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

森 晃[†,☆] 青山幹雄[††]

サービス指向アーキテクチャ（SOA）を非同期メッセージで実現するための非同期メッセージ交換方法とそれに基づく非同期サービス指向アーキテクチャ（非同期 SOA）の設計方法を提案する．SOA は Web サービスなどと連携してネットワーク上でアプリケーションを開発する基盤技術である．しかし，その設計方法は主として同期メッセージを基礎としているため，非同期メッセージを基礎とするアプリケーションの開発方法は確立されているとはいえない．特に，非同期メッセージの形態は多様で，アプリケーションの要求に適したメッセージ交換方法の選択と適用が必要となる．本稿では，非同期メッセージ交換をその特性に基づいて分類し，非同期メッセージ交換パターン（AMEP）として提案する．非同期 SOA を実現する整合性を持った AMEP の組合せを非同期 SOA パターンとして定義し，パターンに基づく非同期 SOA の設計方法を提案する．Apache Axis などを用いた非同期 SOA の実装を比較・評価し，提案方法が従来提案された非同期メッセージパターンを包含した一般的方法であることを示す．

Design Method of Asynchronous Service-Oriented Architecture Based on the Models and Patterns of Asynchronous Messaging

Akira Mori[†,☆] and Mikio Aoyama[††]

This article proposes a method for designing asynchronous SOA (Service-Oriented Architecture) based on the asynchronous messaging models. SOA is a key technology to develop applications collaborating Web services. However, the development method based on the SOA is still limited since the conventional SOA focuses on synchronous messaging. Asynchronous messaging provides a wide variety of messaging. However, due to the variety, it's necessary to select appropriate messaging model and design the applications behavior around the model. This article proposes AMEP (Asynchronous Message Exchange Pattern) by classifying the messaging with behavior properties. Composing appropriate AMEPs across the messaging hierarchy enables to generate a set of asynchronous SOA patterns, which can be instantiated to design applications based on an asynchronous SOA. We demonstrate the effectiveness of the proposed method by comparing the implementation of SOA based on the Apache Axis and other platforms.

1. はじめに

サービス指向アーキテクチャ（SOA: Service-Oriented Architecture）は，ネットワーク上に点在する Web サービスなどの種々のサービスをメッセージを介して連携し，アプリケーションの提供やビジネスのコラボレーションを実現する[2),17)]．しかし，SOA の基盤技術である Web サービス[15),21)] は，トランスポート層に HTTP を使用したリクエスト/レスポンス型の同期メッセージを主として利用しているため，非同期メッセージを基盤とするワークフロー管理[20)]，サプライチェイン管理[18)]，アプリケーション統合[9)] などの構築には制約がある．特に，非同期メッセージは多様な交換方法があり[7),10),13)]，その利用にはアプリケーションに適したメッセージ交換方法の理解，選択，組合せが必要となることから，設計が複雑となる．たとえば，非同期メッセージを用いた SOA の設計には，次の問題がある．

(1) 多様なメッセージ交換モデルの設計

メッセージの交換形態として，応答のない 1 方向のメッセージ送付と応答のある双方向のメッセージ交換

† 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanazan University
†† 南山大学数理情報学部情報通信学科
Faculty of Mathematical Sciences and Information Engineering, Nanzan University
☆ 現在，富士ゼロックス株式会社
Presently with Fiji Xerox Co., Ltd.

がある．また，メッセージ交換の応答先が 1 つとなる 1 対 1 のメッセージ交換と複数の場合となる 1 対 n の場合がある．それらを実現するソフトウェアアーキテクチャも複数ありうる．

(2) 異なるメッセージ交換モデル間の組合せ制約

SOA では，SOAP などのアプリケーション間のメッセージ交換プロトコルと下位のプロトコルを分離し，組み合わせることができるが[2),21)]，上位と下位のメッセージ交換プロトコルがミスマッチとならないように，選択と組合せの制約を考慮する必要がある．

このような非同期メッセージ交換に基づく SOA の設計を体系的に行うために，本稿では非同期メッセージ交換のパターン化に基づく非同期サービス指向アーキテクチャ（非同期 SOA）の開発方法を提案する．

2. モデルとパターンによる非同期 SOA 設計方法

2.1 非同期 SOA のモデルとパターン

非同期 SOA とは，非同期メッセージ交換を用いてサービス間の連携を行うサービス指向アーキテクチャと定義する．

非同期 SOA 設計方法の要素技術として，以下に示すメッセージ交換のモデルとパターンを定義する．

(1) メッセージ交換共通モデル：同期，非同期メッセージ交換と SOA，非同期 SOA の関係を定義するメッセージ交換のメタモデル．

(2) 非同期メッセージモデル：非同期メッセージ交換をトポロジとインタラクションで分類した，プラットフォームと独立なメッセージ交換の論理モデル．

(3) 非同期メッセージ交換パターン AMEP（Asynchronous Message Exchange Pattern）：非同期メッセージモデルを SOA で実装することを想定し，4 階層の階層構造でモデル化し，各階層でメッセージ交換の特性を基準にその形態をパターン化．

(4) 非同期 SOA パターン：非同期メッセージ交換パターンを階層間の制約などを考慮し，適切に組み合わせて実現されたアーキテクチャパターン．

2.2 モデルとパターンに基づく非同期 SOA の設計方法

図 1 に提案するモデルとパターンに基づく非同期 SOA 開発方法のプロセスを示す．非同期メッセージ交換を実現するメッセージプロトコルとその実装アーキテクチャは多様である．そのため，本稿では，SOA を前提とする．しかし，プラットフォームなどの実装条件に依存しない一般モデルに基づき，プラットフォームに依存する特性を付加し，設計条件を段階的に明確にするアプローチはモデル駆動開発と共通する．

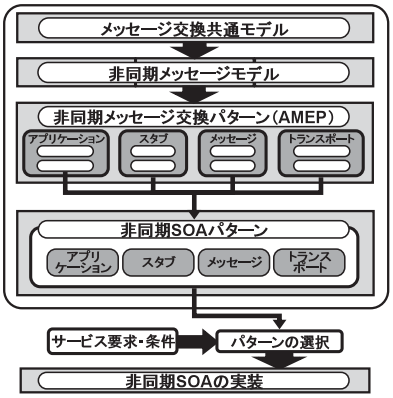

図 1 モデルとパターンに基づく非同期 SOA 開発
Fig. 1 Model-and-pattern driven development of asynchronous SOA.

