2009－SE－163 (2)
2009／3／18

軽量メッセージ交換のモデルとパターンに基づく軽量サービス指向アーキテクチャ設計方法の提案

池崎 崇† 中道 上‡ 青山 幹雄‡

†南山大学 大学院 数理情報研究科 ‡南山大学 数理情報学部 情報通信学科

Web サービスの多くはメッセージ交換プロトコルに SOAP を用いるが，送信情報が少量の軽量メッセージ交換においてメッセージの複雑さと処理量の増大が問題となっている．この問題を解決する方法として，REST(REpresentational State Transfer)を適用した軽量メッセージ交換が提案されている．しかし，軽量メッセージ交換の構成要素とその組み合わせによるアーキテクチャの特性(属性)は多様であるため，軽量メッセージ交換に基づくサービス指向アーキテクチャ(軽量 SOA)の設計が困難となっている．本稿では，軽量メッセージ交換の構成要素と属性を構造と挙動の視点から抽出し，属性に基づく軽量 SOA のモデルとパターンを定義する．そして，定義したモデルとパターンに基づく軽量 SOA の体系的な設計方法を提案する．例題を用いて提案する設計方法の妥当性を示す．

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

Takashi Ikezaki†, Noboru Nakamichi‡, Mikio Aoyama‡

†Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
‡Dept. of Information and Telecommunication Engineering, Nanzan University

Although many of Web services use SOAP for a message exchange protocol, increase of the message complexity and message throughput become the problem in lightweight message exchanges. As a method to solve the problem, the lightweight message exchanges applied to REST (REpresentational State Transfer) are proposed. However, the components of the lightweight message exchanges and the attributes of the architecture by the combination of the components are diverse, the design of lightweight SOA, a class of SOA based on the lightweight message exchanges, becomes difficult. In the article, we extract the components and attributes of the lightweight message exchanges from a viewpoint of structure and behavior, and define the models and patterns of lightweight SOA based on the attributes. Based on the models and patterns that we defined, we propose a systematic design method of lightweight SOA. We prove the design method by the examples.

1. はじめに

現在，Web サービス[1]のメッセージ交換には，送信情報が少量の軽量メッセージ交換が多く利用される．一般に，Web サービスのメッセージ交換には SOAP が用いられるが，上記の場合においてメッセージの複雑さや処理量が増大する．この解決策として，REST (REpresentational State Transfer)[3]の技術要素を適用した軽量メッセージ交換が提案されている[4]．しかし，軽量メッセージ交換の構成要素とそれらの組み合わせによるアーキテクチャの特性(属性)は多様であり，軽量メッセージ交換に基づくサービス指向アーキテクチャを設計するためのモデルが未確立である．

本稿では，軽量メッセージ交換に基づくサービス指向アーキテクチャを軽量 SOA(Service-Oriented Architecture)と定義する．軽量 SOA を設計するためのモデルとして，構造と挙動の視点から軽量メッセージ交換の構成要素とその組み合わせによる属性を定義し，属性に関する構成要素を組み合わせたパターン(ABAP：Attribute-Based Architectural Pattern)を定義する．そして，定義したモデルとパターンに基づく軽量 SOA の体系的な設計方法を提案する．

2. 軽量 SOA とその設計問題

2.1. Web サービスのメッセージ交換における問題

現在，Web サービスのメッセージ交換には SOAP のメッセージプロトコルが利用され，リクエスタとプロバイダ間の二者間でメッセージ交換が行われている．図 1 に SOAP による Web サービスのメッセージ交換アーキテクチャを示す．

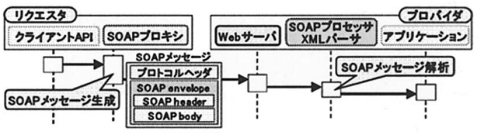

図 1：SOAP のメッセージ交換アーキテクチャ

近年，送信情報が少量の軽量メッセージを交換する Web サービスが増加傾向にある．この軽量メッセージ交換に SOAP のメッセージプロトコルを用いると，以下の問題が生じる．

(1) メッセージの複雑さの増大

SOAP メッセージは，送信すべき情報に加えて，送信に必要となる付加情報がエンベロープに内包される．そのため，メッセージのネストが深くなり，複雑化する．

(2) メッセージ処理量の増大

SOAP のメッセージ交換は，SOAP メッセージの生成と解析の処理が必要となる．また，上記したメッセージの複雑さが増大することで，メッセージ処理量も増大する．その結果，メッセージ交換に多くの時間が必要となるため，性能のボトルネックとなっている[8]．

2.2. メッセージ交換の問題に対する解決策

上記の問題に対して，REST の技術要素に基づくメッセージ形式の POX (Plain Old XML)を適用したアーキテクチャが提案されている[4]．図 2 に示す POX によるメッセージ交換のアーキテクチャを適用することで，上記の問題は以下のように改善される．

