

# イベント駆動型 XML パーサーを用いた WebSphere Application Server の WS-Security の パフォーマンス改善

野ヶ山 尊秀 高瀬 俊郎 上野 憲一郎

## Performance Improvement of Web Services Security in a WebSphere Application Server Using an Event Driven XML Parser

Takahide Nogayama, Toshiro Takase and Kenichiro Ueno

WS-Security は Web サービスにセキュリティを適用する仕様であるが、この WS-Security を適用した場合に性能が著しく劣化する課題がある。これは、暗号化、復号化、電子署名に関する処理のほかに、XML 文書に関するさまざまな処理が増えるためである。特に XML 文書を格納するための木構造データを作成する処理の負荷が大きい。本論文では WS-Security の処理をイベント駆動型 XML パーサーのイベント上でいい、木構造データの作成を避けることで性能改善を図った。この提案手法を WAS の WS-Security 実装に応用し、性能改善を確認した。

WS-Security is known as a specification to apply security for web services. But if we apply WS-Security to web services, then the performance of the web services is significantly decreased not only because of encryption decode and digital signatures but also extra costs for XML related processing. In particular, the computational cost for creating tree structured data for XML documents is very expensive. This paper proposes WS-Security processors which work on XML parser events. They enable us to avoid creating a tree structured data for XML documents. We applied this approach to WS-Security in WAS and confirmed that our approach enhanced WS-Security performance.

Key Words & Phrases : XML, Web サービス, WS-Security, 高速化, イベント駆動型 XML パーサー  
XML, Web Services, WS-Security, Performance Optimization, Event  
Driven XML Parser

### 1. はじめに

Web サービスは、コンピューター間に相互運用性を持つ通信を提供するソフトウェア・システムのことを指す。さまざまな異なるプラットフォーム間の通信に適しているため、普及が進んでいる。標準的な Web サービスの仕様として Java API for XML-based Web Services (JAX-WS) 2.0 [1] が一般に普及している。JAX-WS 2.0 は、SOAP プロトコル [2] により Extensible Markup Language (XML) [3] 文書を通信するための仕様である。

Web Services Security (WS-Security) [4] は Web サービスにセキュリティを適用するプロトコルである。XML デジタル署名 [5] によるメッセージの署名機能、XML 暗号化 [6] によるメッセージの暗号化機能、メッ

セージに鍵情報や認可情報を運ぶセキュリティ・トークンを付加する機能が規定されている。

一方、セキュアな通信プロトコルとして、ネットワーク層での保護を行う Security Architecture for Internet Protocol (IPSec) [7]、トランスポート層での保護を行う Secure Socket Layer (SSL) [8] や、Transport Layer Security (TLS) [9] や、Secure Shell (SSH) [10] などのプロトコルが知られている。これらのプロトコルはネットワーク層やトランスポート層の通信路のセキュリティを保証しているが、アプリケーション層の通信路のセキュリティは保証していない。

例えば、あるユーザーがある企業のデータセンター内のサーバーに SSL/TLS を用いて通信する場合を考える。多くの場合、SSL/TLS 接続はユーザーのコンピュー

提出日:2008年5月12日 再提出日:2009年6月1日

ターとゲートウェイ間の通信路を暗号化するが、ゲートウェイで一度復号化され、平文による通信としてサーバーに接続される。もしデータセンター内に攻撃者がいた場合、この通信を容易に盗み見ることが可能となる。

このように、データセンター内のネットワークに攻撃者がいる可能性を認めた場合、アプリケーション層でのセキュリティが必要となる。また複数の経路をたどるような複雑な通信においては、データの選択的な保護が必要となる。WS-Securityはこのようなセキュリティ要件を可能とする。

ITビジネスの普及とともに、このような複雑なセキュリティを要求する環境が急増している。戦略的アウトソーシングやビジネス・プロセス・アウトソーシングなどの外部委託や、クラウド・コンピューティングなどの普及により、これまで一カ所で管理していたデータが複数の異なる場所に分散し始めている。すなわち、自社のデータを他社のデータセンターやクラウド内に配置することや、自社のデータセンターやクラウド内に他社のアプリケーションが配置されることが一般的に行われるようになった。