まず，プラットフォームとは独立なメッセージ交換の一般モデルを非同期メッセージモデルとして定義する．これは，非同期メッセージ交換の設計にモデル駆動開発の考えを応用したものである．

非同期メッセージモデルを実現するために，SOA をターゲットとするプラットフォームに依存するモデルとして 4 階層モデルを導入する．SOA では，アプリケーション間のメッセージプロトコルとトランスポートのメッセージプロトコルを分離して組み合わせできることに着目したものである．

4 階層の各階層のメッセージ交換の特性に基づき，各階層内の非同期メッセージ交換の形態を非同期メッセージ交換パターン AMEP としてパターン化する．

さらに，4 階層の AMEP を要求に応じて組み合わせ，要求仕様に適した非同期メッセージ交換を実現するアーキテクチャを非同期 SOA パターンとして定義する．

AMEP の組合せには制約がある．アプリケーションの開発者は，要求仕様やプラットフォームの特性に基づき，この制約の範囲内でパターンのマッチングをとり，選択されるパターンが実現可能かを判断する必要がある．そのため，非同期 SOA パターンは，あらかじめ階層間の整合性を保証した AMEP の組合せをアーキテクチャパターンとして提示する．要求仕様に基づき非同期 SOA パターンを選択し，再利用することにより非同期 SOA の構築を可能とする．

3. 非同期 SOA と非同期メッセージ交換のモデル化とパターン化

3.1 メッセージ交換共通モデル

図 2 に非同期 SOA とそのメッセージ交換共通モデ

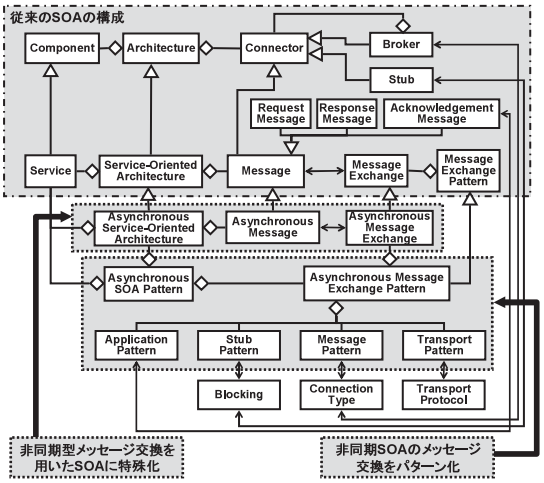

図 2 非同期 SOA と非同期メッセージ交換のメタモデル
Fig. 2 Meta-model of asynchronous SOA and asynchronous messaging.

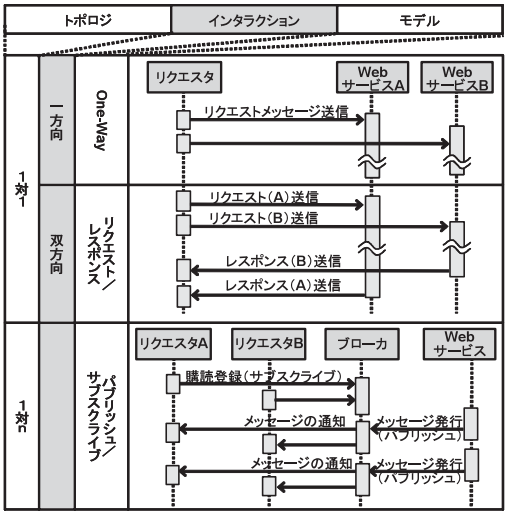

図 3 非同期メッセージモデル
Fig. 3 Asynchronous messaging models.

ルのメタモデルを示す．

SOA は，コンポーネントとコネクタによって構成されるソフトウェアアーキテクチャの拡張と定義し，サービスとメッセージから構成され，メッセージ交換によってサービスを利用する．SOA を拡張して，メッセージ交換に非同期メッセージ交換を用いた非同期 SOA を定義する．

非同期SOA パターンは，MEP（Message Exchange Pattern）[8] を拡張した AMEP により構成される．AMEP はそれを構成する 4 階層のパターンの集約として定義する．各階層の AMEP はメッセージ交換の性質によりパターン化される．

3.2 非同期メッセージモデル

非同期メッセージ交換は，同期型のリクエスト/レスポンス型とは異なり通信形態が様々である．アプリケーション間でのメッセージ交換の形態や特性による分類が必要である．本稿では，メッセージ交換の一般的分類基準として，メッセージの配信構造であるトポロジとメッセージ交換のスタイルであるインタラクションを導入し非同期メッセージの実装によらない一般的モデルとして，次の 3 つの非同期メッセージモデルを定義する（**図 3**）[1),16)]．

(1) One-Way メッセージモデル：応答メッセージのない 1 方向メッセージ交換を 1 対 1 で行うモデル
(2) リクエスト/レスポンスメッセージモデル：1 対 1 のメッセージ交換を双方向で行うモデル
(3) パブリッシュ/サブスクライブメッセージモデル：複数のリクエスタに同一のメッセージを発行し，通知するモデル

3.3 非同期メッセージ交換パターン

3.3.1 非同期メッセージ交換の実現

アプリケーション間の非同期メッセージモデルを実現するアーキテクチャには様々な形態がある．本稿では，SOA の主要プラットフォームである Web サービスを前提とし，Web サービスの持つ次の 2 つの特性に着目して非同期メッセージモデルのパターン化を行う．

(1) プロトコルの階層的構造化
Web サービスのメッセージ交換では，アプリケーション層とトランスポート層を分離できるので，各層でプロトコルを選択し，組み合わせることが可能である．したがって，非同期メッセージモデルを階層的なプロトコルの組合せとして構造化できる．
(2) プロトコルの多段階フィルタリングモデル
Web サービスでは WSDL，SOAP に準拠した標準インタフェースと標準メッセージ交換を基礎とする．これは，データフローアーキテクチャパターンの，多段フィルタとしてモデル化できる．

本稿では，Web サービスのメッセージ交換を**図 4** に示す 4 階層に分け，各階層のメッセージ交換の特性に基づき，非同期メッセージ交換パターン AMEP を定義する．提案する 4 階層のカテゴリと分類基準となるメッセージ交換の特性を**表 1** に示す．

3.3.2 アプリケーションレベルの特性とパターン化

アプリケーションレベルはアプリケーション間のメッセージ交換の挙動を定義する．アプリケーションレベ

表 1 4 階層モデルのメッセージ交換の特性
Table 1 Properties in message exchange hierarchical model.