(1) メッセージの複雑さ

POX メッセージには SOAP エンベロープのような構造がないため，SOAP メッセージよりも簡素なメッセージ構造となり，メッセージの複雑さが減少する．

(2) メッセージ処理量

POX メッセージは XML データをそのまま送信するため，SOAP 構文に関する処理が不要となり，メッセージ処理量が減少し，メッセージ処理速度が向上する．

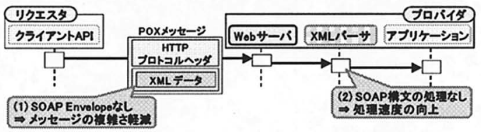

図 2:POX のメッセージ交換アーキテクチャ

2.3. 軽量メッセージ交換と軽量 SOA

本稿における軽量メッセージ交換とは，Web サービスで多く用いられる通信プロトコルの HTTP を用いたメッセージ記述量の少ないメッセージ交換と定義する．その例として，上記した POX がある．図 3 に示すように，軽量メッセージ交換に基づくサービス指向アーキテクチャを軽量サービス指向アーキテクチャ(軽量 SOA)と定義する．

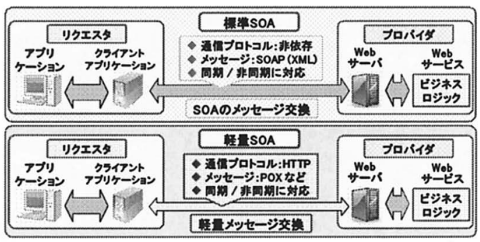

図 3:軽量メッセージ交換に基づく軽量 SOA

2.4. 軽量 SOA の設計問題

軽量 SOA の設計には，以下の問題がある．

(1) 軽量メッセージ交換の構成要素の多様化

図 4 に示すように，軽量メッセージ交換は軽量メッセージとメッセージ交換の構成要素を組み合わせることで実現される．この構成要素は多様であるため，それらの組み合わせによるアーキテクチャの特性(属性)も多様化する．これにより，構成要素の組み合わせが特性に対してどのように影響するかが不明確となり，特定の特性を満たす軽量 SOA の設計が困難となる．

(2) 軽量メッセージ交換の構成要素の相互運用性

軽量 SOA を実現する際に，軽量メッセージ交換の構成要素が多様であるため，構成要素の組み合わせによってはアーキテクチャの相互運用性が保証されない可能性がある．そのため，構成要素間の相互運用性とその範囲を考慮する必要がある．

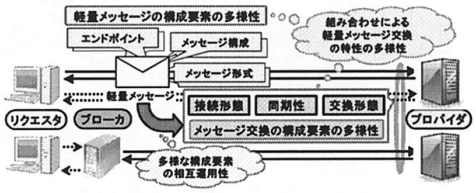

図 4:軽量 SOA の設計問題

3. モデルとパターンによる軽量SOA設計

3.1. 問題に対するアプローチ

本稿では，アーキテクチャの構成要素が持つべき特性を「属性」と定義する．上記の問題は，属性を考えることで統一的に扱うことが可能である．軽量メッセージ交換の構成要素の多様性を扱うために，軽量メッセージ交換に基づくアーキテクチャの構造と挙動における構成要素の持つべき特性を属性として捉える．軽量メッセージ交換の構成要素を組み合わせて非機能要求を満たす軽量 SOA を設計するためには，非機能要求に対応した属性を満たすように構成要素を組み合わせる必要がある．そのため，本稿では属性を満たす構成要素の組み合わせを定義し，それに基づいて軽量 SOA を設計する．

3.2. 軽量 SOA のモデルとパターン

本稿では，図 5 に示すように軽量メッセージ交換における構造の視点と挙動の視点から構成要素と属性を定義し，属性に基づく軽量SOAのモデルを定義する．また，モデルに基づいて属性と属性に関連する構成要素を抽出し，それらの組み合わせをパターンとして定義する[5]．

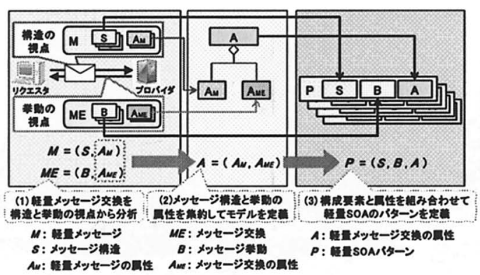

図 5:軽量 SOA のモデルとパターン

(1) 構造と挙動の視点による軽量メッセージ交換の分析

軽量メッセージ交換を構造の視点である軽量メッセージと挙動の視点であるメッセージ交換に分類し，構成要素と属性を分析する．

図 5 に示すように，軽量メッセージ(M)は，メッセージの構成要素であるメッセージ構造(S)と軽量メッセージの属性(A_M)の対として定義する．メッセージ交換(ME)は，メッセージ交換の構成要素であるメッセージ挙動(B)とメッセージ交換の属性(A_{ME})の対として定義する．

(2) 軽量メッセージ交換の属性を集約したモデル定義

構造の視点による軽量メッセージの属性(A_M)と挙動の視点によるメッセージ交換の属性(A_{ME})を集約し，軽量メッセージ交換の属性(A)としてモデルを定義する．