このような高い要望に反して、WS-SecurityはSSL/TLSに比べ普及していない。この理由は主に2つ考えられる。1つは設定の複雑さである。WS-Securityはさまざまな設定が可能な反面、設定項目が複雑になる。もう1つは性能劣化である。この論文の狙いは、WS-Securityの普及のために性能劣化を解消することである。

一般に通信路にセキュリティを適用すると性能が劣化する。すなわちスループットの低下や応答時間の増加を招く。WS-SecurityをWebサービスに適用すると性能が著しく劣化する。SSL/TLSを適用した場合に比べて顕著に劣化するため、WS-Securityの普及を妨げる一つの大きな要因となっている。

SSL/TLSと比べWS-Securityの性能が劣化する主な要因は、XMLに関連する処理が増えているためである。SSL/TLSの場合は、データ部分のバイト列に対して暗号化、復号化、ハッシュ計算などを行えばよい。しかしWS-Securityの場合はデータの内容がXML文書のため、暗号化されたデータを取り出すために木構造データを作成し探索する処理が必要になる。一般に木構造データはメモリー使用量が大きいいため、そのメモリー開放が劣化要因となることもある。

木構造データの作成を高速に行う方法として、牧野らは、差分XMLパーズングを用いて木構造データの作成を減らす手法[11]により性能改善を図っている。寺口らは、牧野らの手法を改良し差分XMLパーズングのメモリー使用量を削減する手法[12]により、さらなる性

能改善を図っている。これらの手法は最終的には木構造データを作成するため、メモリー使用量の問題は残る。

木構造データを作成しない方法として、牧野らはイベント駆動型APIの一つであるSimple API for XML (SAX) [16]を用いてストリーム型のWS-Securityを実装し、木構造データを作成せずにWS-Securityの処理を行う手法[13]により性能改善を図っている。この手法は、Webサービス・アプリケーションもSAXを用いた実装に変える必要がある。しかしWebサービス仕様であるJAX-WS 2.0では、Webサービス・アプリケーションはJava Architecture for XML Binding (JAXB) 2.0 [14]オブジェクトを処理すると規定しているため、SAXイベントを一度シリアライズし、JAXBオブジェクトを作成するコンポーネントに入力することになり、同じXMLに対して2度XMLパーズングを行うことになってしまう。

本論文ではイベント駆動型APIであるStreaming API for XML (StAX) [17]上で稼働するストリーム型のWS-Securityを実装し、さらにJAXBオブジェクトの作成までの一連のストリームをつなげる方法を提案する。これにより木構造データを作成せずにWS-Securityの処理を実行できる。

本論文ではまず背景として、一般的なXMLの知識とXMLの処理方法について述べる。次にWebサービスとWS-Securityについて述べ、その課題点を指摘する。最後に筆者らが提案する手法を述べ、性能評価実験によってその有効性を示した。また、提案手法をWebSphere® Application Server (WAS)のWS-Security実装に組み込み、エンド・ツー・エンドでの性能評価実験を行い、有効性を確認した。

## 2. Webサービス・エンジンのXML文書の処理

### 2.1 2種類のXMLパーサーAPI(木構造データ型APIとイベント駆動型API)

XMLを木構造データとして読み込むAPIとしてDocument Object Model (DOM) [15]が知られている。一方イベント駆動型APIとしてSAXやStAXが知られている。

木構造データ型APIの利点は、もともと木構造であるXML文書を直感的に扱えることである。また部分木の削除や挿入を容易に行うことができる。しかしXML文書全体をメモリー上に読み込んでしまうため、動作が遅い。

