基幹系取引電文をサービス単位としたサービス再利用フレームワーク

福井 健太郎　藤田 桂一

Service Reuse Promotion Framework Focusing on Granularity of Core Banking Transactions

Kentaro Fukui and Keiichi Fujita

SOA（Service Oriented Architecture）環境を実現するに当たり，再利用可能なサービスの粒度をどのように定義するかが課題になることが多い．本論文では，既存の基幹系取引単位をサービス粒度として扱い，その粒度にてサービス化を促進するフレームワークについて提案する．本フレームワークでは，基幹系電文定義を用いてサービスのメッセージ定義を生成し，電文内の各項目とサービスで利用するメッセージ定義内の項目を連動させることにより，サービス化を実現することが可能となる．本フレームワークを適用することにより，短期間かつ低コストでサービス化を行うことができ，サービスの再利用を促進する環境を実現できる．

In this paper, a framework for promoting a service reuse environment by identifying service granularity as the "transaction size of the core banking system" will be discussed. This framework focuses on the transaction unit of the core banking system, and successfully exports message definitions from the core banking system. By generating the service definitions from the message definitions, this framework enables the environment for automatic generation of a service code for the core banking system. By implementing this framework in a project, the results show that this framework successfully enables the environment to promote the extraction of service from the existing core banking system.

Key Words & Phrases：SOA，エンタープライズ・アーキテクチャー，Web サービス，WebSphere® Message Broker
SOA, Enterprise Architecture, Web Service, WebSphere Message Broker

1. はじめに

SOA（Service Oriented Architecture）の概念は成熟・実装の段階に達しており，SOA を踏まえた数多くのソリューション・デザインが実際に検討されるようになった［1］．そのような中，実案件として SOA を推進し，SOA を導入することにより大きな効果を上げるためには，サービスの再利用の推進が不可欠である［2］といわれている．再利用を推進することにより，似た業務アプリケーションを複数作ることがなくなり，IT 投資の際の ROI（Return On Investment）を向上することが可能となる．しかし，実際に SOA レベルでの再利用を推進し，それに伴い大きく ROI 向上を実現した例はいまだ少なく，サービスの再利用環境の実現は SOA 戦略の成功の鍵を握っていると考えられる．

サービス再利用を推進するためには，そのサービスの粒度をどのように定義するかが大きな課題となる［1］．IBM では SOMA（Service-oriented modeling and architecture）［1,3,4］に代表される各種手法が用意されており，サービスの粒度の定め方のガイドラインが実証されつつある．しかし，SOMA での考え方は CBM（Component Business Modeling）［5］や Goal Service Modeling など，トップダウンのアプローチを主体に考えられている．トップダウンのアプローチは企業全体としての視点でシステムを見直すことができる反面，さまざまな面で変革を行う必要があり，場合によっては適用が難しいケースも考えられる．

サービス・モデリングをボトムアップ・アプローチで実施する手法として，仮想サービス・リファクタリング手法［6］も検討されている．このアプローチは，日本の大規模金融機関など，すでにある程度の業務 IT 資産を有している環境下において，Martin Fowler の "Refactoring"［7］の概念を応用したものであり，本手法を用いることによりトップダウン・アプローチを利用せずにサービスの候補・粒度を決められる．しかし本手法は，

提出日:2008年9月8日　再提出日:2009年2月16日

現行システムのサービス粒度を再構築することを前提にしており，複数のベンダーが混在する環境においては，サービス候補を洗い出し，それに合わせて再構築を推進することは現実的に難しい場合がある．

そこで本論文では，サービスの粒度を残高照会・口座明細照会のような，基幹系一取引単位と定義し，取引単位でサービス化するフレームワークを提案する．本フレームワークを用い，サービス粒度を小さくかつ明確にすることにより，他ベンダーが構築した基幹系システムの既存業務を，短期間・低コストでサービス化することも可能となる．また本フレームワークを用いることにより，SOA 製品に対する知識を有さない基幹系業務開発者が，WSDL（Web Services Description Language）を用いて基幹系業務をサービス化することができるため，基幹系業務のサービス化を大きく推進することが可能となる．本論文では，残高照会電文のような基幹系取引電文の構造を用いて，サービスのメッセージ定義および基幹系電文変換定義を生成した一実装例を示し，フレームワークを実際に金融機関の基幹系システムに適用した結果について考察する．

