LL-006

Proxyにおける XML処理プラグイン呼出し方式
Invocation framework for XML processing plug-ins in a proxy

井口 圭一† 小山 和也† 細野 繁† 藤田 悟†
Keiichi Iguchi, Kazuya Koyama, Shigeru Hosono, Satoru Fujita

1. はじめに

従来から Web においては，DeleGate[1]のように中継ノードでキャッシュや流量制御，ウィルススキャンなどを行う技術が広く使用されてきた．DeleGate は送受信されるメッセージ内容等の状況に応じて様々なフィルタ処理を行うために，外部フィルタ等の Plug-in 技術を用いて柔軟な構成変更が可能なように設計されている．

我々は Web サービスにおいて Web における DeleGate と同様に中継ノードで様々なフィルタ処理を行う Web サービス Proxy を研究開発している．Web サービス Proxy ではフィルタ処理を行う Plug-in で XML 文書の解析処理を必要とする場合が多い．しかし，XML 文書の解析処理は負荷が大きいため，従来の Plug-in 呼出し手法をそのまま Web サービス Proxy に使用すると，複数の Plug-in で解析処理を重複して行い，性能が低下するという問題が発生しうる．本稿では，このような Web サービス Proxy において XML 文書を効率よく処理可能な Plug-in インターフェイス MFI4P を提案し，実験によりその有効性を示す．

2. Web サービス Proxy 技術における課題

従来から，サーバや Proxy などにおいて Plug-in などの形で外部モジュール化されたプログラムを用いて，メッセージの内容に応じて動的に通信の暗号化や署名検証，ログ書出し等の機能の利用を決定し，実行する技術が広く使用されている．こういった技術には例えば，J2EE の Servlet Filter[2]，DeleGate の外部フィルタ，ICAP（Internet Content Adaptation Protocol[3]）などがあげられる．

Servlet Filter では，フィルタは Servlet Container からメソッド呼出しとして実行される．DeleGate の外部フィルタは独立したプログラムとして実行され，コンテンツはファイルディスクリプタとして渡される．また ICAP は，Proxy 等において付加機能の提供を分散的に行うためのプロトコルで，Proxy から機能を提供するサーバへ HTTP に似た形式でコンテンツの送信，変更依頼を行う．

これら従来技術では，Plug-in のインターフェイスは固定的に決定されており，内部の処理にインターフェイスの形式とは異なる形式が必要な Plug-in は各々で変換処理を行う必要がある．例えば図 1に示すように XML 文書を文字列 Stream の形式（以下 Stream 形式）で渡すインターフェイスを採用すると，複数の Plug-in が DOM 形式を必要とする場合に各々で DOM 構築の処理が重複して実行される．一方，DOM 形式で渡すインターフェイスを採用するとフレームワークが必ず一度 DOM 構築を行うため前述の重複処理は回避できるが，ログ出力など XML 解析が不要な Plug-in しか使用しない場合に DOM 構築処理が無駄になる．

†日本電気株式会社 インターネットシステム研究所
E-mail: k-iguchi@ap.jp.nec.com

そこで，本稿ではそれぞれ異なるデータ形式を要求する複数の Plug-in を必要とする場合に，必要最低限のデータ形式変換により効率よく処理可能な Multi-format Interface for Plug-in（MFI4P）を提案する．

3. MFI4P

MFI4P システムは，必要なデータ形式の宣言を持つ Plug-in，データ形式の変換を行うアダプタ，Plug-in に適切なデータ形式でコンテンツを渡すために Plug-in とアダプタの順序を設定し呼出すフレームワークからなる．

フレームワークは実行する各 Plug-in のデータ形式宣言を順次確認し，前の Plug-in の出力形式と次の Plug-in が要求する入力形式が一致していない場合には，実行リストにそれらのデータ形式間を変換するアダプタを挿入する．その後，変換アダプタを含めた Plug-in を実行することにより，各 Plug-in が必要なデータ形式でコンテンツを受け取ることが出来る．これにより，例えば連続した Plug-in が同じ形式を要求する場合には，新たなデータ形式変換処理が不要になり，解析負荷を低減することができる．

