非同期 Web サービスの信頼性向上に関する研究

2005MT020　服部　正敏　　2005MT121　上野　佳宏　　2005MT133　長澤　伸治

指導教員　青山　幹雄

1. はじめに

現在 Web サービスは，ステートレスで同期型が多い．本研究では，Web サービスのステートフル非同期メッセージのアーキテクチャを提案し，プロトタイプを実装する．プロトタイプを用いてアーキテクチャの妥当性を検証し，評価を行う．

2. 非同期 Web サービスの問題点

2.1. 非同期 Web サービスの問題点

非同期 Web サービスはブロッキングがないため，処理効率が向上する．しかし，以下の 2 点の問題点がある．

(1) リクエスタの視点での信頼性の低下

リクエスタのリクエストメッセージがプロバイダに受信されたか確認できない．またサービスが処理中にエラーを引き起こした場合，リクエスタはエラーを確認できない．

(2) メッセージ交換の複雑化

リクエスタにメールサーバやコールバックなどのレスポンスメッセージを待つ仕組みが必要となる．さらに結果の問合せを行う場合，ポーリングの仕組みも必要となる．そのためリクエスタ，もしくはプロバイダのメッセージ交換の複雑化や負荷が増大すると考えられる．

3. 関連研究

Web サービスにおいて，既存のサービスプロバイダを修正せずに Web サービス中にコールバックを実装し，非同期通信の実現方法としてプロキシパターンが提案されている[3]．クライアント，サービスプロバイダ間でコールバックを可能とするプロキシを作成する．これによりクライアントは通知を受け取るスレッドに専念でき，サービスプロバイダは処理に専念できる．またプロキシとサービスプロバイダとのメッセージ交換方法はリクエスト／レスポンス型である．そこでコールバックプロキシはコールバック登録情報をレジストリに保持し，リクエスタからの要求により One-Way 型のリクエストを送信することで非同期通信を実現する．しかし，サービスプロバイダとプロキシ間はコールバックの有効期限が切れるまでポーリングをし続ける問題がある．

4. アーキテクチャの提案

非同期メッセージ交換において，サービスの状態をリクエスタが確認できるステートフル非同期メッセージアーキテクチャを提案する．

4.1. ステートフル非同期メッセージアーキテクチャ

提案するアーキテクチャは，図 1 に示すプロセスに従い，非同期リクエスト／レスポンス型でリクエスタがサービス処理状態を確認できるシステムを実現する．非同期 Web サービスは参照モデル(図 2)に基づいて 3 階層のメッセージ交換パターン(図 3)を構成する．このパターンをサービスの要求や条件に基づきパターンの選択を行い，アーキテクチャの実装を行う．