イベント駆動型APIの場合、XML文書を一連のイベントとして処理し、処理し終わったイベントは破棄される。

XML 文書全体をメモリーに読み込むことがないため、木構造データ型 API よりも高速である。またイベントを次々とイベント処理プログラムに渡すことで、簡単に並行処理プログラムを記述することができる。しかしイベントが過ぎ去ってしまった箇所を再び読みたい場合、再び XML 文書の先頭から読み込まなければならない。

イベント駆動型 API にはプッシュ型とプル型の2つのタイプがある。プッシュ型は XML パーサーがイベント処理プログラムに一方的にイベントを送る。プル型はイベント処理プログラムが XML パーサーにイベントのリクエストを行い、イベントを受け取る。このような特徴のため、プッシュ型 API は XML 文書全体をイベントに変換してしまうが、プル型は任意の場所でイベントの作成を止めておくことが可能である。

木構造データ型 API とイベント駆動型 API の両方の特徴を備えた API として、JDOM [18], dom4j [19], XOM [20], Axis Object Model (AXIOM) [21] などが知られている。イベント駆動型 XML パーサーを木構造データ作成プログラムで包むことで実現している。

## 2.2 2種類のXMLパーサーAPIを活用できるデータ構造 AXIOM

WebSphere Application Server 6.1 Web Services Feature Pack (WAS 6.1 FP) 以降の Web サービス・エンジンは、AXIOM を通じてイベント駆動型 API と木構造データ型 API の両方で XML 文書にアクセスすることができる。

AXIOM では、XML パーサーとして StAX パーサーを使用し、木構造データ AXIOM の作成に StAX ビルダーを使用する。これらの利用形態について図 1 に示した。

AXIOM は、遅延評価を用いて、XML パージングと木構造データの作成を遅らせ、必要になったときに必要な処理のみを行う。まず AXIOM のルート要素のみを作成し、AXIOM を扱うプログラムが子要素を取得しようとしたときに初めて StAX パーサーにイベントの要求を行い、子要素に対応する AXIOM を作成する (図 1 の a)。このため、プログラムから要求されなかった個所の構

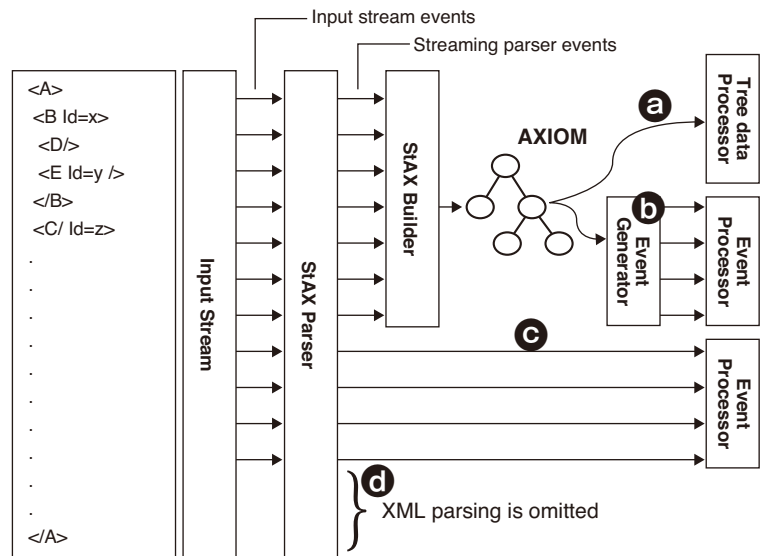


図1. AXIOMによるさまざまなXML文書の利用形態

文解析と木構造データの作成を省くことができる (図 1 の d)。

高速化やメモリー節約のため、AXIOM の作成を省き、イベント駆動型プログラムを用いて直接 StAX パーサーからの StAX イベントを処理すること (図 1 の c) ができる。ただしこの場合は StAX イベントが破棄されてしまうため、2 回以上アクセスする必要がある場合は利用できない。