図 2に MFI4P を採用した Proxy の例を示す．この例では，Plug-in1 は入力形式として Stream を宣言し，Plug-in2, 3 は DOM 形式を宣言している．フレームワークは初め Plug-in1，2，3 の実行リストを，Plug-in1，Plug-in2 間のデータ形式が一致していないため，Stream→DOM 変換アダプタを挿入する．同様に Plug-in3 と外部への出力形式が一致していないため，DOM→Stream 変換アダプタを挿入する．そして出来上がった，Plug-in1，Stream→DOM 変換アダプタ，Plug-in2，Plug-in3，DOM→Stream 変換アダプタの順に実行することにより，各 Plug-in は宣言した通りのデータ形式でコンテンツを受け取ることが出来る．

ここでデータ形式変換回数を比較すると，従来手法では各々の Plug-in で行っていた Stream→DOM，DOM→Stream の変換が一組の変換アダプタで行えていることがわかる．同じデータ形式を要求する Plug-in が多く連続する場合にはその差はさらに顕著になる．

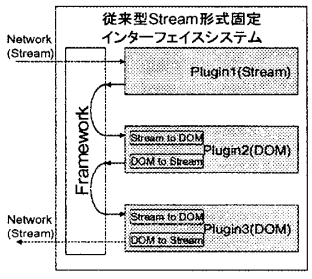

図 1 固定インターフェイスシステム (Stream 形式)

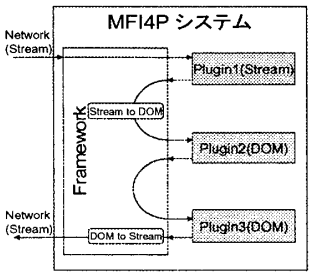

図 2 MFI4P システムの構成

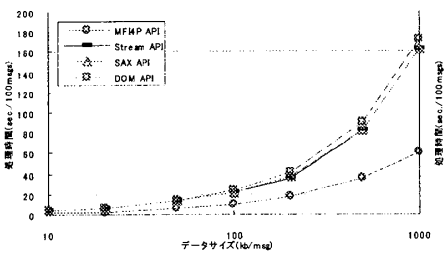

図３シナリオ１

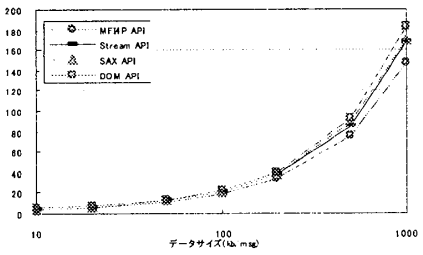

図４シナリオ２

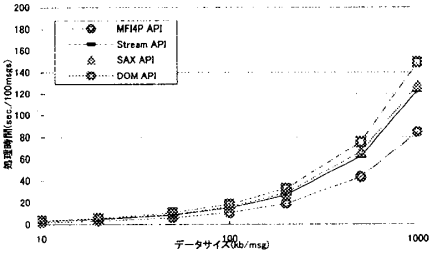

図５シナリオ３

4. 評価実験

本提案方式の性能を評価するため，Plug-in のインターフェイスに MFI4P を使用する Proxy での処理性能を測定し，従来の固定インターフェイス方式を使用する Proxy と比較した。

4.1 実験設定

測定には入力に必要とするデータ形式として，XML 文書を表現する代表的な Stream，SAX，DOM をそれぞれ宣言する 3 つの何も行わない Plug-in を準備した．また，変換アダプタとして Stream，SAX，DOM の各形式間を相互に変換できる 6 つのアダプタを準備した．Stream と SAX，DOM の変換には Xerces2.6.2[4]を，SAX と DOM との変換には Xalan 2.6.0[4]に含まれる DOM2SAX，SAX2DOM をそれぞれ使用した．比較する固定インターフェイス方式の Proxy としては，インターフェイスのデータ形式に Stream，SAX，DOM の 3 つの形式を採用した Proxy をそれぞれ準備した．

MFI4P の効果は実際に使用される Plug-in とその順序に依存するため，本実験では使用する Plug-in の決定ルールとして，以下の 3 つのシナリオを準備した．シナリオ 1 では，全メッセージに Stream，Stream，SAX，SAX，DOM，DOM の順で 6 つの Plug-in を実行する．同じ形式の Plug-in が連続することから，MFI4P がもっとも優位性を示すと思われる．シナリオ 2 では同じ Plug-in の順序を DOM，SAX，Stream，DOM，SAX，Stream に変更する．この場合毎回データ形式変換処理が必要になるので MFI4P にとって不利な設定となる．シナリオ 3 では使用する Plug-in をランダムに 1～10 個選択する．様々な順序で Plug-in が実行される場合の平均的な性能を求めることが出来る．