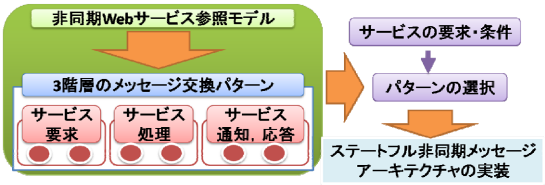

図 1: 開発プロセス

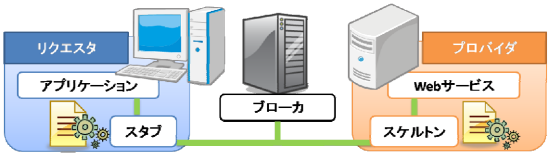

図 2: 非同期 Web サービス参照モデル

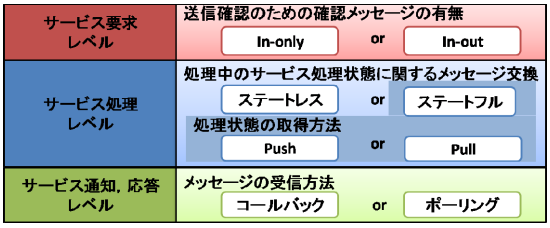

図 3: 3 階層のメッセージ交換パターン

図 2 に基づき図 3 で構成されたパターンを 3 階層のシーケンスとして適用する(図 4)．次に 3 階層のメッセージ交換パターンについて説明する．

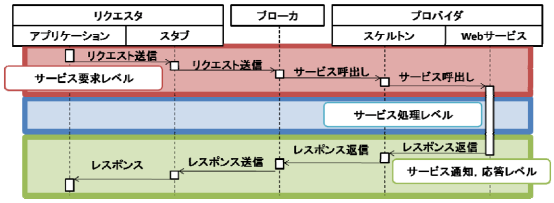

図 4: 参照モデルに基づき 3 階層のメッセージ交換パターンを適用したシーケンス

4.2. サービス要求レベル

プロバイダのメッセージの受信をリクエスタが確認できるリクエストメッセージ方法を提案する(図 5).

(1) In-only パターン

従来の One-Way 型メッセージモデル. アプリケーションからのリクエストを一方向で送信する.

(2) In-out パターン

レスポンスと独立した確認メッセージを返信するメッセージモデル. アプリケーションのサービスを起動したことを確認できる.

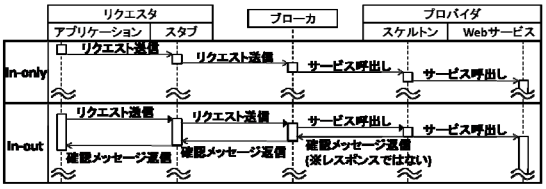

図 5:In-only , In-out パターン

4.3. サービス処理レベル

プロバイダのリクエスト受信後の処理について, サービス処理状態を確認しないステートレスパターンとサービス処理状態を確認するステートフルパターンに分類する(図 6).

(1) ステートレスパターン

サービス処理をプロバイダが行う間, リクエスタはプロバイダのサービスと接続を行わない.

(2) ステートフルパターン

リクエスタはサービス処理を行うプロバイダの処理状態を取得し, 処理の進行や正常性を確認する.

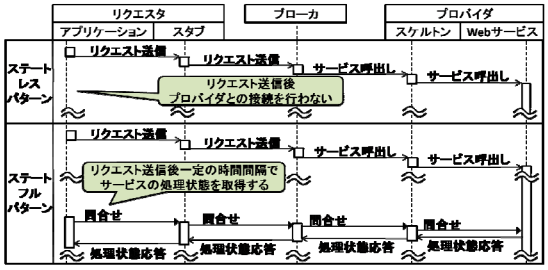

図 6:ステートレス, ステートフルパターン

4.3.1. ステートフルパターンの分類

ステートフルは, サービス処理状態を取得する方法を以下の二つのパターンで分類する(図 7).

(1) Push 型

リクエストメッセージを受信したサービスは, サービスの処理を行うと共に, 現在の処理状態を一定間隔でリクエスタに送信する. そのためリクエスタはサービスの処理状態を確認できる.

(2) Pull 型

リクエスタは, プロバイダまで問合せすることで, サービスの処理状態を確認する.

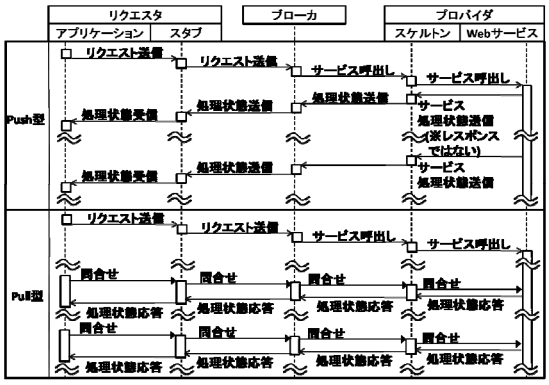

図 7:Push 型, Pull 型

4.4. サービス通知, 応答レベル

リクエスタがプロバイダからのメッセージを取得する方法をコールバック, ポーリングの二つのパターンで分類する(図 8).

(1) コールバックパターン

リクエスタは, リクエストメッセージ送信後, コールバックスレッドの役割を果たすスタブを生成し, プロバイダからのメッセージを待機する.