以降，第 2 章では基幹系取引単位を基準としたサービス化フレームワークの概念について述べ，各コンポーネントの内容について述べる．第3章では本フレームワークの具体的な実装例について述べ，第4章にて本フレームワークを金融機関の勘定系取引に適用したことによる効果について述べた後，第5章にてまとめる構成となる．

2. 基幹系取引単位を基準としたサービス化フレームワークの概念

本フレームワークの概念を示したのが図 1 である．本フレームワークの実行環境としては，サービス利用者（Requester）とサービス提供者（基幹系システム）の間に，ESB（Enterprise Service Bus）およびハブ・システムを配置することを前提としている．そして，基幹系業務の取引電文の定義をサービス・ジェネレーターが取引電文単位で読み込み，サービス・ジェネレーターが基幹系取引定義を元に，Requester が利用する WSDL ファイル，ESB で利用する実行モジュール，ハブで利用する実行モジュールを出力する．そして，ジェネレーターから出力された定義を各実行環境内のコンポーネントに反映することにより，基幹系取引をその取引の単位でサービス化する環境を実現する．以下，それぞれのコンポーネントの役割について述べる．

2.1 実行環境構成

本フレームワークは，Requester・基幹系システム間に ESB およびハブを配置することを前提としている．ESB

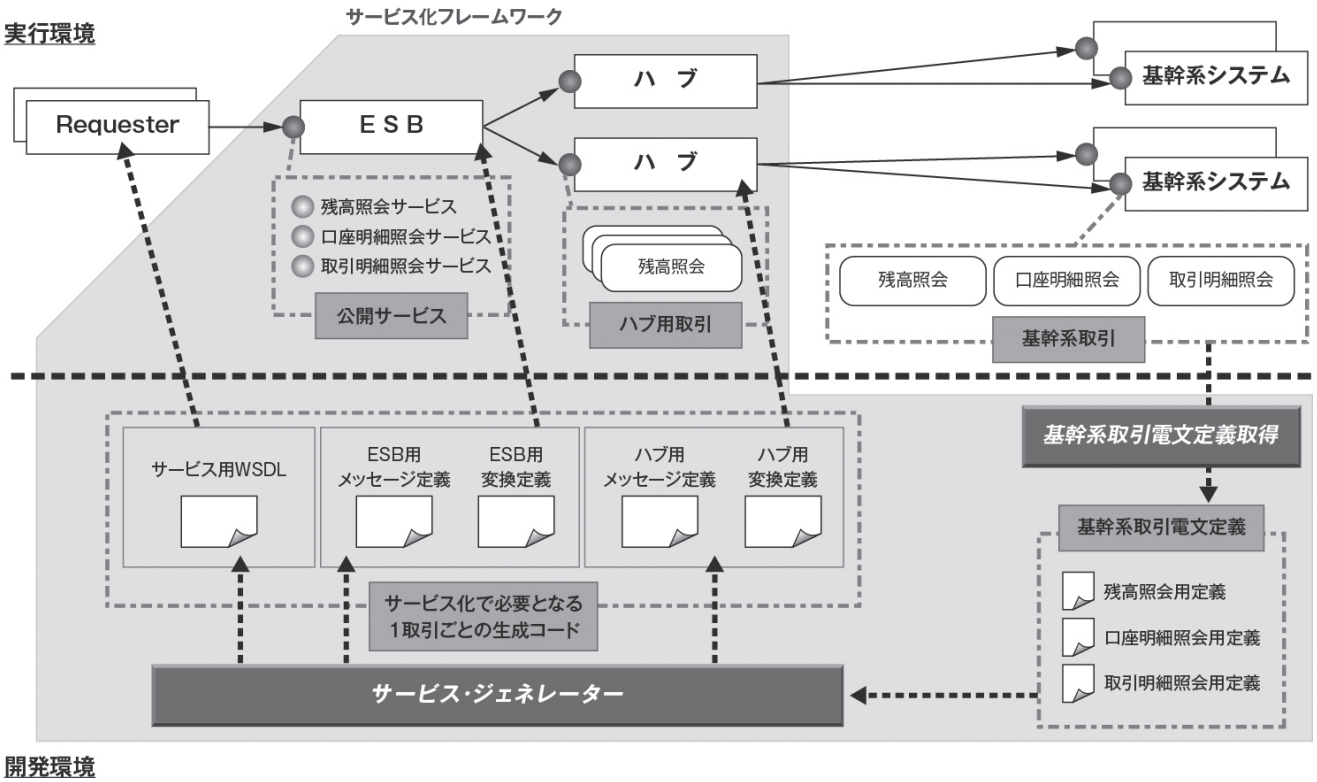


図1. 本フレームワークの概念図

では主にSOAP/XML形式のメッセージを受け付け，それをハブが受信できる形式（一般的には固定長電文）に変換することを想定しており，ハブではESBから受け取った電文を各種基幹系システムに特化した電文形式に変換する．