(3) 構成要素と属性の組み合わせによるパターン定義

軽量メッセージ交換の属性(A)に基づいて関連するメッセージ構造(S)とメッセージ挙動(B)を組み合わせ，軽量 SOA パターン(P)としてパターンを定義する．

3.3. モデルとパターンに基づく軽量 SOA 開発プロセス

図 6 に軽量 SOA の開発プロセスを示す．軽量 SOA の設計は，軽量 SOA のモデル化とパターン化，モデルとパターンに基づく軽量 SOA 設計という二つのプロセスから構成される．以下に軽量 SOA のモデル化とパターン化のプロセスについて説明する．モデルとパターンに基づく軽量 SOA 設計のプロセスは後述する．

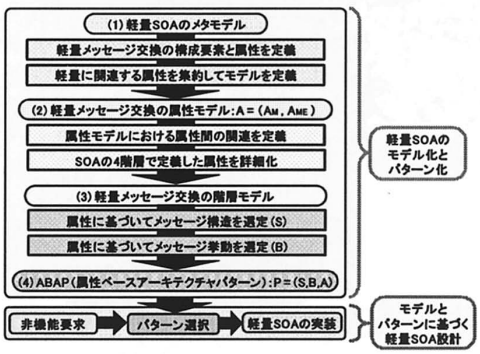

図 6：軽量 SOA の開発プロセス

(1) 軽量 SOA のメタモデル

SOA のメタモデルと SOA のメッセージ交換に軽量メッセージ交換を用いた軽量 SOA の関係を表した軽量 SOA のメタモデルを定義する．

(2) 軽量メッセージ交換の属性モデル

軽量メッセージ交換の構成要素と属性を構造と挙動の視点から定義し，軽量メッセージ交換の属性を集約して属性モデルを定義する．また，属性モデルの属性間の関連も定義する．

(3) 軽量メッセージ交換の階層モデル

SOA がメッセージ送受信に関与する 4 階層に分割可能であるため，階層モデルを用いて多くの属性に関与する属性を各階層で定義する．

(4) 属性ベースアーキテクチャパターン(ABAP)

属性を満たす構造と挙動の構成要素を選定し，それらを組み合わせた属性ベースのパターン(ABAP：Attribute-Based Architectural Pattern)を定義する．

4. 軽量 SOA のモデル化とパターン化

4.1. 軽量 SOA のメタモデル

図 7 に示すように，SOA はコンポーネントとコネクタによって構成されるアーキテクチャの拡張であり，サービスとメッセージから構成される．サービスはメッセージ交換により利用される．SOA のメタモデルを拡張し，メッセージ交換に軽量メッセージ交換を用いた軽量 SOA を定義する．また，軽量メッセージ交換の構成要素と属性を組み合わせたパターン(ABAP)を定義する．

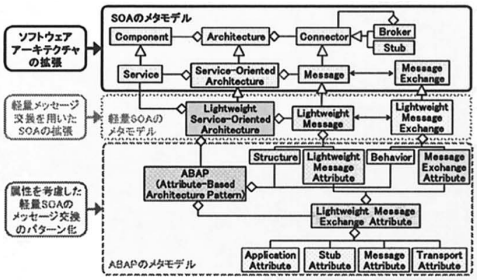

図 7：軽量 SOA のメタモデル

4.2. 軽量メッセージ交換の構成要素と属性

SOA におけるメッセージ交換は，メッセージとインタラクションにより構成される．本稿では，図 8 に示すようにメッセージに関する構造の視点とインタラクションに関する挙動の視点から軽量メッセージ交換の構成要素と属性を分析し，各構成要素と属性を定義する．

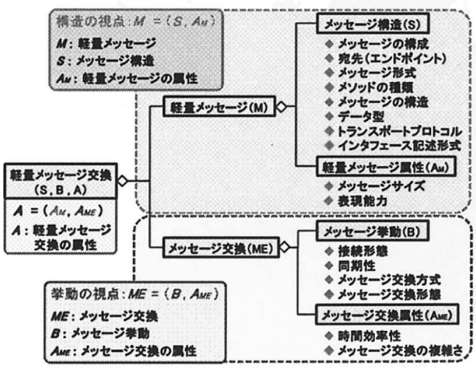

図 8：軽量メッセージ交換における構成要素と属性

(1) 構造の視点における構成要素

構造の視点における構成要素をメッセージ構造(S)とする．本稿では，図 8 に示す八つの要素を軽量メッセージの構成要素として抽出した．

軽量メッセージの構成要素と各構成要素のパラメータを表 1 に示す. これらのパラメータ選択により, 軽量メッセージが構成される.

表 1:軽量メッセージの構成要素とパラメータ

軽量メッセージの構成要素		パラメータ
メッセージの構成		ヘッダ+ボディ/ヘッダ+エンベロープ
宛先(エンドポイント)		処理プロセッサ/サービス
メッセージ形式		SOAP+XML/XML/JSON など(MIME タイプ)
メソッドの種類	アプリケーションメソッド	独自に定義(RPC)/HTTP メソッドを利用
	プロトコルメソッド (HTTP)	GET/POST/PUT/DELETE など
メッセージの構造		木構造を持つ/木構造を持たない
データ型		単純データ型/複合データ型
トランスポートプロトコル		トランスポート独立/HTTP に依存
インタフェース記述方式		WSDL/WSDL 2.0/WADL など

(2) 構造の視点における属性

構造の視点における属性を軽量メッセージ属性(A_M)とする. 本稿では, 以下に示す二つの要素を軽量メッセージの属性として抽出した.

