX Security '12* (2012).
[16] 財団法人全国地域情報化推進協会：地域情報プラットフォームガイドライン技術解説要約，V2.5 (2012).
[17] 菅野　淳：ID-WSF2.0 を利用したセキュアな情報流通，カンターラ・イニシアチブ・セミナー 2011 (2011).
[18] Liberty Alliance Project の技術と活動，XML コンソーシアム・セミナー (Sep. 2005).
[19] Nokia Web Services Framework for Devices – A Service-oriented Architecture, NOKIA CORPORATION.
[20] Web Services Framework API: Using the API, available from ⟨http://www.developer.nokia.com/document/Cpp_Developers_Library/GUID-759FBC7F-5384-4487-8457-A8D4B76F6AA6/html/Web_Services_Framework_API4.html⟩.
[21] 藤井亜里砂，石川清彦，森住俊美，菊池由美，山田智一，河盛雅仁，川添雄彦：複数デバイス間での認証情報連携によるシームレスなコンテンツ視聴サービス，社団法人映像情報メディア学会技術報告 (Sep. 2008).
[22] 爰川和宏，宮島麻美，大野　浩，中村　亨，前田裕二：医療・健康情報の流通・活用に向けた情報連携基盤の提案，情報処理学会研究報告，Vol.2009-DPS-141, No.14 (2009).
[23] 堀川桂太郎：コンシューマ向け ID 連携サービスの構築・運用の実際とその戦略性, 信学技報, IN2009-117 (2010-1), 電子情報通信学会 (2009).
[24] 山村千草，藤井亜里砂，石川清彦，本間祐次，小尾高史，谷内田益義，李　中淳：放送を起点とした個人向け通信サービス利用におけるユーザー機器認証フレームワーク，FIT2010（第 9 回情報科学技術フォーラム），L-035 (2010).
[25] Hatakeyama, M. and Shima, S.: Privilege Federation between Different User Profiles for Service Federation, *DIM '08*, *Proc. 4th ACM Workshop on Digital Identity Management*, ACM (2008).
[26] 厚生労働省：「医療等分野における情報の利活用と保護のための環境整備のあり方に関する報告書」（案），第 9 回社会保障分野サブワーキンググループ及び医療機関等における個人情報保護のあり方に関する検討会の合同開催，入手先 ⟨http://www.mhlw.go.jp/stf/shingi/2r9852000002gdlt-att/2r9852000002gdqj.pdf⟩.
[27] 富士通（株）：社会保障情報基盤システム，入手先 ⟨http://pr.fujitsu.com/jp/news/2010/03/25-1/index.html⟩.
[28] Liberty Alliance Liberty ID-WSF Security Mechanisms Core, available from ⟨http://www.projectliberty.org/liberty/content/download/3478/23060/file/liberty-idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf⟩.
[29] ID-WSF 2.0 SecMech SAML Profile, available from ⟨http://projectliberty.org/liberty/content/download/894/6258/file/liberty-idwsf-security-mechanisms-saml-profile-v2.0.pdf⟩.
[30] 下道高志：クラウドコンピューティングの現状と欧米におけるプライバシーへの取組み（《特集　ネット検索サービス事業の諸問題》），法とコンピュータ学会誌，No.28, pp.119–125, 法とコンピュータ学会 (2010).
[31] Forrester Research: eCommerce Web Site Performance Today – An Updated Look At Consumer Reaction To A Poor Online Shopping Experience (Aug. 2009), available from ⟨http://www.damcogroup.com/white-papers/ecommerce_website_perf_wp.pdf⟩.
[32] Liberty ID-SIS Geolocation Service Specification, available from ⟨http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_sis_1_0_specifications/⟩.

下道 高志　（正会員）

慶應義塾大学卒業．東京電機大学大学院後期博士課程在学中．AT&T ベル研究所と共同で UNIX 国際機能を開発．サン・マイクロシステムズにて Java，アイデンティティ技術の仕様策定・適用，クラウド API の仕様策定等に従事．2010 年より日本オラクル（株）に移籍．官民におけるアイデンティティ関連技術の適用実装に従事．警察庁総合セキュリティ対策会議委員，IPA Ruby 標準化ワーキンググループ委員，総務省スマートクラウド研究会技術ワーキンググループ構成員，ISO SC27/WG5 エキスパート等を歴任．ISACA 東京支部教育委員会委員長．CISA, CISM.

佐々木 良一 （フェロー）

1971 年 3 月東京大学卒業．同年 4 月日立製作所入社．システム開発研究所にてシステム高信頼化技術，セキュリティ技術，ネットワーク管理システム等の研究開発に従事．2001 年 4 月より東京電機大学工学部教授，2007 年 4 月より未来科学部教授．工学博士（東京大学）．1998 年電気学会著作賞受賞．2002 年情報処理学会論文賞受賞．2007 年総務大臣表彰等．著書に，『IT リスクの考え方』岩波新書 2008 年等．日本セキュリティ・マネジメント学会会長，内閣官房情報セキュリティセンター情報セキュリティ補佐官．