ここで，本フレームワークをESBとハブで分離する設計とした理由は，ESBに基幹系接続関連のロジックを持たせた場合，本来のバスとしての役割を果たしにくくなると考えたためである．また，IBMが発表している「オンデマンドを支えるSOA導入への10の提言[2]において，金融系の大規模システムでシステム間連携を実現しているハブについてはそのまま再利用することを推奨していることも考慮した．

2.2 基幹系取引電文定義取得

本フレームワークは，サービスの単位を基幹系取引の単位と対応させることを前提としている．基幹系取引の単位とは，「基幹系取引における一つのINとOUTの電文構造およびその間に含まれる業務ロジックを組み合わせたもの」と定義される．基幹系取引の中でも勘定系の取引を例に挙げると，残高照会，口座明細照会，取引明細照会などを一つの取引単位として扱う．なお，金融系の取引では，口座明細照会の中でも，普通口座明細照会，当座口座明細照会，定期口座明細照会などの複数の種類が含まれることがあるが，本フレームワークでの取引単位は，このそれぞれを一つの取引単位（=サービスの単位）として定義する．

本フレームワークでは，これらの基幹系取引をサービス化することを目的としており，本フレームワークの中で，各基幹系システムの各取引にて用いられる電文定義を取得する仕組みが不可欠となる．

2.3 サービス・ジェネレーター

本フレームワークには，基幹系取引電文定義を元にサービス化用各種リソースを生成するサービス・ジェネレーターが必要となる．基幹系取引をサービス化するに当たり，サービス・ジェネレーターに求められる要件は以下の通りである．

- 入力として基幹系取引電文定義を用いる
- ハブ用メッセージ変換定義（メッセージ定義・変換定義），ESB用メッセージ変換定義（メッセージ定義・変換定義），WSDLの3種類の定義を出力する
- 基幹系取引電文定義・ハブ用定義・ESB用定義の3つを参照し，それぞれの電文変換定義を柔軟