1) メッセージサイズ

メッセージは送信すべき情報とメッセージを送信するために必要な情報によって構成され, これらの情報量を合わせることでメッセージサイズが決定する.

2) 表現能力

メッセージに包含される情報の記述能力であり, 表 1 に示す軽量メッセージの構成要素のパラメータ選択によって表現能力に違いが生じる.

(3) 挙動の視点における構成要素

挙動の視点における構成要素をメッセージ挙動(B)とする. 本稿では, 図 8 に示す四つの要素をメッセージ交換の構成要素として抽出した. メッセージ交換の構成要素と各構成要素のパラメータを表 2 に示す. これらのパラメータ選択により, メッセージ交換が実現される. メッセージ交換形態のパラメータは, 非同期のメッセージ交換の場合のみ選択する必要がある.

表 2:メッセージ交換の構成要素とパラメータ

メッセージ交換の構成要素	パラメータ
接続形態	直接(二者間通信)/間接(ブローカ利用)
同期性	同期通信/非同期通信
メッセージ交換方式	リクエスト/レスポンス(R/R)方式 / パブリッシュ/サブスクライブ(P/S)方式
メッセージ交換形態	Pull 型/Push 型

(4) 挙動の視点における属性

挙動の視点における属性をメッセージ交換属性(A_{ME})とする. 本稿では, 以下に示す二つの要素をメッセージ交換の属性として抽出した.

1) 時間効率性

ネットワークの遅延時間やメッセージの処理時間などのメッセージ交換に必要とされる時間やメッセージの処理件数に関する特性である.

2) メッセージ交換の複雑さ

メッセージの交換回数やメッセージ交換に関与するコンポーネント数などの特性である.

(5) アーキテクチャ属性

軽量メッセージ交換の構成要素には, 同期性などの軽量性には直接関連せずにアーキテクチャを構成する際に必要となる属性が存在する. 本稿では, この属性をアーキテクチャ属性と定義する.

4.3. 軽量メッセージ交換の属性モデル

構造と挙動の視点から抽出した属性を集約し, 各属性を構成する属性や要素の関係を定義した属性モデルを図 9 に示す. 各属性を構成する属性や要素には関連する構成要素が存在し, それらのパラメータ選択によって属性に差異が生じるため, 属性に関連する構成要素のパラメータ選択が重要となる.

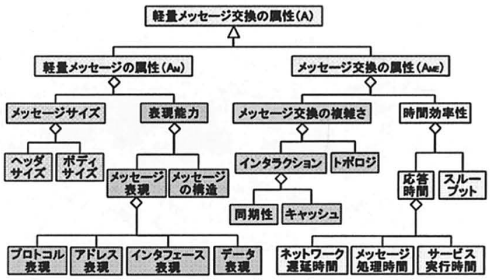

図 9:軽量メッセージ交換の属性モデル

(1) メッセージサイズ

メッセージヘッダとボディのサイズにより構成される.

(2) 表現能力

図 9 に示す四つの表現に基づくメッセージ表現とメッセージの構造により構成される.

(3) メッセージ交換の複雑さ

同期性とキャッシュに基づくインタラクションとトポロジによって構成される.

(4) 時間効率性

図 9 に示すように, ネットワーク遅延時間とメッセージ処理時間, サービス実行時間に基づく応答時間とスループットにより構成される.

4.4. 属性モデルにおける属性間の関連

図 9 に示す属性モデルの各属性は, 他の属性に関与しているが, 属性間の関係が不明確である. そこで, 図 10 に示すように属性モデルの各属性に関連する構成要素から属性間の依存関係を分析し, 属性間の関連を定義する.

4.5. 階層モデルにおける表現能力の属性の定義

図 10 に示した属性間の関連から, メッセージ表現能力の属性は他の多くの属性に関与していると考える. そこで, メッセージ表現能力の属性に関連する構成要素に基づいて軽量メッセージを構成する.

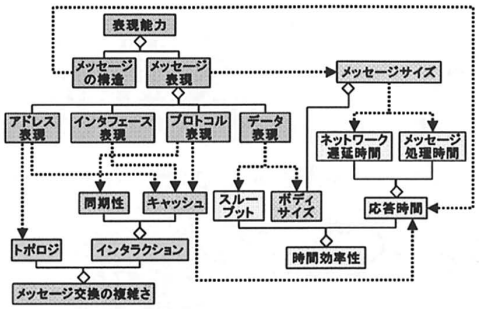

図 10:属性間の関連

本稿では，表 3 に示すように SOA のメッセージ交換が 4 階層に分割可能であることから，階層モデルを用いて各階層に関連するメッセージ表現能力の属性を定義する[7]. 表 3 に SOA のメッセージ交換における各階層の性質と各階層に関連するメッセージ表現能力の属性を示す.