また、AXIOM から StAX イベントを作成し、イベント駆動型プログラムで処理することが可能である (図 1 の b)。この StAX イベントは、StAX パーサーが生成する StAX イベントと同じである。この機能により、AXIOM が作成されてしまった要素に対して、イベント駆動型プログラムで処理することが可能となる。

## 3. WebサービスとWS-Security

### 3.1 イベント駆動型APIを利用したWeb サービスの処理

JAX-WS 2.0 は Java 上で実現する Web サービスの

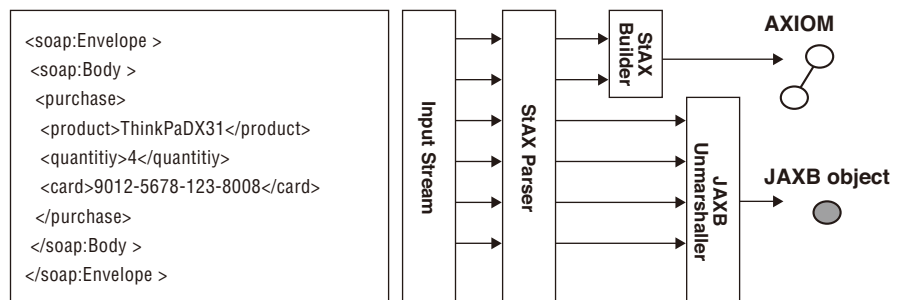


図2. イベント駆動型APIを利用したWebサービスの受信側での処理

仕様について定めている。JAX-WS 2.0 では、JAXB による XML データ・バイnding を用いて、XML 文書を JAXB オブジェクトに変換してから Web サービス・アプリケーションに渡すように規定している。

AXIOM による Web サービスの受信側での XML の処理の概要を図 2 に示した。SOAP Envelope と SOAP Body の AXIOM だけが作成され、SOAP Body の子要素の StAX イベントは JAXB Unmarshaller によって処理されたのち破棄され、JAXB オブジェクトが作成される。

### 3.2 木構造データ型APIを利用したWS-Securityの処理

Web サービス送信側で典型的な WS-Security 設定 (SOAP Body に XML 電子署名を施したのち、SOAP Body の子要素が XML 暗号化により暗号化される) が適用されたメッセージを処理する Web サービスの受信側の概要を図 3 に示した。

Web サービスの受信側では、まず XML 暗号により暗号化された SOAP Body の子要素を復号する。SOAP Body の子要素は <xenc:EncryptedData> であり、この要素は暗号化されたデータを保持している。まず Decryption Handler が <xenc:EncryptedData> に格納された SOAP Body から暗号化されたデータを取り出し、復号を行う。復号化された値は XML の断片のバイト列となっている、この断片は StAX パーサーと StAX ビルダーによって AXIOM に変換され、<xenc:EncryptedData> と入れ替えられる。

次に SOAP Body に対して付与されている XML 電子署名の検証を行う。まず SOAP Body 全体の AXIOM

を走査して Exclusive XML Canonicalization (XML C14N) [22] による XML 正規化を行う。正規化された SOAP Body 要素から digest 値を計算し、送信側が添付した digest 値との比較を行い署名の検証が終わる。

最後に、JAXB オブジェクトを作成し、Web サービス・アプリケーションに渡す。復号された SOAP Body の子要素の AXIOM から StAX イベントを生成し、JAXB Unmarshaller がそのイベントを処理して JAXB オブジェクトが作成される。

### 3.3 木構造データ型APIを利用したWS-Securityの課題

一般にセキュリティを適用した場合、必ず暗号化、復号化、電子署名の添付、電子署名の検証の処理は増える。しかし WS-Security ではそれ以外に XML に関連した処理が増える。

WS-Security を適用していない Web サービスの実装の場合、平文の StAX イベントがそのまま JAXB Unmarshaller に入力されている (図 2)。しかし WS-Security を適用した場合は平文の StAX イベントは AXIOM に変換され、StAX イベントを再び作成して JAXB Unmarshaller に入力している。このため、StAX イベントから AXIOM を作成する処理と AXIOM から StAX イベントを作成する処理が余分に必要となる。また、暗号化されたデータを含む要素 <xenc:EncryptedData> の AXIOM を作成する処理も余分に必要となる。