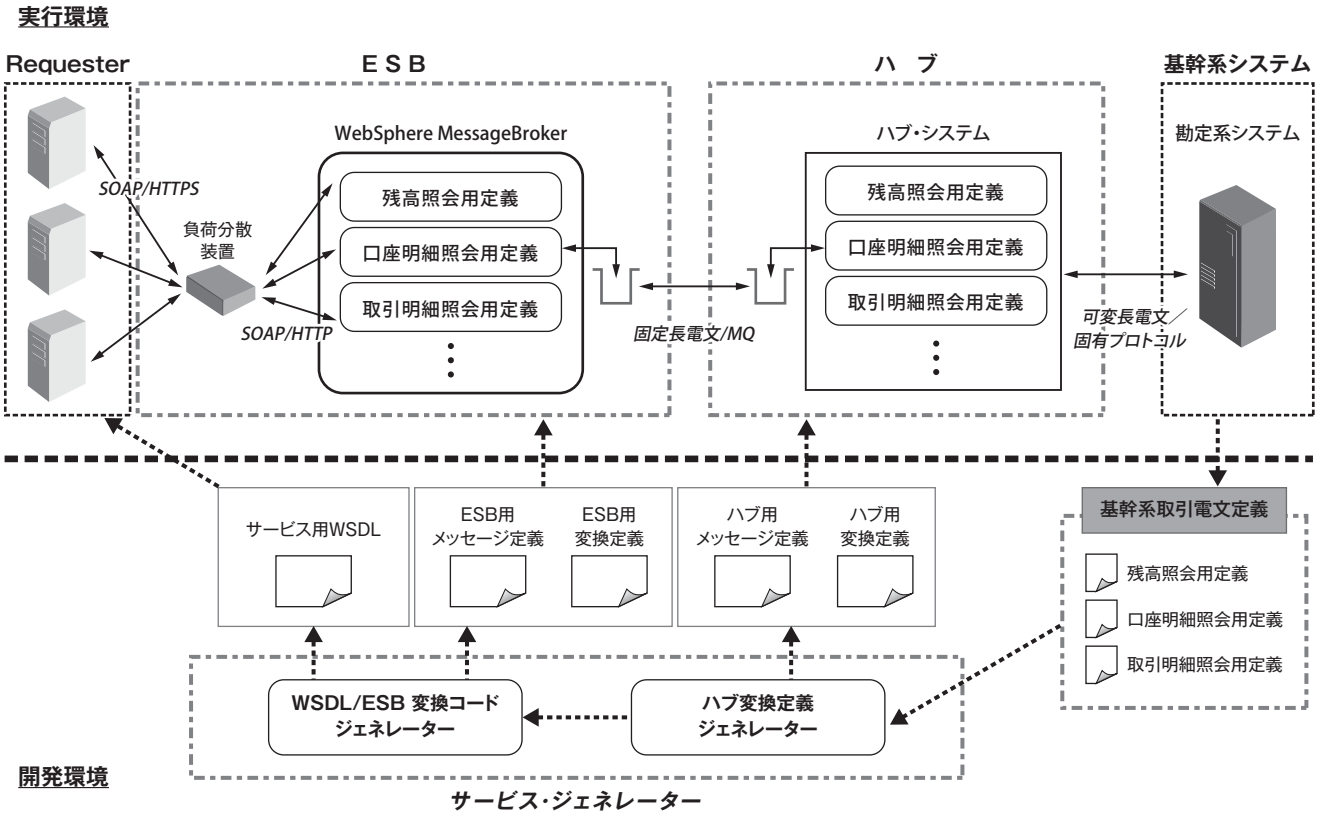


図2. 本フレームワークの実装例

にかつシンプルに実装するインターフェースを持つ

3. サービス化フレームワークの具体的実装例

本章では，前章で述べた概念に基づき，実際に大手金融機関にて構築したサービス化フレームワークの実装例を紹介する．

図２が，今回実際のプロジェクトで本フレームワークを利用した具体例である．この環境では，ESB として WMB（WebSphere Message Broker）を核としたシステムを構築し，ハブを介して勘定系システムに接続する構成となる．また，サービス・ジェネレーターとして，勘定系の電文定義を取り込み，ハブ用変換定義を生成するハブ変換定義ジェネレーター，ハブ変換ジェネレーターから出力された定義を取り込み，WSDL および WMB 用実行モジュールを生成する WSDL/ESB 変換コード・ジェネレーターの 2 つで構成される．以下，サービス化フレームワークの中心となるサービス・ジェネレーターに焦点を当てて紹介し，本フレームワークの実際の利用の流れについても説明する．