(2) ポーリングパターン

リクエストメッセージ送信後, リクエスタはブローカに一定間隔で問合せを行い, サービスの処理が未処理なら未処理応答を取得し, 処理が完了していれば応答メッセージとしてレスポンスを取得する.

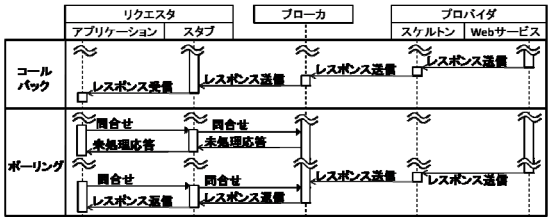

図 8:コールバック, ポーリング

5. プロトタイプの実装

ステートフル非同期メッセージモデルに基づきプロトタイプの 3 階層のメッセージ交換パターンを選択, 実装する.

5.1. 3 階層モデルの統合シーケンス

3 階層のメッセージ交換パターンの各レベルで選択し, 統合モデルを決定する(図 9).

(1) サービス要求レベルの選択:In-out パターン

(2) サービス処理レベルの選択:Push 型

(3) サービス通知, 応答レベルの選択:ポーリング

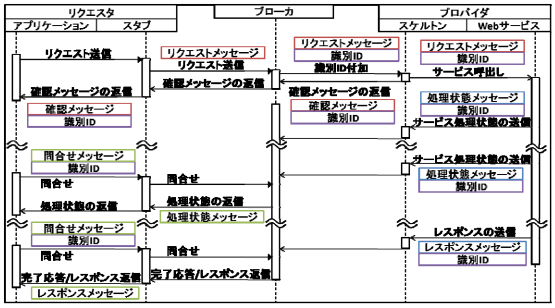

図 9:3 階層のメッセージ交換パターンの統合シーケンス

5.2. 提案したアーキテクチャの実装

信頼性を向上する非同期 Web サービスアーキテクチャを実現に用いたコンポーネントについて以下に示す(図 10). リクエスタのリクエストメッセージ送信からレスポンスメッセージの受信までに以下の処理を行う. サービス要求時はリクエスタ, プロバイダ間で HTTP を用いてメッセージ交換を行う(1). サービス処理時, プロバイダはサービス処理状態を一定間隔で SMTP を用いてブローカに送信する. サービス処理が完了時もレスポンスを SMTP で送信を行う. ブローカは受信したメッセージから処理状態をキャッシュするデータベースに処理状態やレスポンスを登録する(2). サービス通知, 応答時ではリクエスタはブローカに対して HTTP を用いて問合せを行い, レスポンスとして処理状態を取得する. ブローカにレスポンスがキャッシュされていればレスポンスを取得する(3).

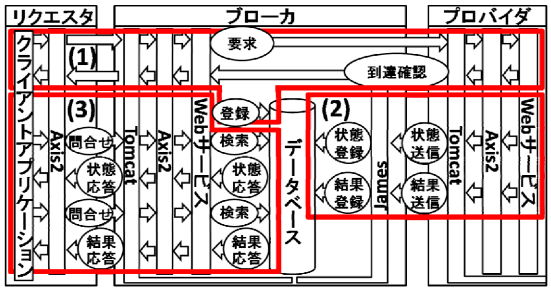

図 10:提案したアーキテクチャの実装

6. プロトタイプの検証と評価

6.1. 実装環境

プロトタイプを以下の環境で実装した(図 11).

リクエスタ, ブローカ, プロバイダ環境		ブローカのソフトウェア環境		プロバイダのソフトウェア環境	
実行環境	Java 2 SE 6.0	SOAPサーバ	Apache Axis2 1.4	SOAPサーバ	Apache Axis2 1.4
OS	WindowsXP Home SP2	Webサーバ	Apache Tomcat 6.0	Webサーバ	Apache Tomcat 6.0
メモリ	1280MB	Mailサーバ	Apache James 2.3		
CPU	Celeron(R) M 1.5GHz	データベースサーバ	MySQL Server 5.0		

図 11:実装環境

6.2. アーキテクチャの妥当性

提案するアーキテクチャのプロトタイプを実装し, リクエスタの視点での信頼性とメッセージ交換の複雑性による負荷を以下の 2 点で評価した.