4 階層モデル	意味	メッセージ交換の性質
アプリケーションレベル	サービス間の相互作用	・ 送信確認メッセージの有無
スタブレベル	アプリケーションとメッセージ間のインタフェースでの相互作用	・ レスポンスメッセージの待ち合わせの有無 ・ レスポンスメッセージの受信方法
メッセージレベル	メッセージルーティングの性質	・ エンドポイント間の接続形態
トランスポートレベル	プロトコルの性質	・ トランスポートプロトコルの通信方式

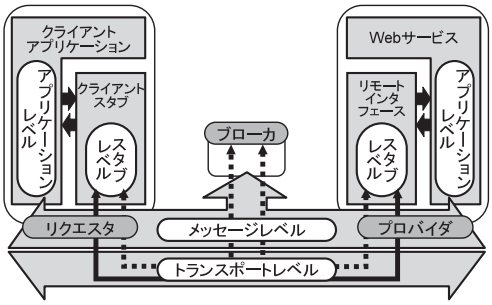

図 4 4 階層メッセージ交換モデル
Fig. 4 Four-layer hierarchical message exchange model.

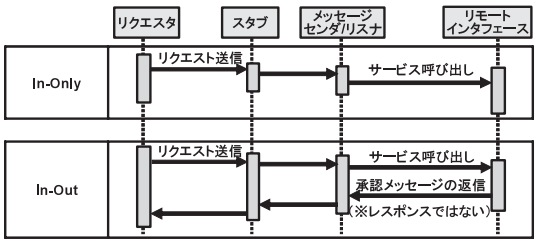

図 5 In-Only パターンと In-Out パターン
Fig. 5 In-Only pattern and In-Out pattern.

ルでは，リクエストメッセージを送信後，メッセージ送信の正常性を確認するメッセージの有無により，確認メッセージを必要としない In-Only パターンと必要とする In-Out パターンに分類できる（**図 5**）．

3.3.3 スタブレベルの特性とパターン化

スタブレベルでは，アプリケーションとメッセージ間の挙動のマッチングをとる．スタブレベルでは，要求メッセージを送信後，リクエスタが応答メッセージを受信するまで待ち合わせるか否かにより，待ち合わせる Blocking パターンと，待ち合わせない Non-Blocking パターンに分類できる（**図 6**）．Non-Blocking パターンを選択することにより，スタブレベルの非同期を実現する．

また，Non-Blocking パターンは非同期に応答メッセージを受信する方法により，プロバイダから通知されるコールバックパターンとリクエスタからの問合せによってメッセージを受信するポーリングパターンに分類できる（**図 7**）．

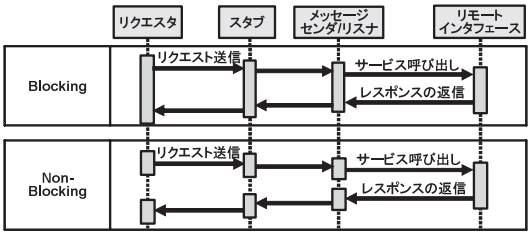

図 6 Blocking パターンと Non-Blocking パターン
Fig. 6 Blocking and Non-Blocking patterns.

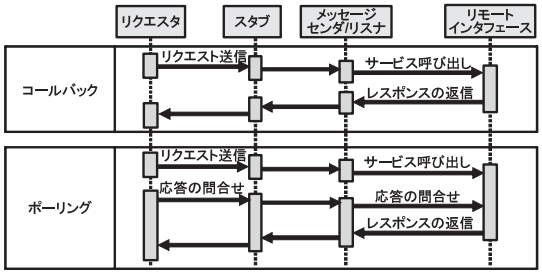

図 7 コールバックパターンとポーリングパターン
Fig. 7 Callback and polling patterns.

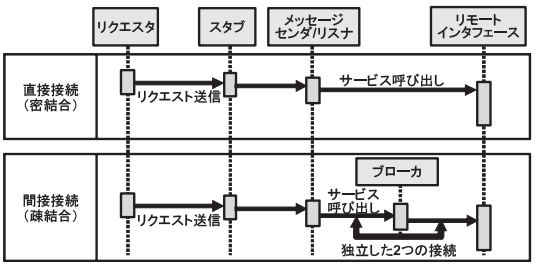

図 8 直接交換パターンと間接交換ターン
Fig. 8 Direct and indirect exchange patterns.

3.3.4 メッセージレベルの特性とパターン化

メッセージレベルはメッセージの授受の方法を規定する．リクエスタとプロバイダ間が直接メッセージ交換する直接交換パターンと，ブローカなどを介して間接的に交換する間接交換パターンに分類できる（**図 8**）．間接交換ではリクエスタとプロバイダは独立した 2 つのメッセージ交換に分離できるため，各メッセージ交換が同期メッセージ交換であってもエンドポイント間では非同期メッセージ交換が実現できる．

3.3.5 トランスポートレベルの特性とパターン化

トランスポートレベルでは，リクエスタとプロバイダ間のトランスポートの接続方法により，Single Transport パターンと Dual Transport ポートパターンに分類できる（**図 9**）．

例として，同期型のプロトコルである HTTP を用いた非同期メッセージ交換を示す．通常，HTTP では Single Transport として同期メッセージ交換を行う．しかし，要求と応答を 2 つの HTTP セッションに分離し，独立に送信することによりトランスポートレベルでの非同期メッセージ交換を実現することもできる．

3.4 非同期 SOA パターン

4 階層の AMEP を組み合わせることで非同期 SOA を実装する雛形となる非同期 SOA パターンを定義する．しかし，**表 2** に示すように，各階層の AMEP の中には非同期メッセージモデルに対応できない組合せがある．これは，非同期メッセージモデルを実現できる AMEP には制約があり，非同期メッセージモデルごとに適切な AMEP の選択と組合せが必要であることを意味する．非同期メッセージモデルが実現可能な AMEP を選択し，階層的アーキテクチャパターンにまとめ，次の 3 つの非同期 SOA パターンとして提案する．

(1) One-Way 型非同期 SOA パターン
(2) リクエスト/レスポンス型非同期 SOA パターン
(3) パブリッシュ/サブスクライブ型非同期 SOA パターン

3.4.1 One-Way 型非同期 SOA パターン

One-Way 型は 1 方向のメッセージ交換を行うパターンである．One-Way 型非同期 SOA パターンでは，アプリケーションレベル，メッセージングレベルでは任意のパターンの選択が可能である（**図 10**）．しかし，1 方向メッセージでは応答メッセージを受信しないため，スタブレベルでは Non-Blocking パターンが選択され，さらにトランスポートレベルでは，要求メッセージを送信するのみの Single Transport パターンが選択される．

3.4.2 リクエスト/レスポンス型非同期 SOA パターン

リクエスト/レスポンス型はリクエスタとプロバイダが 1 対 1 である双方向のメッセージ交換を行うパターンである．双方向のメッセージであるリクエスト/レスポンス型非同期 SOA パターンは各レベルの分類基準に依存しないため，各レベルですべてのパターンが適用可能である（**図 11**）．

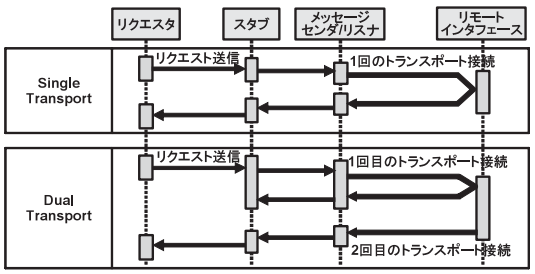

図 9 Single Transport パターンと Dual Transport パターン
Fig. 9 Single Transport and Dual Transport patterns.