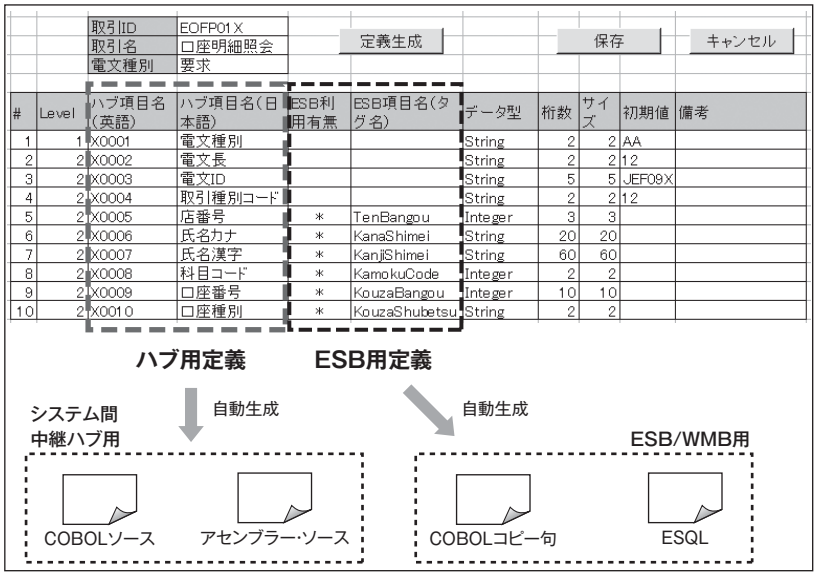


図3. ハブ変換定義ジェネレーター

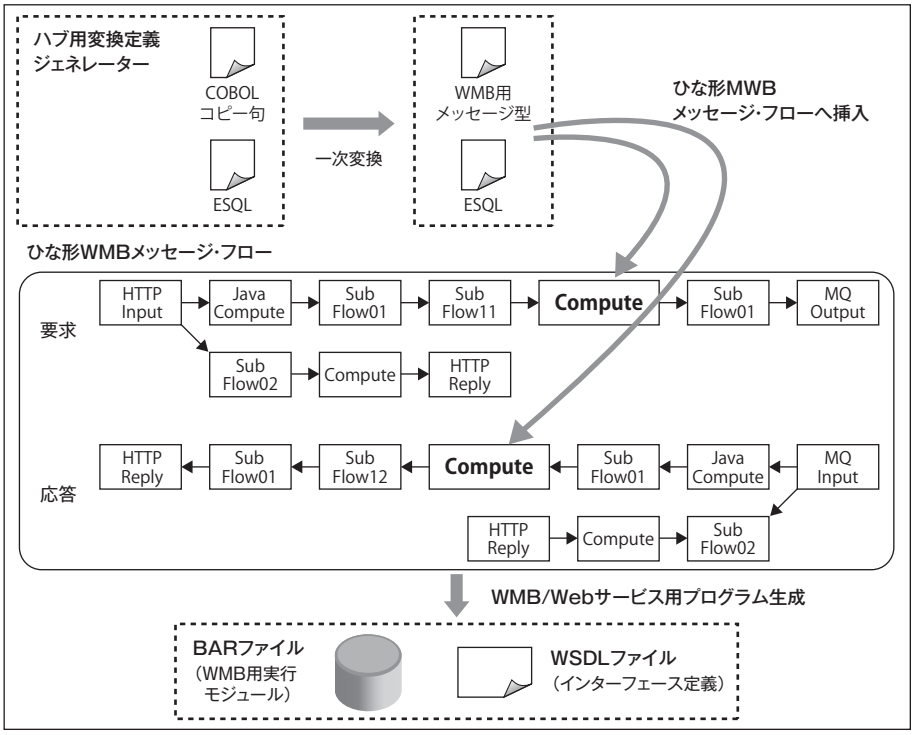


図4. WSDL/ESB変換コード・ジェネレーター

3.1 ハブ変換定義ジェネレーター

今回のプロジェクトでは，勘定系取引電文定義の取り込み，ハブ用変換定義の生成，および WSDL/ESB 変換コード・ジェネレーター用の定義生成用に，図３のような Excel® ベースのハブ変換定義ジェネレーターを開発した．

ハブ変換ジェネレーターは，30 画面程度から構成されており，ESB メッセージ定義，ハブ電文定義，基幹系システム電文定義の 3 者間でどのように電文変換を行うかを定義する．そして，基幹系システムの電文変換用に Excel 上に定義した電文変換定義から，ハブ上で稼働するメッセージ・フォーマッター用の COBOL ソース・コードをジェネレートする．このメッセージ・フォーマッターは，レガシー基幹系システムの固有の電文形式を隠ぺいし，他システムから使いやすいシステム間連携電文（ハブ形式電文）として汎用化を行う．また，電文変換実行時のステップ数をできるだけ小さくするために，Excel 上の電文定義に基づいて，あらかじめ初期値／固定値をセットしたアセンブラー・コードを生成し，それをロード・モジュール形式に変換した上で，実行時にハブ上の電文変換エリアに初期値としてセットするなどの工夫も行っている．

このように，ハブ変換ジェネレーターは，Excel 上に定義した電文変換定義に基づいて，基幹系システムの電文を，ハブ形式電文を挟んで，ESB メッセージに変換するための各種定義や変換プログラム用ソース・コードなどを整合性のとれた形でジェネレートする.

3.2 WSDL/ESB変換コード・ジェネレーター

ハブ変換定義ジェネレーターで出力された ESB メッセージ定義およびメッセージ変換定義を用い，WSDL および ESB 用変換プログラムを生成するために,本プロジェクトでは Eclipse Plugin によるジェネレーターを開発した.