表 3:各階層の性質と関連する表現能力の属性

階層(レベル)	階層の性質	表現能力の属性
アプリケーションレベル	サービス間の相互作用	データ表現
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	インタフェース表現
		メッセージ表現
メッセージレベル	メッセージルーティングの性質	アドレス表現
トランスポートレベル	トランスポートプロトコルに関する性質	プロトコル表現
		インタフェース表現

4.6. 階層モデルによる軽量メッセージパターンの選定

表 3 に示した階層モデルの各階層における表現能力の属性に関連する構成要素のパラメータを選定することで，軽量メッセージが構成される. 本稿では，表 4 に示す四つの軽量メッセージパターンを抽出する. 各パターンには固定と可変のパラメータが存在する.

(1) SOAP メッセージパターン

一般に Web サービスで利用されているメッセージプロトコルに基づいたメッセージパターンである.

(2) POX メッセージパターン

HTTP プロトコルを用いて XML データをメッセージ交換するメッセージパターンである. HTTP メソッドは GET と POST のみ利用可能とする. また利用可能なメッセージ形式は XML とする.

(3) REST メッセージパターン

REST の多くの技術要素を適用したパターンである. メッセージ形式は XML や JSON などのメッセージ形式が適用可能である. また，アプリケーションメソッドは HTTP メソッドのみであり，HTTP に依存する[11].

(4) Ajax メッセージパターン

Ajax の技術要素に基づいたメッセージパターンである. REST パターンと同様に，様々なメッセージ形式を適用可能である. また，エンドポイントは SOAP パターンと同様に処理プロセッサとなる.

表 4:階層モデルによる四つの軽量メッセージパターン

階層(レベル)	表現能力の属性	構成要素	軽量メッセージパターン			
			SOAP	POX	REST	Ajax
アプリケーション	データ表現	データ型	*	*	単純データ型	*
		メッセージ形式	SOAP +XML	XML	*	*
スタブ	インタフェース表現	インタフェース記述	WSDL	WSDL 2.0 WADL	WSDL 2.0	WSDL
		メソッド	*	*	HTTP メソッド	HTTP メソッド
	メッセージ表現	メッセージの構成	ヘッダ + エンベロープ	ヘッダ + ボディ	ヘッダ + ボディ	ヘッダ + ボディ
メッセージ	アドレス表現	エンドポイント	処理プロセッサ	サービス	サービス	処理プロセッサ
トランスポート	プロトコル表現	プロトコル	*	HTTP	HTTP	HTTP
	インタフェース表現	HTTP メソッド	POST/ (GET)	GET/ POST	*	*

4.7. 制約条件によるメッセージ交換モデルの選定

表 2 に示したメッセージ交換における各構成要素のパラメータを組み合わせることでメッセージ交換モデルが構成される. 現在，リクエスタとプロバイダの二者間でメッセージ交換する Web サービスが多いため，本稿では表 5 に示す三つのメッセージ交換モデルを抽出する. これらのモデルに対して，上記の四つの軽量メッセージパターンを用いる場合，メッセージ交換モデルを構成するパラメータの選択に制約がある.

表 5:メッセージ交換モデルの制約条件

メッセージ交換モデル			軽量メッセージパターン			
トポロジ	インタラクション					
接続形態	同期性	交換形態	SOAP	POX	REST	Ajax
直接	同期		○	○	○	○
	非同期	Pull 型	非同期 API の利用	非同期通信モデルが必要		プロバイダ側も非同期が必要
		Push 型				○

表 5 から，同期によるメッセージ交換はすべての軽量メッセージパターンで実現可能である. しかし，非同期によるメッセージ交換を実現するためには，各パターンにおけるメッセージ交換モデルのパラメータ選択の制約条件を満たす必要がある. Ajax パターンは，非同期通信に基づいたメッセージパターンと考えられるため，非同期のメッセージ交換が実現可能である.

5. 属性ベースアーキテクチャパターン(ABAP)

非機能要求を満たす軽量 SOA を体系的に設計するために，非機能要求を属性として捉え，属性を満たすように構成要素を組み合わせたパターンを属性ベースアーキテクチャパターン(ABAP : Attribute-Based Architectural Pattern)と定義する[2][6]. 表 6 に本稿で定義する ABAP スキーマとその具体例を示す.

(1) 軽量性の属性(A)

メッセージサイズや時間効率性などの属性.

(2) アーキテクチャ属性(A)

非同期やトポロジなど軽量性に関連しない属性.

(3) 属性の条件
属性に対する定性的条件や定量的条件.

(4) 構成要素の条件
アーキテクチャ設計に対して満たすべき制約条件.

(5) メッセージ構造(S)
属性の条件を満たす軽量メッセージパターン.

(6) メッセージ挙動(B)
属性の条件と構成要素の条件により軽量メッセージパターンを実現可能なメッセージ交換モデル.

(7) 効果
パターンによって得られる効果やその副作用.