また、木構造データ上で行う XML C14N では、名前空間の範囲の確認を何度も行うことになるが、その度に先祖の要素をたどって XML 名前宣言 [23] を探す

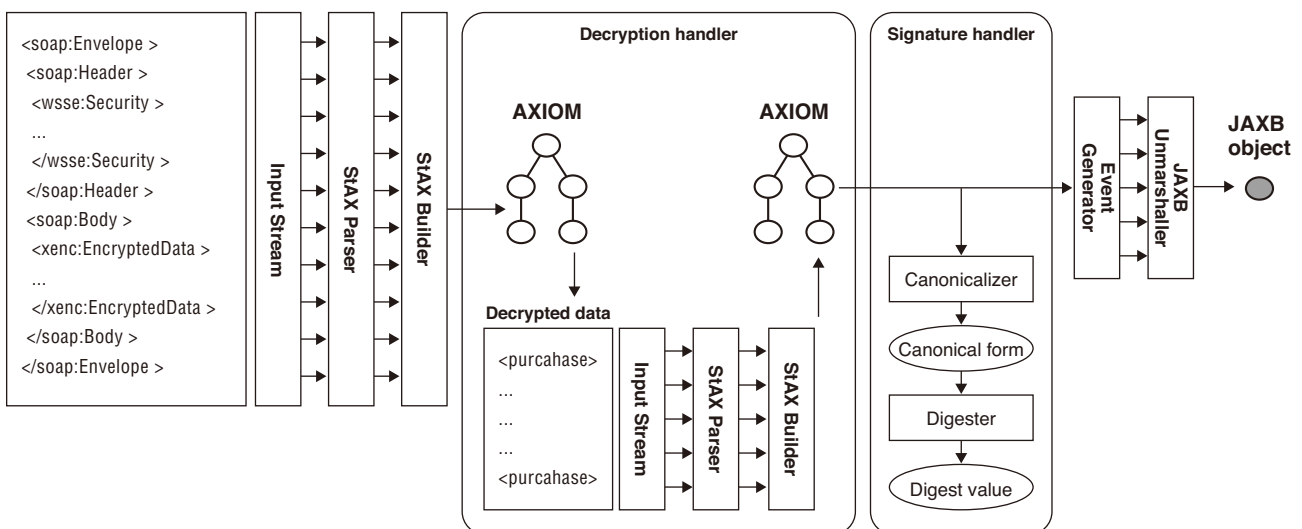


図3. 典型的なWS-Securityのシナリオを処理する木構造データを利用したプログラムの処理フロー



表1. スタンド・アロン・プログラムによる性能比較

Message size (byte)	AXIOM-based WS-Security [a]	StAX filter-based WS-Security [b]	Rate [b/a]
3k (byte)	12,969 (ms)	6.750 (ms)	52%
10k (byte)	26,015 (ms)	11,047 (ms)	46%
100k (byte)	220,985 (ms)	99,250 (ms)	45%

SOAP Header 内の Security header は削除し、この部分の AXIOM 作成が計測に含まれないようにした。

実験は IBM xSeries x365 (Xeon MP 3.0GHz x 4 CPUs, 8GB RAM, 4MB L3) Windows 2003 Enterprise Edition (32bit), IBM J9 VM (build 2.3, J2RE 1.5.0 IBM J9 2.3 Windows XP x86-32 j9vmwi3223-20071007 (JIT enabled)) 上で行った。Java の option には “-Xms1024m -Xmx1536m” を使用した。

実験結果を表 1 に示した。2 列目と 3 列目の値は 10,000 回の繰り返しに要した時間である。4 列目は 2 つの処理の比であり、値が少ない方が提案手法による性能改善の効果が大きいことになる。提案手法が WS-Security 処理を約 50% 高速化している。