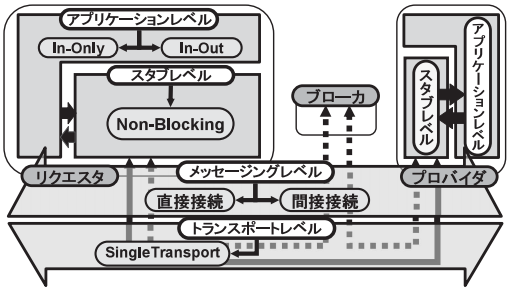

図 10 One-Way 型非同期 SOA パターン
Fig. 10 Asynchronous SOA pattern of One-Way.

表 2 非同期メッセージモデルと AMEP の対応関係
Table 2 Relationship between asynchronous message models and AMEPs.

			非同期メッセージモデル		
			One-Way	リクエスト/レスポンス	パブリッシュ/サブスクライブ
アプリケーションレベル	In-Only		○	○	○
	In-Out		○	○	○
スタブレベル	Blocking		×	○	○
	Non-Blocking	コールバック	○	○	○
		ポーリング		○	×
メッセージレベル	直接交換		○	○	×
	間接交換		○	○	○
トランスポートレベル	Single Transport		○	○	×
	Dual Transport		×	○	○

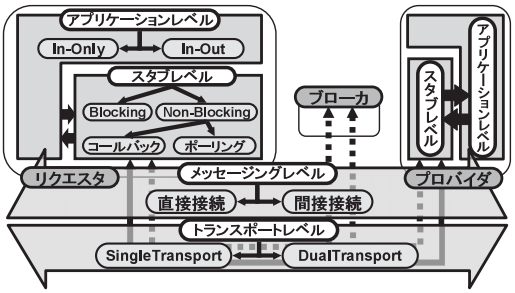

図 11 リクエスト/レスポンス型非同期 SOA パターン
Fig. 11 Asynchronous SOA pattern of request/response.

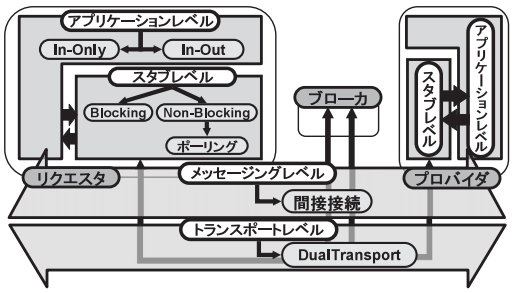

図 12 パブリッシュ/サブスクライブ型非同期 SOA パターン
Fig. 12 Asynchronous SOA pattern of publish/subscribe.

3.4.3 パブリッシュ/サブスクライブ型非同期 SOA パターン

パブリッシュ/サブスクライブ型はプロバイダが複数のリクエスタに同一のメッセージを通知する 1 対多のメッセージ交換である[6]．リクエスタは，あらかじめメッセージサーバ（ブローカ）にメッセージ購読要求（サブスクライブ）を送信し，プロバイダからメッセージが発行（パブリッシュ）されると，登録された複数のリクエスタにメッセージが配信される（**図 12**）．

パブリッシュ/サブスクライブ型非同期 SOA パターンでは，アプリケーションレベルとスタブレベルでのパターンの選択が可能である．メッセージ購読要求とメッセージ発行はメッセージサーバとしてのブローカを介して行われるため，メッセージレベルではブローカを中継する間接交換が必然的に選択される．

また，リクエスタからのサブスクライブとプロバイダのパブリッシュはそれぞれ異なるトランスポート接続であるため，トランスポートレベルではデュアルトランスポートパターンが選択される．

4. パターンに基づく非同期 SOA の実装

非同期 SOA パターンを用いて，非同期メッセージ交換を用いた非同期 SOA を実装する方法を示す．例として，実行環境には，オープンソースの Web サービスエンジンである Apache Axis[3),4)] と，非同期プロトコルである SMTP 上で SOAP メッセージを交換する SOAP over SMTP を用いる．

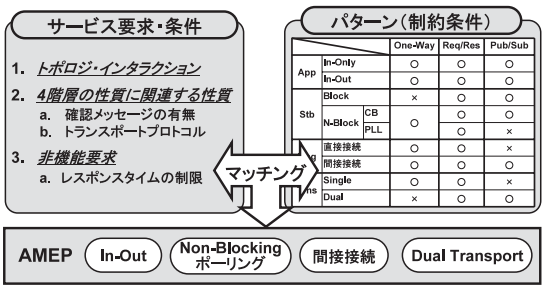

図 13 AMEP の選択
Fig. 13 Selection of AMEP.

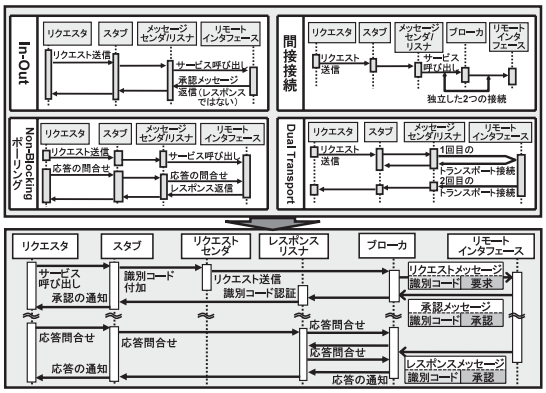

図 14 4 階層 AMEP の統合
Fig. 14 Integration of four layers of AMEP.