本ジェネレーターは，ESB の機能と密接に連携をとっており，ESB が持つロギング機能や障害対応機能を共通化された機能をひな形メッセージ・フローとして最初に取り込む. このひな形の部分は ESB 構築時に厳正にテストされているコードであり，このひな形を利用することにより，ESB 上のプログラムの品質を確保する. そして，**図 4** のように，ハブ変換定義ジェネレーターで生成された ESB メッセージ定義を業務ごとの WMB メッセージ・フローとして変換し，ひな形のメッセージ・フロー上へ挿入することにより，WMB 用実行モジュールおよび WSDL ファイルを生成する.

3.3 本フレームワークの利用手順

本フレームワークを用いることにより，SOA 製品に対する知識を有さない勘定系業務開発者が，勘定系取引の単位で勘定系業務をサービス化することができ,勘定系業務のサービス化を大きく推進することが可能となる. しかし，勘定系業務開発者がスムーズに作業を行うためには，上述した各種ジェネレーターの利用手順を明確に提示する必要がある.

そこで本プロジェクトでは,**表1**のように手順・担当者を明確にし，設計・開発・テスト・フェーズまでを考慮した利用手順を確立した.

4. 本フレームワークを用いることによる効果

本フレームワークの一番の特長は，基幹系取引単位でサービス化を行うことにより，基幹系取引を半自動的にサービス化することが可能である点にある. これにより基幹系取引のサービス化を大きく促進することが可能となる.

SOA を推進するに当たり，サービスの粒度を定めた上でサービス化を推進することの重要性については，頻繁に論じられている［1］. しかし，実際の作業レベルで，サービス化作業を誰が担当するかについて考慮がされていないことが多々ある. IBM は ESB を実現する製品として，WebSphere ESB［8］, WebSphere Message Broker［9］, DataPower®［10］などを発表しているが，これらの製品を使いこなすためには専門のスキルが必要となり，これを有する開発者の数には限りがある. その結果，既存のサービス化のプロセスでは，ESB 上の変換アプリケーションなどを別途開発する必要が出てく

表1.本フレームワークの利用手順

Step	作業名	作業内容	担当			開始目安	資料URL
			基幹系	ESB	接続元		
1	電文レイアウト決定	電文レイアウトを、基幹系担当者・Requester 側担当者間で決定	○	-	△	S/I Xカ月前	http:// xx.vv/ab…
2	基幹系プログラム開発	基幹系プログラムを開発（基幹系プログラムを新規開発する場合）	○	-	-	S/I Xカ月前	http:// xx.vv/ab…
3	基幹系プログラム・アクセス・コード	基幹系プログラムと連携するハブ定義をハブ変換定義ジェネレーターを用いて開発	○	-	-	S/I Xカ月前	http:// xx.vv/ab…
4	WSDL/ESB変換コード開発	WSDL/ESB 変換コード・ジェネレーターを用いて開発	○	-	-		http:// xx.vv/ab…
5	WSDLの受け渡し	生成された WSDL ファイルを受け渡す	○	-	△		http:// xx.vv/ab…
6	Requesterコード開発	Requester 用の接続コードを開発	-	-	○	S/I Xカ月前	http:// xx.vv/ab…
7	開発系疎通テスト	開発した Requester コードを使い、基幹系システムとの接続試験を開発環境にて実施	△	△	○		http:// xx.vv/ab…
8	開発系障害テスト	開発した Requester コードにて障害時の動作について確認	△	△	○		http:// xx.vv/ab…
9	本番系疎通テスト	本番環境にて Requester コードを使い、基幹系システムとの接続試験を実施	△	△	○	S/I Xカ月前	http:// xx.vv/ab…
10	本番系障害テスト	本番環境にて Requester コードを使い、基幹系システムとの障害試験を実施	△	△	○		http:// xx.vv/ab…
11	本番系性能テスト	本番環境にて Requester コードを使い、基幹系システムとの性能試験を実施	△	△	○		http:// xx.vv/ab…
12	プログラム配置	各システムにて最終リリース・プログラムを配置	-	○	-	S/I Xカ月前	http:// xx.vv/ab…
13	サービスイン	サービスイン	△	△	○	-	http:// xx.vv/ab…