処理するメッセージは，XML の階層が 5 段までの比較的フラットな文書タイプ A と，100 段の階層が深い文書タイプ B の 2 種類について，10Kbyte から 1M までサイズを変えた SOAP 文書を使用し，各データサイズの文書をそれぞれ 100 通処理した際の処理時間を測定した．測定環境は JDK1.4.2，Windows 2000 Server，Pentium4 2GHz，メモリ 1GB である．

4.2 実験結果

表 1に各変換アダプタの処理性能をしめす．DOM 構築処理がやや重く，SAX, DOM から Stream へのシリアライズ処理は比較的軽いことが分かる．

図 3から図 5に各シナリオでタイプ A の SOAP 文書を，図 6にシナリオ 1 でタイプ B の SOAP 文書を，それぞれ 4 つの Proxy で処理した際の処理時間を処理した結果を示す．縦軸は 100 メッセージの処理時間（秒），横軸は対数表示のメッセージのデータサイズ（kb）である．図 3，図 4によるとシナリオ 1，2 では順序を除いて実行する Plug-in の種類と数は同じなので，従来の固定インターフェイスを使用する Proxy ではそれぞれほぼ同じ処理時間となっているが，MFI4P はシナリオ 1 では型変換処理を省略できるため半分以下の時間で処理が行えている．また，MFI4P が苦手なシナリオ 2 でも従来手法と同程度の性能を示す．各形式の Plug-in をランダムな順序で実行するシナリオ 3 および，XML の階層が深いタイプ B の文書についても，いずれのデータ形式を採用した従来手法よりも高速に処理が終了しており，複数の入力形式を宣言する Plug-in が混在する状況で MFI4P が一般的に有効であると言える．

次に，MFI4P を採用することによるオーバヘッドについて考察する．MFI4P では固定 API と比較して，毎回必要なデータ形式変換を決定するためのオーバヘッドが存在する．しかし，この必要な変換を決定するオーバヘッドを測定すると 1000 個の Plug-in を使用する場合でも 1kb のメッセージに対するデータ変換コストに対して 1%程度と僅かであり，単一のデータ形式以外必要としないことが明らかな場合以外は，MFI4P を採用することによるデータ形式変換負荷の削減効果の方が大きいだろう．

5. おわりに

本稿では，Plug-in を使用する Web サービス Proxy において XML 解析負荷を低減する MFI4P を提案し，実験によりその有効性を示した．

また本稿では触れなかったが，Plug-in の実行順序を変更することが出来れば，実験におけるシナリオ 2 の場合でも，シナリオ 1 と同じ性能を発揮することが出来ることになる．しかし，実際には各 Plug-in 間では実行順序の制約が存在する場合があり，その制約をどのように記述し，制約を満たした中で並べ替えを行うかが今後の課題である．

参考文献

[1] DeleGate http://www.DeleGate.org/
[2] J2EE Servlet Filter http://java.sun.com/products/servlet/Filters.html
[3] ICAP http://www.i-cap.org/docs/icap_whitepaper_v1-01.pdf
[4] Apache XML Project: Xerces，Xalan. http://xml.apache.org/

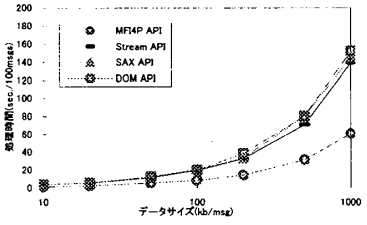

図６タイプＢ文書の処理時間
（シナリオ１）

	TypeA (sec)	TypeB (sec)
Stream2SAX	4.18	3.92
Stream2DOM	6.38	5.17
SAX2DOM	5.04	5.29
SAX2Stream	3.06	2.40
DOM2Stream	2.25	1.86
DOM2SAX	3.59	3.10

表１変換アダプタの処理時間
(sec/ 100kb・100msgs)