また、WAS の WS-Security 実装を提案手法で置き換え、別のクライアント・マシンから Web サービスを呼び出し、スループットを計測することで、エンド・ツー・エンドの性能比較を行った。提案手法を採用した WAS ではリクエストとレスポンスがともに 3k バイトの Web サービス・アプリケーションにおいて約 10%、10k バイトのケースにおいて約 20%、100k バイトのケースにおいて約 30% のスループット増加を確認した。

## 5.2 性能評価実験の考察

実験の結果、SOAP Body の内容のメッセージ・サイズが大きくなればなるほど提案手法の性能改善の効果が大きくなった。これには、処理比率の変化と、Garbage Collection (GC) の 2 つの要因が考えられる。

WS-Security の処理には、メッセージと関係ない処理と、メッセージ・サイズの増加によって処理量も増す処理がある。例えば、暗号化エンジンの初期化やダイジェスト値の比較などの処理量は、メッセージ・サイズに依存せず一定である。一方、木構造データの作成、暗号化、そしてダイジェスト値の計算の処理量は、メッセージ・サイズに依存して増加する。

提案手法の効果はメッセージに関する処理を削減している。このためメッセージ・サイズが大きくなれば全処理量に占めるメッセージの処理量の比率が大きくなるた

め、提案手法の性能改善の効果も大きくなる。

メモリー確保と同様にメモリーの開放も処理量が多い。Java では GC が起動すればすべての処理が一時停止するため、GC の頻度が高いプログラムは遅いプログラムである。GC は空きメモリー量が減少した場合に起動する。このため、単位時間当たりに新たなオブジェクトを作成し開放した量が GC の頻度に比例する。

提案手法は木構造データを作成しないため、作成と開放するオブジェクトの量が少ない。このため、木構造データ型 API を利用した WS-Security 処理に比べ、GC の発生回数を抑えることができた。

## 6. おわりに

本論文では、WS-Security の性能を向上させるための手法としてイベント駆動型プログラムによる WS-Security の処理を行う StAX フィルターを提案した。スタンド・アロンの性能評価実験により、WS-Security 実装の範囲においておよそ 50% の性能改善を確認した。WAS を用いたエンド・ツー・エンドの性能評価実験では、10% から 30% の性能改善を確認した。

しかし、提案手法はすべての XML 処理には適用できない。XML パーサーのイベント・フィルターによる処理を行う場合、過ぎ去ってしまったイベントに対する読み込みや書き込みなどの処理はできない。この場合はもう一度読み直すか、木構造データ型 API を利用した方法を採用すべきである。

同様のストリーミング処理はメッセージを送り出す処理にも適用可能である。Web サービス・アプリケーションは JAXB オブジェクトを XML 文書に変換して送出する。WS-Security を適用すれば、この JAXB オブジェクトを一度 AXIOM に変換しなければならない。木構造データ作成の問題を解決する WS-Security 処理の送信プログラムの高速化が今後の課題である。

本論文では SOAP Header の木構造データ作成の処理性能の改善は考慮しなかったが、一般に典型的な WS-Security の設定を有効にすれば SOAP Header のサイズは約 7k バイト以上になる。XML の構造も複雑となるため、SOAP Header に対する性能改善は今後の重要な課題である。

参考文献

[1] Java Community Process: "JSR 224 Java API for XML-Based Web Services (JAX-WS) 2.0," <http://jcp.org/en/jsr/detail?id=224> (2006).

[2] W3C Recommendation: "SOAP version 1.2," <http://www.w3.org/TR/soap12-part0/> (2007).

[3] W3C Recommendation: "Extensible Markup Language (XML) 1.0," <http://www.w3.org/TR/2006/REC-xml-20060816/> (2006).

[4] OASIS standard: "Web Services Security: SOAP Message Security," <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf> (2004).

[5] W3C Recommendation: "XML-Signature Syntax and Processing," <http://www.w3.org/xmlsig-core/> (2001).