るため，各種接続システムはESBに接続するために別途コストがかかることを憂慮し，ESBを経由せずに直接接続するなど，サービス化が推進されないこととなる．環境によっては，ESBを導入し，ESB上のアプリケーションを開発する専門チームを構成するところもある．しかし，サービス化するためには，各業務の内容をある程度理解した上でサービスの粒度を決定する必要があり，効率的なサービス化推進に結び付きにくい．

そのような中，実プロジェクトで複数の業務で共通的に利用される即時性取引（残高照会・口座明細照会など）に本フレームワークを適用したところ，各種ジェネレーターがESBやハブ用の変換コードを自動的に生成するため，各基幹系業務担当者によって高品質なESB接続コードを開発することが可能となることが確認された．その結果，各基幹系システム上の取引のサービス化が進み，ESBのサービスを利用するRequesterの数も一定期間経過後は急速に増えることが分かった．以下，このサービス化プロセスの変革によってもたらされた本フレームワークの効果について，サービス化コスト・品質・サービス化推進の観点で述べる．

(1) サービス化におけるコスト効果

既存業務をサービス化する上で，本フレームワークを用いることにより，開発期間短縮，開発工数削減を実現することが可能となった．これは，本フレームワークを利用することで，作業者はサービス化に当たってSOA製品の専門知識を問われなくなり，ジェネレーターなどのツール操作を理解するのみで開発が行えたからである．また，ESBに接続するまでの手順も明確にしたことにより，ESBを保守する部門に作業がほとんど発生しないため，保守・運用管理コストも削減することが可能となった．

(2) サービス化における品質の向上

本フレームワークのジェネレーターを用いることにより，あらかじめテストされたロギング機能や障害対応機能などを全取引で共通利用することが可能となった．その結果，一般的に考慮不足になることが多い各種非機能要件を充足した，高品質なサービス化コードを生成することが可能となった．また，あらかじめ性能面で最適化された共通機能を利用するため，一定の性能品質を確保することができる．今回のプロジェクトでは，勘定系の即時性取引にて，ESBを使用しない場合（ハブに直結する場合）と比べても遜色のない性能を実現できた．

(3) サービス化の推進

本フレームワークを利用したプロジェクトでの，基幹系取引数の増加および本番環境へ接続するRequesterの数の増加の関係を示したのが，以下の図５である．

この図５に示すように，本フレームワークを利用したことにより，ESBを経由する取引およびRequesterの数が，ある一定期間経過後に急激に増加したことが分かる．これは本フレームワークを利用し，基幹系取引をサービス化するコストを削減し，かつ高品質なサービス化コードを生成する仕組みを構築したことで，各基幹系開発者およびRequester開発者が本環境の有用性を認識した結果であると考えられる．ESBを構築し，最初に数個のサービスを登録したが，それ以降増加しないという事例が存在する中，本フレームワークを利用することによる有用性がこの結果より実証されている..

5. おわりに

本論文では，再利用可能なサービスの粒度を「基幹系一取引単位」と定義し，サービス化するフレームワークを提案した．そして基幹系取引電文の構造を用い，サービスのメッセージ定義および基幹系電文変換定義を生成する実装例を紹介した．

本フレームワークを用いることにより，以下の利点を享受できることを確認した．

- サービス粒度を小さく，かつ明確にしたことにより，基幹系システムの業務を，高品質・短期間・低コストでサービス化することが可能であること

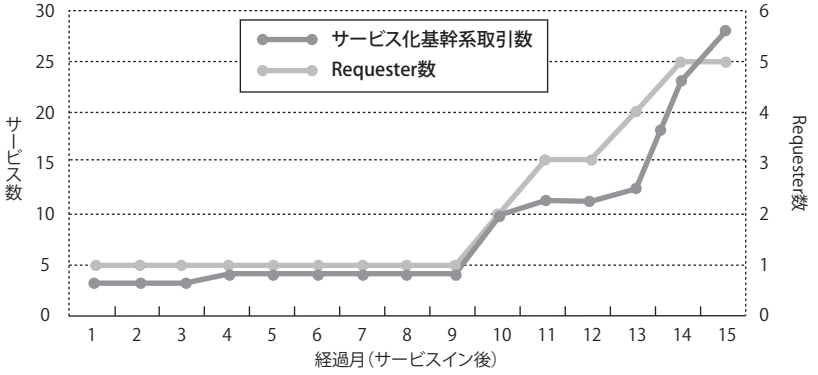


図5. ESB上の取引増加傾向

- SOA に関する高度な知識を有さない開発者が，基幹系取引の単位でサービス化することができ，既存業務・新規業務のサービス化を大きく推進することが可能であること
- ESB へサービスを公開する勘定系システム上の取引が，一定期間経過後に急激に増えるとともに，ESB のサービスを利用する Requester の数も急激に増え，サービス再利用を促進する環境を構築できること