(1) サービスの処理状態の確認
(2) プロバイダのサービス処理状態の送信からリクエスタの処理状態の確認までの時間

6.3. サービス処理状態の確認

プロトタイプでは, リクエスタがプロバイダのサービス処理状態を確認できるようにした. その状態確認の例を図 12, 13 に示す. 図 12 はリクエストに対する確認メッセージとして識別 ID をリクエスタが受信し, サービス要求レベルの振舞いである確認メッセージの受信を確認した. また, 図 13 は, リクエスタが識別 ID を基にして, ブローカに対して, 処理状態の確認をしている. そのためサービス処理レベルの振舞いであるサービスの処理状態の確認をした.

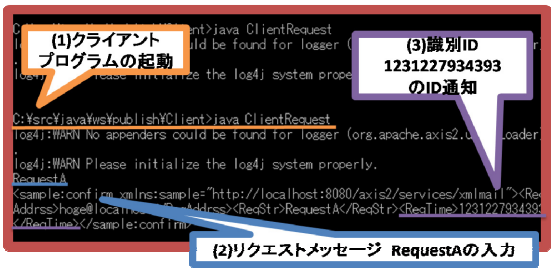

図 12:リクエスト送信画面

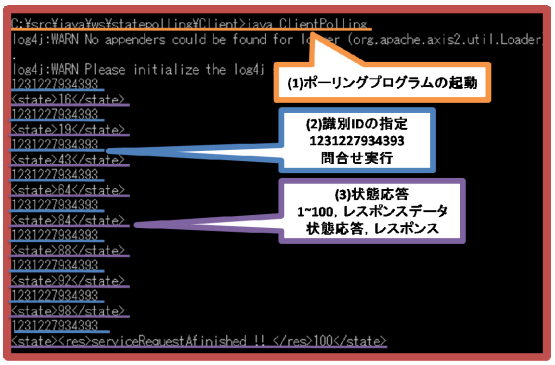

図 13:リクエスタ状態確認画面

6.4. プロバイダのサービス処理状態の送信からリクエスタの処理状態の確認までの時間

リクエスタ, ブローカ, プロバイダ間のサービスの処理状態の通知を時間要素に分けて検証した(図 14). また, 各時間要素の定義を表 1 に示す.

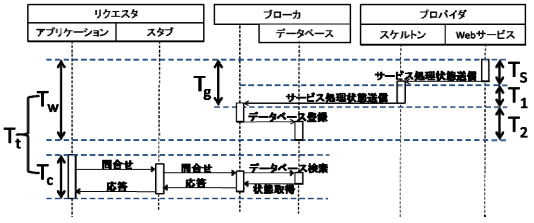

図 14:サービスの処理状態の通知時間

表 1:各時間要素の定義

時間	定義
T_s	サービス処理状態の送信準備時間(プロバイダ)
T_g	サービス処理状態の送信準備(プロバイダ)から，メッセージ受信(ブローカ)までの時間
T_1	リクエスト送信(プロバイダ)からメッセージ受信(ブローカ)までの時間
T_2	メッセージ受信からデータベース登録(ブローカ)までの時間
T_w	サービス処理状態の送信(プロバイダ)からデータベース登録(ブローカ)までの時間
T_c	ブローカへ問合せし，データベースに格納されている処理状態を確認するまで(リクエスタ)の時間
T_t	サービス処理状態の送信準備(プロバイダ)から処理状態の確認(リクエスタ)までの時間