4.1 AMEP の選択

非同期 SOA の実現には，開発するサービスの要求に基づいた 4 階層 AMEP の選択が必要となる．パターンの選択は，まず外部仕様として，開発するサービスのトポロジとインタラクションを与えて，メッセージモデルを決定する．次に，各メッセージモデルの AMEP の制約条件とサービスの要求のマッチングをとり，選択される AMEP が実現可能かを判断する．AMEP の選択の可否を判断するためのサービス要求は，4 階層の性質に関連する要求だけでなく，信頼性や待ち時間の制限といった非機能要求も必要となる（**図 13**）．

4.2 AMEP を統合した非同期 SOA パターン

4 階層から選択された AMEP のメッセージ交換シーケンスを組み合わせて，非同期 SOA を実現する非同期 SOA パターンのメッセージ交換シーケンスを決定する．非同期 SOA パターンのメッセージ交換シーケンスは上位層の AMEP からトップダウンで組み合わせる．**図 14** の例では，各層の性質である，承認メッ

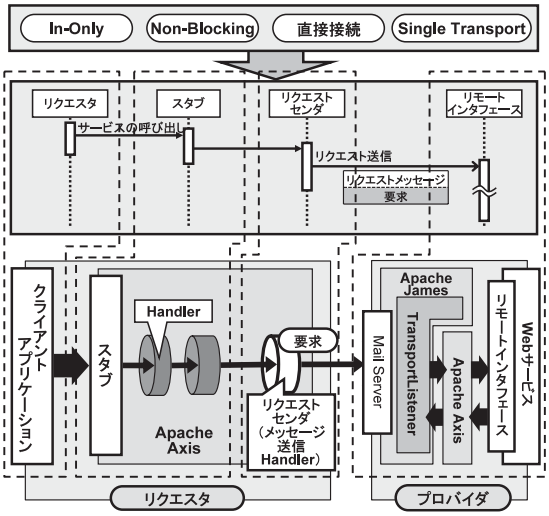

図 15 1 方向メッセージアーキテクチャ
Fig. 15 Architecture of One-Way message.

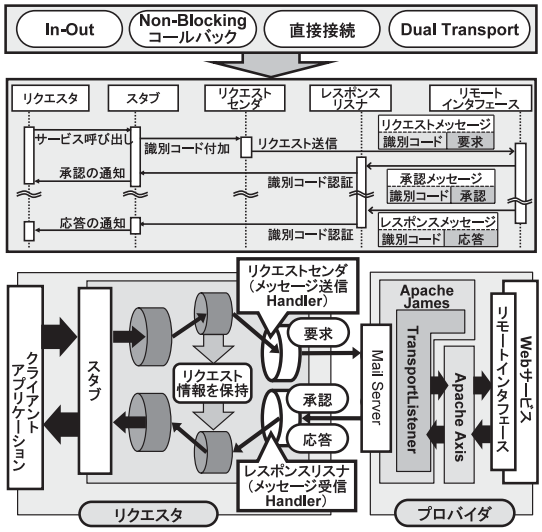

図 16 コールバック型双方向メッセージアーキテクチャ
Fig. 16 Architecture of Two-Way message using callback.

セージの受信，問合せによるレスポンスの受信，ブローカ，2 回のトランスポート接続を順に非同期 SOA パターンに追加して決定する．

4.3 非同期 SOA パターンに基づく非同期 SOA の実装

非同期 SOA パターンを用いて非同期 SOA を構築する場合，非同期 SOA パターンの各要素に対応した機能を持つコンポーネントとの対応付けを行う．

本稿では，非同期 SOA パターンに基づく非同期 SOA の実装例として，3 つのアーキテクチャを示す．Apache Axis は，ハンドラと呼ぶフィルタの接続により実現されるデータフローアーキテクチャであることから，パターンの各要素を Apache Axis の該当する Handler に対応付けて非同期 SOA を構築する．

(1) 1 方向メッセージアーキテクチャの実現

One-Way 型 AMEP を用いた 1 方向メッセージを実現するアーキテクチャを**図 15** に示す．

リクエストセンダを実現するメッセージ送信 Handler は，各 Handler で処理された SOAP メッセージのバインディングヘッダに SMTP ヘッダを付加し，Web サービスを提供するメールサーバにリクエストメッセージを送信する．サーバが SOAP メッセージを受信すると Web サービスプロバイダは該当する Web サービスを起動する．

(2) コールバック型双方向メッセージアーキテクチャ

コールバックパターンを用いた双方向メッセージの実現には 1 方向のアーキテクチャに加え，次のコンポーネントが必要となる．

1) 要求と応答の対応関係を ID，送信者情報により特定する Handler（スタブ）
2) メッセージ受信 Handler（レスポンスリスナ）
3) イベント駆動により応答をリクエスタに返信する Handler（スタブ）

上記の 3 つのコンポーネントを組み入れたコールバック型双方向メッセージアーキテクチャを**図 16** に示す．送信時にメッセージ識別用の ID 要素を付加し，リクエスタ側で要求メッセージの情報を一時的に保存する．応答メッセージをメッセージ受信 Handler が受信し，保存した送信情報との照合を行い，発信元へイベント駆動により応答を返信する．

(3) ポーリング型双方向メッセージアーキテクチャ

ポーリングパターンを用いた双方向メッセージの実現にはコールバックパターンの 3 つのコンポーネントに加えて，応答メッセージの着信の有無をリクエスタ側のメールサーバに問い合わせるメッセージ受信 Handler が必要となる．

ポーリング型双方向メッセージのアーキテクチャを**図 17** に示す．

リクエスタは要求メッセージを送信後，メッセージ受信 Handler がメールサーバに対して応答メッセージの有無を周期的に問い合わせる．送信メッセージの ID に対応する応答メッセージが着信している場合は発信元へ応答を返信する．未到着の場合，POP で周期的に問合せを繰り返す．

表 3 Apache の AMEP への対応
Table 3 Correspondence of Apache Axis to AMEP.

			非同期メッセージモデル								
			One-Way			リクエスト/レスポンス			パブリッシュ/サブスクライブ		
			非同期 SOA パターン	Axis 1.x	Axis 2	非同期 SOA パターン	Axis 1.x	Axis 2	非同期 SOA パターン	Axis 1.x	Axis 2
アプリケーション	In-Only		○	×	○	○	○	○	○	×	×
	In-Out		○		○	○	○	○	○		
スタブ	Blocking		×		×	○	○	○	○		
	Non-Blocking	コールバック	○		○	○	×	○	○		
		ポーリング				○		×	×		
メッセージ	直接接続		○		○	○	○	○	×		
	間接接続		○		○	○	○	○	○		
トランスポート	Single Transport		○		○	○	○	○	×		
	Dual Transport		×		×	○	×	○	○		

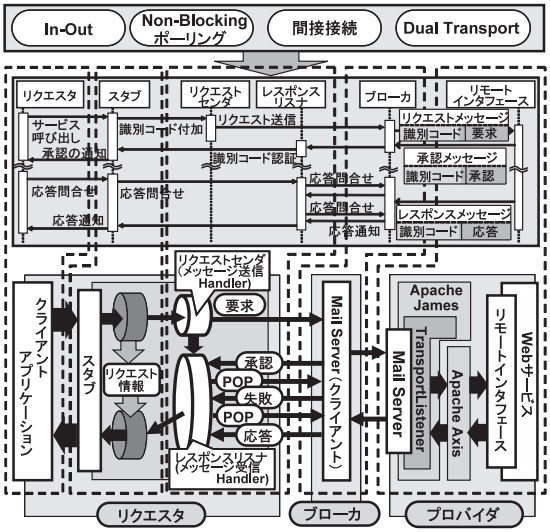

図 17 ポーリング型双方向メッセージアーキテクチャ
Fig. 17 Architecture of Two-Way message using polling.