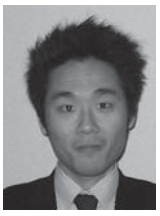
本フレームワークは，非ベンダー依存であり，ほかの基幹系システムでサービス化を推進する場合にも適用可能である．また，既存の基幹系取引電文をサービス化することにより，レガシー・トランスフォーメーションなどの推進も実現できると考えられる．さらに，今後 SOMA などのトップダウン・アプローチの中で既存資産の分析・サービス化に本フレームワークを応用することも可能であると考える．

謝辞

本論文を記述するに当たり支援していただいた佐藤光進氏，中西三枝子氏，渡邊悠貴氏，山口顕治氏，住谷司氏，井上理氏，山崎聡氏，井上卓氏，佐藤令二氏，青木雄氏，遠藤正幸氏，武井安雄氏，北沢強氏，豊村明彦氏，また各種情報を提供していただいた高橋辰徳氏，山根雅彦氏，大久保匡幸氏，齋藤幸二郎氏にあらためて深謝いたします．

参考文献

［1］ 中丸 毅 : “SOA を実現するための手法とアプローチ —ビジネスモデリングからサービスモデリングそして実装へ— ,” http://www.ibm.com/jp/provision/no51/pdf/51_article2.pdf（2008）.
［2］ “銀行システムにおける SOA の考え方,” http://www.ibm.com/jp/finance/solutions/fsn/mar2006_soa.html（2008.08.24）.
［3］ “SOA terminology overview, Part 3: Analysis and design,” http://www.ibm.com/developerworks/webservices/library/ws-soa-term3/（2008.08.24）.
［4］ 北澤 治郎 , 増広 順一 , 阿部 渉 : “CIR（Component Infrastructure Roadmap）—企業の将来像を描くIT 基盤ロードマップ策定手法—,” PROVISION, No.58, pp.48-53（2008）. http://www.ibm.com/jp/provision/no58/pdf/58_article5.pdf
［5］ “CBM,” 金融ソリューション NEWS Vol.32, 2004 年 10 月号
［6］ 山下眞澄 : “SOA を活用した Rapid Enterprise Renovation（SOA RER）のアーキテクチャ,” PROVISION, No.54, pp.72-79（2007）. http://www.ibm.com/jp/provision/no54/pdf/54_paper1.pdf
［7］ Martin Fowler, “Refactoring,” 児玉公信 訳, ピアソン・エデュケーション, ISBN 4-89471-228-8
［8］ “WebSphere Enterprise Service Bus, ” http://www.ibm.com/software/integration/wsesb/（2008.08.24）.
［9］ “WebSphere Message Broker,” http://www.ibm.com/software/integration/wbimessagebroker/（2008.08.24）.
［10］ “WebSphere DataPower SOA Appliances,” http://www.ibm.com/software/integration/datapower/（2008.08.24）.



日本アイ・ビー・エム株式会社
GBS AIS 金融ソリューション
ITスペシャリスト
博士（工学）

福井 健太郎 Kentaro Fukui

［プロフィール］
2003 年日本 IBM 入社．IT スペシャリストとして，Web 系フレームワーク・各種先進技術を用い，主に金融のお客様向けプロジェクトを担当．2004 年 SAINT にて IEEE Computer Society Best Paper Award 受賞．2005 年 IBM Redbook® 記述．2006 年博士（工学）取得．

日本アイ・ビー・エム株式会社
GBS AIS 金融アプリケーション
マネージャー
シニアITアーキテクト

藤田 桂一 Keiichi Fujita

［プロフィール］
1988 年日本 IBM 入社．金融営業推進配属．1991 年より金融機関のお客様におけるシステム開発プロジェクトに従事．IT アーキテクトとして，情報系システム，大規模 DB システム，基幹系システム間の連携 HUB システム，ALM システムなど金融分野におけるシステム／アプリケーションの設計・開発を中心に担当．