表 6:ABAP スキーマと ABAP の具体例

ABAP スキーマ		ABAP の具体例
項目	内容	
軽量性の属性(A)	軽量性に直接関連する属性	メッセージサイズ
アーキテクチャ属性(A)	軽量性に直接関連しない属性	二者間通信
属性の条件	属性に対する前提条件	メッセージサイズが n バイト以下
構成要素の条件	構成要素に対する制約条件	ブローカの利用不可
メッセージ構造(S)	軽量メッセージパターン	REST パターン
メッセージ挙動(B)	メッセージ交換モデル	直接 / 同期 / R/R 方式 / -
効果	パターンの効果や副作用	HTTP に依存

6. 軽量 SOA 設計方法

6.1. モデルとパターンに基づく軽量 SOA 設計方法

上記のモデルとパターンに基づいて軽量 SOA を設計する. 軽量 SOA の設計プロセスを図 11 に示す.

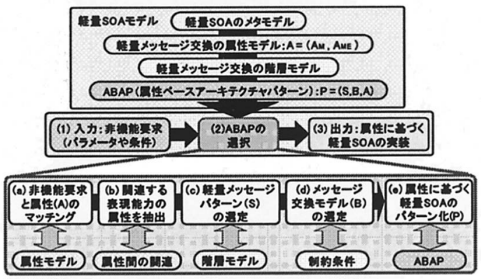

図 11:モデルとパターンに基づく軽量 SOA 設計プロセス

(1) 入力:非機能要求
満たすべき非機能要求に関するパラメータや条件.

(2) ABAP の選択
以下のプロセスから ABAP を選択.

(a) 非機能要求と属性(A)のマッチング
軽量メッセージ交換の属性モデルに基づいて非機能要求に対応する属性を抽出.

(b) 関連する表現能力の属性を抽出
属性間の関連に基づいて抽出した属性に関与する表現能力の属性を抽出.

(c) 軽量メッセージパターン(S)の選定
階層モデルに基づいて抽出した表現能力の属性と条件から軽量メッセージパターンの候補を選定.

(d) メッセージ交換モデル(B)の選定
選定された軽量メッセージパターンから制約条件に基づいてメッセージ交換モデル(B)を選定.

(e) 属性に基づく軽量 SOA のパターン化(P)
属性(A)を満たす軽量メッセージパターン(S)とメッセージ交換モデル(B)の組み合わせをパターン化.

(3) 出力:属性に基づく軽量 SOA の実装
軽量 SOA パターンに基づいて軽量 SOA を実装.

6.2. 軽量 SOA に対する非機能要求

軽量 SOA 設計に対する入力として, 軽量 SOA で満たすべき非機能要求がある. この非機能要求は, メッセージサイズや時間効率性などの軽量性に関するパラメータや条件とトポロジやインタラクションなどのアーキテクチャ属性に関するパラメータや条件を想定する.

6.3. ABAP 選択による軽量 SOA パターンの選定

入力として受け取った非機能要求に関するパラメータや条件に基づいて, 以下のプロセスで ABAP を選択し, 属性に基づく軽量 SOA パターンを選定する.

(1) 属性モデルによる非機能要求と属性のマッチング
非機能要求を満たす軽量 SOA を設計するには, 非機能要求に対応する属性を抽出する必要がある.

そこで, 図 12 に示すように, 非機能要求における軽量性の属性に関するパラメータを属性モデルにおけるメッセージサイズや時間効率性などの属性と対応付けることで, 非機能要求に対応する属性を抽出する.

同様に, 非機能要求におけるトポロジやインタラクションなどのアーキテクチャ属性に関するパラメータを属性モデルにおけるメッセージ交換の複雑さの要素と対応付けて, アーキテクチャ属性を決定する.

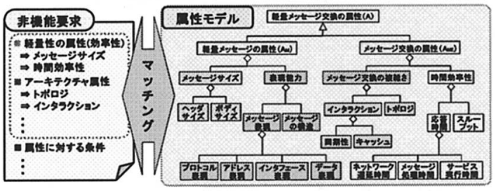

図 12:非機能要求のパラメータと属性のマッチング

(2) 抽出した属性に影響するメッセージ表現能力の属性
非機能要求のパラメータと属性のマッチングから抽出した属性を満たす軽量メッセージ交換を実現するために, アーキテクチャ全体を通してメッセージの送受信に関与する属性を満たす必要がある. そのため, 階層モデルを用いて軽量メッセージ交換の構成要素を選定するために, 抽出した軽量性の属性に影響を与えるメッセージ表現能力の属性を明確にする必要がある.

そこで, 図 10 に示した属性間の関連から, 抽出した軽量性の属性に影響するメッセージ表現能力の属性を抽出する.