5. 非同期 SOA 設計方法の評価

本稿で提案した非同期 SOA の設計方法は，実装に依存しないアプリケーション間のメッセージ交換モデルから実装へ体系的に展開できる点である．

5.1 Apache Axis による非同期 SOA の実現可能性評価

表 2 に示すように，非同期 SOA パターンを実現できる AMEP には制約があることから，SOA のプラットフォームを階層ごとに AMEP と対応付けることにより，プラットフォームが実現できる非同期 SOA パターンの範囲を知ることができる．たとえば，Web サービスの主要なプラットフォームである Apache Axis の 1.x と 2.0 の 2 つのバージョンの各 Handler の機能を分析し，AMEP との対応範囲を比較した．これに基づき，2 つのバージョンで実現できる非同期 SOA の範囲を**表 3** に示す．

Axis 1.x 系は，One-Way 型とパブリッシュ/サブスクライブ型に対応していないため，いずれのパターンも実現できない．リクエスト/レスポンス型では，1 回の HTTP 接続を用いて応答を待ち合わせる，Blocking パターンと Single Transport パターンのみに対応しているため，Non-Blocking パターンと Dual Transport パターンは選択不可となる．一方，Apache Axis 2[3),19)] では，One-Way 型で選択可能なすべての AMEP に対応している．また，リクエスト/レスポンス型では，2 回のトランスポート接続に対応しているため，Dual Transport パターンの選択が可能となる．

以上のように，Apache Axis のバージョンにより実現できる非同期 SOA パターンに制限があることが分かる．いずれのバージョンでも，One-Way 型とリクエスト/レスポンス型の場合は，ほぼすべてのパターンに対応できる．しかし，すべてのパターンへの対応は困難であるため標準で対応していない AMEP には独自の Handler を追加して非同期 SOA の実装する必要がある．

5.2 Apache Axis にパターンを付加した One-Way 型非同期 SOA の実現

AMEP を用いると，ある非同期 SOA を実現するためにプラットフォームに欠けているパターンを発見できる．そのため，プラットフォームを再利用しながら，不足パターンを追加することにより実装できる．

5.2.1 SOAP over SMTP の実現方法

4 階層 AMEP として In-Only，Non-Blocking，直接交換，Single Transport パターンを組み合わせた，1 方向メッセージを送信する非同期 SOA パターンの実装を示す．Apache Axis 1.2 は 1 方向の非同期メッセージ交換に対応していないため，この非同期 SOA パターンは，そのままでは実装できない．パターン内のリクエストセンダとリモートインタフェースに対応

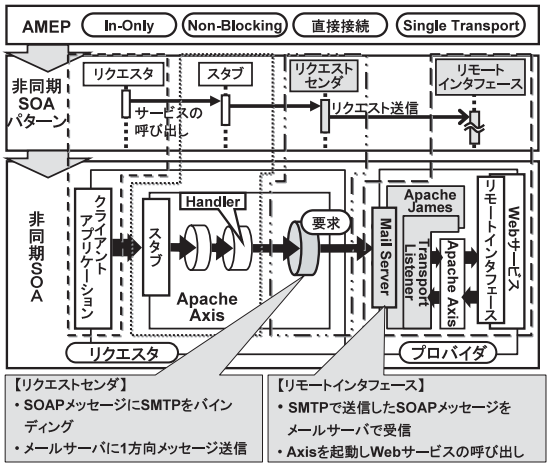

図 18 SOAP over SMTP の実現方法
Fig. 18 Method of achieving SOAP over SMTP.

するコンポーネントが未実装である（**図 18**）.

そこで，リクエストセンダに対応する Handler として，SOAP メッセージを送信するトランスポートプロトコルに SMTP をバインディングし，メールサーバに 1 方向のリクエストメッセージを送信する Handler を追加する．さらにプロバイダ側には，SMTP で送信されたメッセージをメールサーバで受信し，Axis を起動してサービスを呼び出すためのリモートインタフェースを実装する[14].

5.2.2 非同期型 SOAP メッセージの送信

SOAP over SMTP で SOAP メッセージを送信するためにリクエスタ側の Apache Axis に SMTP 用のリクエストセンダである SMTPSender を実装，配置した．SMTPSender は，各 Handler で処理された SOAP メッセージのバインディングヘッダに SMTP ヘッダを付加し，Web サービスを提供するメールサーバに対してリクエストメッセージを送信する．

SMTPSender が行う処理の流れを**図 19** のシーケンス図に示す．Apache Axis の AxisClient から invoke メソッドが呼び出されると，SMTPSender は MessageContext オブジェクトを受け取る．SMTPSender は MessageContext から送信元と送信先のメールアドレスを取得する．得られたメールアドレスを MIME 形式のメールメッセージを表す MimeMessage オブジェクトに渡す．最後に MessageContext から SOAP で記述されたリクエストメッセージ本体を取り出し，MimeMessage にバイト列として出力しメッセージを Web サービスプロバイダへ送信する．

5.2.3 Mailet を用いたサービスプロバイダの実装

SMTP で送信される SOAP メッセージをメールサーバで受信し，Axis を起動して Web サービスを呼び出すために，Apache James[5] の Mailet を用いて，AxisMailet を実装した．AxisMailet は電子メールとして受信した SOAP メッセージを MessageContext に渡して，起動した Axis の Handler に渡す．

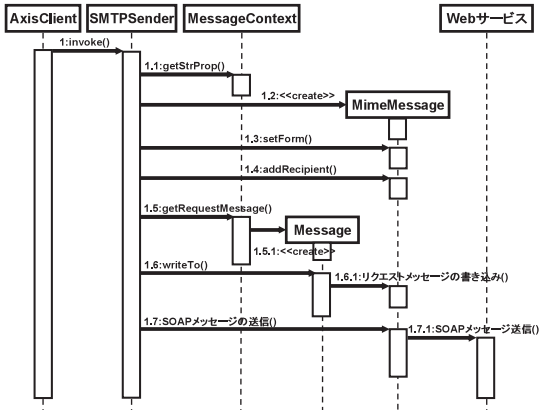

図 19 SMTPSender のシーケンス図
Fig. 19 Sequence diagram of SMTPSender.