時間要素 T_s, T_g, T_w, T_c を実装したプロトタイプから測定した．T_1, T_2, T_tは，それぞれ(1)~(3)式に示す．また，T_w, T_c の時間の推移を図 15，16 に示す．

$$T_1 = T_g - T_s \quad (1)$$

$$T_2 = T_w - T_g \quad (2)$$

$$T_t = T_w + T_c \quad (3)$$

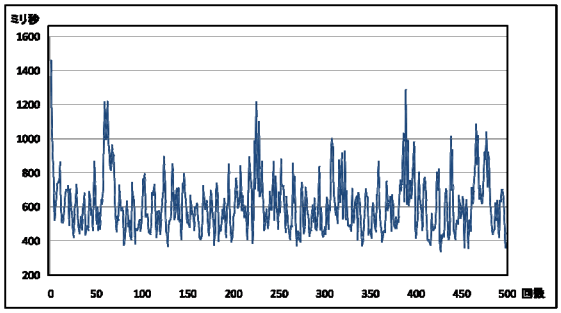

図 15:T_w の推移

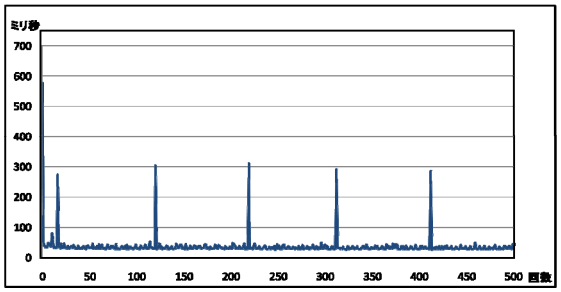

図 16:T_c の推移

図 15 のように T_w の測定では，非常に不規則になった．一方，図16のようにT_cは一定でかつ，周期的なピークが発生した．これらはJavaのガベージコレクション(GC)の影響であり，T_s, T_g からも同様に検出された．次に各時間要素の平均時間を示す(表2)．ただしT_sは150ミリ秒，T_gは600ミリ秒，T_wは1000ミリ秒，T_cは200ミリ秒を基準に，それ以上の値をGC の異常値として排除した．

表 2:1 回あたりの各要素の平均時間(単位:ミリ秒)

回数	100	500	1000	平均値
T_s	87.8	79.5	72.9	80.1
T_g	271.4	278.4	238.7	262.8
T_1	186.4	198.4	166.3	183.7
T_2	297.0	316.9	358.9	324.3
T_w	568.4	595.3	597.6	587.1
T_c	44.1	32.7	31.7	36.2
T_t	612.5	628.0	629.3	623.3

6.5. プロトタイプの評価

実装したプロトタイプでは処理状態を確認できたため，提案したアーキテクチャは非同期 Web サービスメッセージ交換の信頼性向上に有効である．また検証結果より，実装した環境では，T_tが 623 ミリ秒以上掛かるため，623 ミリ秒以下の間隔で状態の更新が必要なサービスには有効ではないと考えられる．

7. 今後の課題

リクエスタの視点からさらに信頼性を向上するには，処理状態メッセージの詳細化が必要である．また，処理状態を取得する時間を短縮するために REST の技術要素を適用したメッセージ交換方法[2]も検討する必要がある．

8. まとめ

本研究では，ステートフル非同期メッセージアーキテクチャを提案し，プロトタイプを実装した．プロトタイプではサービスの処理状態の確認を行い，サービスの信頼性の向上を示した．また，処理状態通知に必要な時間を測定し，提案したアーキテクチャの妥当性を検証し，評価した．

参考文献

[1] D. Jayasinghe, Quick Start Apache Axis2, Packt Publishing, 2008.

[2] 池崎 崇, 永橋 陽一郎, 軽量 Web サービスアーキテクチャに関する研究, 2006 年度南山大学卒業論文, 2007.

[3] K. Qian, et al. Asynchronous Callback in Web Service, Proc. IEEE SNPD '06, Jun. 2006, pp. 375-380.

[4] 森 晃, 服部 隆尚, 非同期型 Web サービスに関する研究, 2004 年度南山大学卒業論文, 2005.

[5] 森 晃, 青山 幹雄, 非同期メッセージ交換のモデルとパターンに基づく非同期サービス指向アーキテクチャ設計方法, 情報処理学会論文誌, Vol. 48, No. 8, Aug. 2008, pp. 2567-2576.