(3) 階層モデルによる軽量メッセージパターンの選定
抽出したメッセージ表現能力の属性と非機能要求

の条件に基づいて，階層モデルを用いて軽量メッセージパターンを選定する．図 13 に示すように，階層モデルの各階層で定義したメッセージ表現能力の属性と関連する構成要素のパラメータを選択し，表 4 に示した軽量メッセージパターンから候補を選定する．選択可能なパラメータが複数存在する場合，あらかじめ設定した順序関係に基づいて条件などから選択する．また、選定したパターンの可変パラメータも適宜選択する．

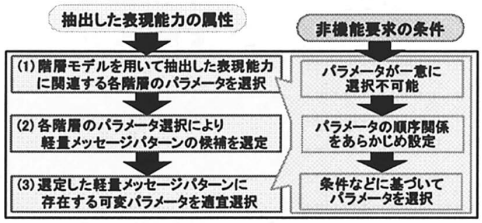

図 13:軽量メッセージパターンの選定プロセス

(4) 制約条件によるメッセージ交換モデルの選定

図 14 に示すように，選定した軽量メッセージパターンの候補に対して，実現可能なメッセージ交換モデルの候補を選定する．そして，アーキテクチャ属性に基づいてメッセージ交換モデルの絞り込みを行い，表 5 に示した制約条件からメッセージ交換モデルの候補を選定する．

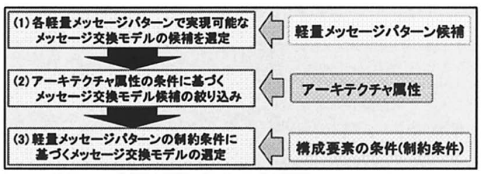

図 14:メッセージ交換モデルの選定プロセス

(5) 軽量 SOA パターンの選定

軽量メッセージパターンとメッセージ交換モデルの組み合わせにより，軽量 SOA の実現が可能となる．非機能要求を満たす軽量 SOA を設計するために，非機能要求に対応する属性を満たすように軽量メッセージパターンとメッセージ交換モデルを組み合わせて軽量 SOA パターンを定義する．図 15 に軽量 SOA パターンの選定プロセスをアクティビティ図を用いて示す．

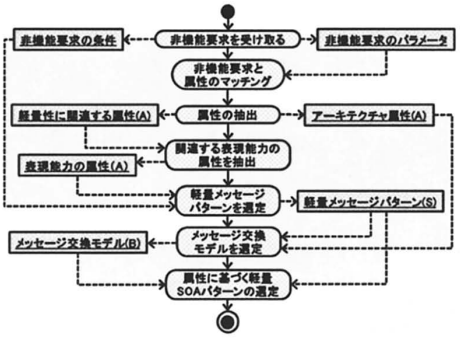

図 15:軽量 SOA パターンの選定プロセス

6.4. 属性に基づく軽量 SOA の実装

ABAP 選択による軽量 SOA パターンに基づいて，Apache Axis2[10]などを用いて軽量 SOA を実現する．

7. 例題による軽量 SOA 設計

図 16 に示すように，本稿で提案した設計方法を用いて，非機能要求を満たす軽量 SOA を設計する．

(1) 軽量SOAに対する非機能要求は，送信するメッセージサイズの削減と非同期メッセージ交換とする．

(2) ABAP の選択

(a) 入力として受け取った非機能要求から軽量性に関する属性のメッセージサイズとアーキテクチャ属性に関する属性の同期性(非同期)を抽出する．

(b) 図 10 に基づいて，メッセージサイズに影響するメッセージ表現能力の属性として，メッセージ表現とデータ表現を抽出する．

(c) 抽出したメッセージ表現能力の属性に関連する各構成要素のパラメータを属性の条件からあらかじめ設定した順序関係に基づいて選択し，図 16 に示す REST と Ajax の軽量メッセージパターンを選定した．その理由として，メッセージ表現ではメッセージサイズを削減するために，メッセージの複雑さが増加するエンベロープを用いずにメッセージ表現可能なパターンを候補とした．また，データ表現ではボディサイズを削減するために，XML を用いずに JSON などの記述量の少ないメッセージ形式を選択可能なパターンを候補とした．

(d) アーキテクチャ属性の非同期性と選定した軽量メッセージパターンの制約条件の両方を満たすメッセージ交換モデルが必要である．そこで，非同期 Push 型のメッセージ交換モデルを選定した．

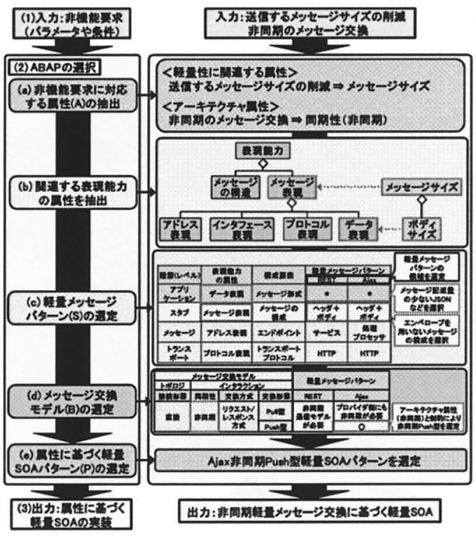

図 16:非同期軽量メッセージ交換に基づく軽量 SOA

(e) 非機能要求から，Ajax の非同期 Push 型の軽量 SOA パターンを選定した．

(3) 選定した Ajax の非同期 Push 型の軽量 SOA パターンから，非同期軽量メッセージ交換に基づく軽量 SOA のプロトタイプを Apache Axis2 を用いて実装し，他のパターンと比較してメッセージサイズの削減と送信時間の減少を確認した．