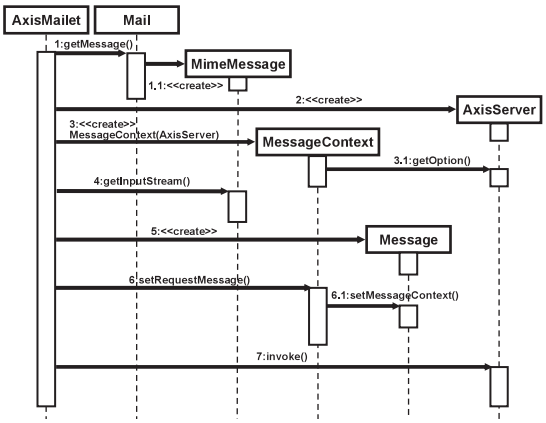

図 20 AxisMailet のシーケンス図
Fig. 20 Sequence diagram of AxisMailet.

AxisMailet の処理の流れを**図 20** に示す．

AxisMailet は Apache James から起動されると，Mail オブジェクトを受け取る．まず，Mail から MimeMessage を取得し，AxisServer オブジェクトを作成する．次に，MessageContext オブジェクトと Message オブジェクトを作成し，MimeMessage から取得したリクエストメッセージを Message にセットし，さらに MessageContext に Message をセットする．最後に AxisServer の invoke メソッドを呼び出して AxisServer に MessageContext を渡す．

前節の SMTPSender の実装を含む，Apache Axis への SOAP over SMTP の実装で作成した Java プロ

グラムの規模を**表 4** に示す．

5.3 非同期 SOA パターンに基づく Ajax の実装

本稿で提案する非同期 SOA パターンは実装環境に依存しないため，Apache Axis 以外にも様々な非同期メッセージ交換を実現することが可能である．例として，非同期 SOA パターンを用いた Ajax アプリケーションの非同期メッセージ交換の実現を示す．

Ajax は**図 21** の下部に示すように，Web サーバから HTTP リクエストによって HTML を受信した後に，JavaScript によって，非同期に Web ページの一部を更新するためのメッセージ交換が行われる．このメッセージ交換は，提案する AMEP から以下の 4 階層パターンを組合せとして実現できる．

1) In-Only パターン
2) Non-Blocking /コールバックパターン
3) 直接接続パターン（間接接続も可）
4) Single Transport パターン

表 4 SOAP over SMTP の実装規模
Table 4 Implementation statistics of SOAP over SMTP.

	行数(LOC)	クラス数(新規)
リクエスタ	310	4
プロバイダ	105	1
合計	415	5

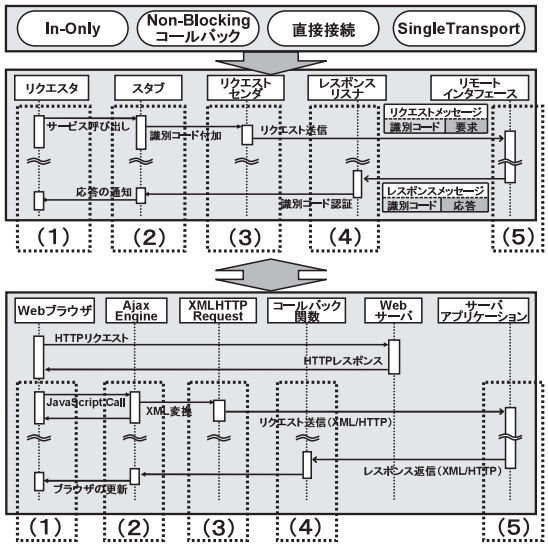

図 21 AMEP を用いた Ajax の実現
Fig. 21 Realization of Ajax using AMEP.

6. 関連研究との比較と考察

非同期メッセージに基づく Web サービスアプリケーションの開発方法にはいくつかの提案がある．これらの非同期メッセージ交換に関するパターンやモデルを本稿で提案する非同期 SOA パターンと対応付け，**表 5** に示す．表から，各パターンの提案が本稿で提案しているパターンのサブセットとなっていることが分かる．

(1) 非同期呼び出しパターン（実現範囲：表 5 実線枠内）

既存の Web サービスフレームワーク上で非同期呼び出しを提供する 4 つの非同期呼び出しパターンが提案されている[22]．このパターンは，1 方向メッセージを確認メッセージの有無，双方向メッセージをレスポンスの受信方法で分類したパターンである．

(2) 2 層非同期通信モデル（実現範囲：表 5 破線枠内）

Web サービス呼び出しを 2 層のメッセージ交換で行う 2 層非同期通信モデルが提案されている[11]．このモデルはメッセージ交換を API 層（本稿のスタブ層に相当）とトランスポート層に分離し，API 層で応答の待合せの有無と受信方法，トランスポート層で HTTP の接続回数を定義している．

(3) プロキシ/ブローカデザインパターン（実現範

表 5 関連研究が提案する非同期メッセージパターンの適用範囲
Table 5 Coverage of asynchronous messaging patterns.

			非同期メッセージモデル		
			One-Way型	リクエスト/レスポンス型	パブリッシュ/サブスクライブ型
アプリケーションレベル	In-Only		(1) ○	○	○
	In-Out		○	○	○
スタブレベル	Blocking		(2)	○	○
	Non-Blocking	コールバック	○	(1) ○	○
		ポーリング		○	×
メッセージレベル	直接交換		○	○	× (3)
	間接交換		○	○	○
トランスポートレベル	Single Transport		(2) ○	○	×
	Dual Transport		×	○	○

囲：表 5 二重線枠内）

パブリッシュ/サブスクライブ型のメッセージ交換において，プロキシやブローカを介してメッセージを通知する方法を定義したプロキシ/ブローカデザインパターンが提案されている[12]．このパターンは，リクエスタがサービスの購読登録を行う際にサービスの身元を知っているか，そして，直接送信できるかという，可視性と到達可能性によりメッセージ交換を分類する．

この 3 つの関連研究は，非同期メッセージ交換を分類したパターンとモデルを定義しているが，いずれも本稿で提案する非同期 SOA パターンのサブセットとして定義できる．

また，(1) の非同期呼び出しパターンでは，1 つの分類基準のみでメッセージ交換が決定される．このため，たとえばリクエスト/レスポンス型で確認メッセージの有無を指定しつつ，レスポンスの受信方法を指定するバリエーションが実現できないという問題がある．一方，提案する非同期 SOA パターンは，メッセージ交換を 4 階層モデルで定義しているため，各階層のパターンを組み合わせた多様なメッセージ交換を実現できる．