[6] W3C Recommendation: "XML Encryption Syntax and Processing," <http://www.w3.org/TR/xmlenc-core/> (2001).

[7] Internet Engineering Task Force: "RFC 2401 Security Architecture for the Internet Protocol (IPSec)," <http://www.ietf.org/rfc/rfc2401.txt> (1998).

[8] A. Frier, P. Karlton, and P. Kocher: "The SSL 3.0 Protocol," <http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt> (1996).

[9] Internet Engineering Task Force: "RFC 2246 The Transport Layer Security (TLS) Protocol Version 1.0," <http://www.ietf.org/rfc/rfc2246.txt> (1999).

[10] Internet Engineering Task Force: "RFC4250-4256 Secure Shell (SSH) Protocol," <http://tools.ietf.org/html/rfc4250> (2006).

[11] Satoshi Makino, Michiaki Tatsubori, Kent Tamura and Yuichi Nakamura: "Improving WS-Security Performance with a Template-Based Approach," Proceedings of 2005 IEEE International Conference on Web Services (ICWS 2005), pp.581-588 (2005).

[12] Masayoshi Teraguchi, Satoshi Makino, Ken Ueno and Hyen-Vui Chung: "Optimized Web Services Security Performance with Differential Parsing," Proceedings of International Conference on Service-Oriented Computing 2006 (ICSOC 2006), pp. 277-288 (2006).

[13] Satoshi Makino, Kent Tamura, Takeshi Imamura and Yuichi Nakamura: "Implementation of WS-Security and Its Performance Improvements," Proceedings of the International Conference on Web Services (ICWS 2003), pp. 256-264 (2004).

[14] Java Community Process: "JSR 222 Java Architecture for XML Binding (JAXB) 2.0," <http://www.jcp.org/en/jsr/detail?id=222> (2006).

[15] W3C Recommendation: "Document Object Model (DOM) Level 3 Specification," <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/> (2004).

[16] David Brownell: SAX2, O'Reilly, ISBN 0-596-00237-8 (2002) .

[17] Java Community Process: "JSR 173 Streaming API for XML (StAX) Specification," <http://jcp.org/en/jsr/detail?id=173> (2004).

[18] Jason Hunter and Brett McLaughlin: "JDOM," <http://www.jdom.org/> (2000).

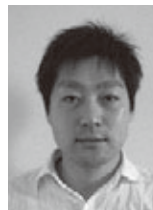
[19] "dom4j," <http://www.dom4j.org/>.

[20] "XOM," <http://www.cafeconleche.org/XOM/>.

[21] Apache Software Foundation: "Axis Object Model," <http://ws.apache.org/commons/axiom/>.

[22] W3C Recommendation: "Exclusive XML Canonicalization Version 1.0," <http://www.w3.org/TR/xml-exc-c14n/> (2002).

[23] W3C Recommendation: "Namespaces in XML 1.0," <http://www.w3.org/TR/REC-xml-names/> (2006).



日本アイ・ビー・エム株式会社  
東京基礎研究所  
副主任研究員

野ヶ山 尊秀 Takahide Nogayama

[プロフィール]

2004年、日本IBM入社。東京基礎研究所にてオートノミック・コンピューティングの研究・製品開発に従事。2006年よりWebサービス・エンジン、WS-Securityの高速化に従事。



日本アイ・ビー・エム株式会社  
東京基礎研究所  
主任研究員

高瀬 俊郎 Toshiro Takase

[プロフィール]

2000年、日本IBM入社。東京基礎研究所にてXML、Webサービス処理の高速化の研究に従事。



日本アイ・ビー・エム株式会社  
大和ソフトウェア研究所  
ITスペシャリスト

上野 憲一郎 Kenichiro Ueno

[プロフィール]

日本IBM大和ソフトウェア開発研究所にて、さまざまなお客様におけるWebSphere関連プロジェクトに参画。WebSphereパフォーマンス専門家として、セミナーなどで講演も実施。