8. 軽量 SOA 設計方法の評価と考察

提案した属性に基づく軽量 SOA 設計方法に対して，以下の観点から評価と考察を行う．

(1) 軽量 SOA のアーキテクチャモデル

軽量 SOA の基礎となる軽量メッセージ交換の構成要素と属性が多様であり，それらを包括するモデルが未確立であったため，軽量 SOA を体系的に設計することが困難であった．そこで本稿では，軽量メッセージ交換の構成要素と属性を抽出し，属性に基づいて軽量 SOA モデルを定義した．この軽量 SOA モデルを属性に基づいて定義したことで，多様な軽量 SOA の構成要素を包括的に扱うことが可能となった．

(2) 属性に基づくパターンを用いた設計方法

軽量 SOA を体系的に設計する方法として，軽量メッセージ交換の属性を満たす構成要素の組み合わせをパターンとして定義し，パターンに基づいて軽量 SOA を設計する方法を提案した．提案した設計方法では，非機能要求とマッチングした属性のパターンに基づいて軽量 SOA を設計することで，非機能要求を満たす軽量 SOA を体系的に設計することが可能となった．

(3) 提案する設計方法の妥当性評価

例題として，メッセージサイズと非同期メッセージ交換に基づく軽量 SOA パターンを想定し，Apache Axis2[10]を用いて Handler の組み合わせを変更することで実装し，属性を満たす軽量 SOA を実現した．

これにより，Axis2 の Handler と軽量 SOA パターンの構成要素を対応付けることで，非機能要求を満たす軽量 SOA をシステマティックに実現可能となる．

9. 関連研究との比較と考察

従来の軽量 SOA は，REST と SOAP のアーキテクチャを決定する要素を比較した研究[9]などに基づいて実現しているが，軽量 SOA の設計方法が提示されていないため，軽量 SOA の体系的な設計が困難である．本稿では，属性に基づいて軽量 SOA のモデルとパターンを定義し，モデルとパターンに基づく軽量 SOA の体系的な設計方法を提案した．

また，非機能要求に基づくアーキテクチャパターンを選択してアーキテクチャを設計する研究(ABAS)[6]では，パターン選択により，非機能要求(属性)を満たすアーキテクチャ設計が可能となる．しかし，属性の一貫性が保証されていないと考える．本稿では，階層モデルにより属性を階層構造と関連付けることで，属性の一貫性が保証された軽量 SOA を設計可能である．

10. 今後の課題

(1) 非機能要求と属性モデルの定量的マッチング方法

メッセージ交換の効率化など抽象度の高い非機能要求に対応する属性を抽出する際に，マッチング精度を上げるための定量的なマッチング方法が必要である．

(2) 各構成要素のパラメータに対する優先度設定方法

非機能要求のパラメータや条件に基づいて軽量 SOA の各構成要素のパラメータを選択する際に，パラメータの優先順位を決定するために基準となる定量的な値を設定する必要がある．

11. まとめ

本稿では，軽量メッセージ交換の構成要素と属性の多様化により，軽量メッセージ交換に基づく軽量 SOA の設計が困難となっている問題に対して，属性に基づいて定義したモデルとパターンによる軽量 SOA の設計方法を提案し，例題を用いて妥当性を示した．

提案した設計方法では，軽量メッセージ交換の構造と挙動における構成要素の持つべき特性を属性として捉え，アーキテクチャ全体を通して属性を満たすように構成要素を組み合わせることで，属性を満たす軽量 SOA の体系的な設計が可能となる．

参考文献

[1] G. Alonso, et al, Web Services, Springer Verlag, 2004.

[2] P. Clements, et al, Evaluating Software Architectures, Addison-Wesley, 2001.

[3] R. T. Fielding , Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation, University of California at Irvine, 2000.

[4] 池崎 崇 ほか, REST を用いた軽量 Web サービスアーキテクチャの提案と評価, 情報処理学会第 69 回全国大会論文集(1), No. 3T-6. Mar. 2007, pp. 571-572.

[5] 池崎 崇 ほか, 軽量サービス指向アーキテクチャ設計方法の提案と評価, 情報処理学会第 71 回全国大会, Mar. 2009 [発表予定].

[6] M. Klein, et al, Attribute-Based Architectural Styles, Technical Report, CMU/SEI-99-TR-022, Oct. 1999.

[7] 森 晃 ほか, 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. 2007, pp. 2566-2577.

[8] A. Ng, et al, An Evaluation of Contemporary Commercial SOAP Implementations, Proc. of the 5th Australasian Workshop on Software and System Architecture, ACS, Apr. 2004, pp. 64-71.

[9] C. Pautasso, et al, RESTful Web Services vs. "Big" Web Services, Proc. WWW '08, Apr. 2008, pp. 805-814.

[10] S. Perera, et al., Axis2, Middleware for Next Generation Web Services, Proc. ICWS '06, Sep. 2006, pp. 833-840.

[11] L. Richardson, et al, RESTful Web Services, O'Reilly, 2007.