7. 今後の課題

(1) 分類基準の依存関係

4 つのレベルに属するメッセージ交換の分類基準には，他のレベルの選択に依存するケースが考えられるため，各レベルの分類基準の依存関係を示す．

(2) パターン適用の判断基準

Web サービス開発者が，提案するモデルから開発に適するメッセージ交換パターンを得るためには，サービスの要求をパラメータとして入力する必要がある．今後は，各サービスに適するメッセージ交換パターンを特定するための判断基準を検討する．

(3) Apache Axis 以外への非同期 SOA パターンの適用

本稿では，Apache Axis を用いた実装を提案したが，他のプラットフォームでも，パターンの要素とアーキテクチャとの対応付けを明確にする必要がある．

8. まとめ

本稿は，非同期メッセージ交換を基礎とする非同期サービス指向アーキテクチャ（非同期 SOA）の開発方法を提案した．非同期 SOA とその基礎となるメッセージ交換のプラットフォーム独立な一般モデルを起点とし，非同期メッセージ交換を実現する階層的なプロトコルを非同期メッセージ交換パターン（AMEP）としてパターン化することにより，非同期 SOA を AMEP の組合せとして設計する方法を提案した．

本提案に基づき，Apache Axis などのメッセージ交換を分析した結果，パターンの組合せ制約，実装プラットフォームの非同期 SOA の実現可能性，適合可能性を判定できることを示した．また，既存のプラットフォームで実現できない場合，AMEP に基づき不足なパターンを抽出し，当該パターンを追加することにより実現できることを，1 方向の非同期メッセージを用いた非同期 SOA を例として示した．

本稿で提案した開発方法は，関連研究に比べ非同期メッセージ交換の広範なパターンを包含し，非同期メッセージに基づく非同期 SOA を設計するうえで広い視野に立った体系的方法を提示している点で有効である．

参考文献

1) Adams, H.: Asynchronous Operations and Web Services (2002). http://www-106.ibm.com/developerworks/library/ws-asynch1.html
2) 青山幹雄：ソフトウェアサービス技術へのいざない，情報処理，Vol.42, No.9, pp.857–862 (2001).
3) The Apache Software Foundation: Apache Axis2 Documentation. http://ws.apache.org/axis2/1_0/index.html
4) The Apache Software Foundation: Axis 1.2. http://ws.apache.org/axis/
5) The Apache Software Foundation: James. http://james.apache.org/
6) Barros, A., et al.: Service Interaction Patterns, *Proc. BPM 2005*, LNCS 3649, pp.302–318, Springer (2005).
7) Chappell, D.A.: *Enterprise Service Bus*, O'Reilly (2004).
8) Gudgin, M., et al. (Eds.): SOAP Version 1.2 Part 2: Adjuncts, 2nd ed., W3C (Dec. 2006). http://www.w3.org/TR/soap12-part2/
9) Hohpe, G., et al.: *Enterprise Integration Patterns*, Pearson Education (2004).
10) Ibbotson, J. (Ed.): SOAP Version 1.2 Usage Scenarios, W3C WG Note (July 2003). http://www.w3.org/TR/xmlp-scenarios
11) 木村利幸：非同期 Web サービス実現アーキテクチャ提案 (2004). http://ws.apache.org/~toshi/docs/JSR224-F2F-no1-jp.pdf
12) Li, L., et al.: Semantic Modeling and Design Patterns for Asynchronous Events in Web Service Interaction, *Proc. IEEE ICWS 2006* (*Int'l Conf. on Web Services*), IEEE Computer Soci-

ety, pp.223–230 (Sep. 2006).
13) Mishra, S.K.: Asynchronous Messaging using Web Services (Aug. 2006). http://www-128.ibm.com/developerworks/webservices/library/ws-soa-async.html
14) 森 晃，青山幹雄：SMTP を用いた非同期型 Web サービスの提案と評価，情報処理学会ソフトウェア工学研究会，Vol.2005-SE-147, pp.73–80 (Mar. 2005).
15) 本 俊也：最新 Web サービスマスタリングハンドブック，秀和システム (2004).
16) Narayanan, M.: Approaches to Asynchronous Web Services (Sep. 2003). http://www-128.ibm.com/developerworks/webservices/library/ws-asoper/
17) Newcomer, E. and Lomow, G.: *Understanding SOA with Web Services*, Addison-Wesley (2005).
18) Poirier, C., et al.: *The Networked Supply Chain*, J. Ross Publishing/APICS (2004).
19) Perera, S., et al.: Axis2: Middleware for Next Generation Web Services, *Proc. ICWS 2006* (*2006 IEEE Int'l Conf. on Web Services*), IEEE Computer Society, pp.833–840 (Sep. 2006).
20) van der Aalst, W. and van Hee, K.: *Workflow Management: Models, Methods and Systems*, MIT Press (2004).
21) Weerawarena, S., et al.: *Web Services Platform Architecture*, Prentice Hall PTR (2005).
22) Zdum, U., Voelter, M. and Kircher, M.: Pattern-Based Design of an Asynchronous Invocation Framework for Web Services, *Int'l. J. of Web Service Research*, Vol.1, No.3, pp.1–14 (July-Sep. 2004).

(平成 18 年 12 月 5 日受付)
(平成 19 年 5 月 9 日採録)

森 晃

1982 年生まれ．2005 年 3 月南山大学数理情報学部情報通信学科卒業．2007 年 3 月南山大学大学院数理情報研究科修士課程修了．同年 4 月富士ゼロックス株式会社入社．ソフトウェア工学，特に Web サービスやサービス指向アーキテクチャに興味を持ち，非同期メッセージ交換を行う Web サービスのパターン/アーキテクチャを研究テーマとする．

青山 幹雄（正会員）

1980 年岡山大学大学院工学研究科修士課程修了．同年富士通（株）入社．大規模ソフトウェア開発と管理，ソフトウェア工学の実践に従事．1986〜1988 年米国イリノイ大学客員研究員．1995 年 4 月〜2001 年 3 月新潟工科大学情報電子工学科教授．2001 年 4 月より現職．Web サービス，サービス指向アーキテクチャや組み込みソフトウェアを対象として，要求工学，ソフトウェアアーキテクチャ技術の研究・開発と教育・人材育成に取り組む．著書は『オブジェクト指向に強くなる』（共著，2003 年）ほか多数．1993 年情報処理学会研究賞受賞．電子情報通信学会，ソフトウェア科学会，IEEE，ACM 各会員．