SOA に基づく SaaS インテグレーションアーキテクチャの提案

M2008MM018　近藤　洋介

指導教員：青山　幹雄

1. はじめに

本稿では，SOA(Service-Oriented Architecture)に基づく SaaS インテグレーションアーキテクチャを提案する．特に，SOA のプロセス連携に着目し，ビジネスプロセスを用いることでオンプレミスアプリケーションと SaaS アプリケーション間のデータ連携の実現を目指す．さらに，Force.com，Apache Axis2，Oracle BPEL Process Manager を用いて提案アーキテクチャの妥当性を評価する．

2. SaaS インテグレーション

2.1. SaaS インテグレーションの定義

SaaS アプリケーションを利用する企業の多くは図 1 に示すようなハイブリッド型のシステム構成をとる．ハイブリッド型のシステム構成とは，基幹業務システムと支援業務システム両方をオンプレミスアプリケーションで構成していた従来のシステム構成から，支援業務システムのみを SaaS アプリケーションに移行した構成となる．このため，オンプレミスアプリケーションと SaaS アプリケーション間のデータ連携が重要となる．

本稿では，ハイブリッド型のシステム構成を想定した，オンプレミスアプリケーションと複数の SaaS アプリケーション間のデータ連携を SaaS インテグレーションとして定義する．現在，SaaS インテグレーションの方法は確立されていない．

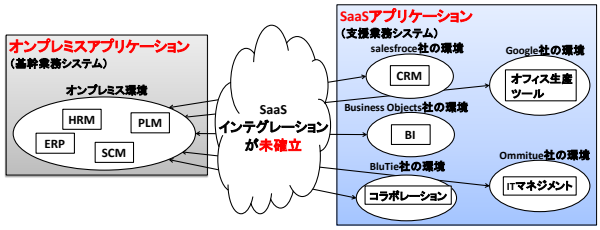

図 1　ハイブリッド型のシステム構成

2.2. 現状の SaaS インテグレーションモデル

現状の SaaS インテグレーションのモデルを図 2 に示す．図 2 のモデルでは，コネクタと Web サービスの技術を用いることで，オンプレミスアプリケーションと SaaS アプリケーション間のデータ連携を実現している．また，SaaS アプリケーション内部では，アプリケーションプロビジョニングの技術を用いることで，当該テナントのデータベースへのアクセスを可能としている．

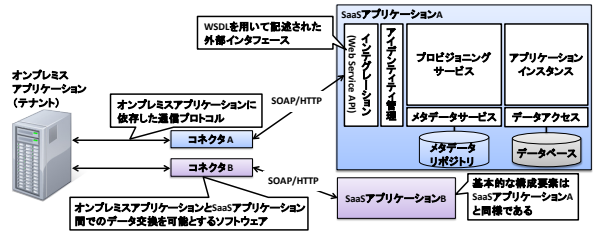

図 2　現状の SaaS インテグレーションモデル

2.3. SaaS アプリケーション特性の定義

SaaS インテグレーションにおける SaaS アプリケーションの特性を以下に定義する．

(1) インタフェースとメッセージング技術の定義：標準技術である WSDL, SOAP, JSON などを用いる．また，SaaS アプリケーション毎に異なるインタフェース(API)を提供する．

(2) 機能粒度の定義：SaaS アプリケーションの機能粒度は API 単位とする．API は基本的にカスタマイズできないため，オンプレミスアプリケーションは SaaS アプリケーションが提供する API に依存する．

(3) シングルインスタンス・マルチテナントの観点から定義：SaaS アプリケーションでは，テナント毎に異なるアプリケーションインスタンスを提供する．また，構成メタデータをカスタマイズすることでテナント毎に異なるデータモデルを提供する．API を利用してデータベースへアクセスする際，テナント毎の識別情報とデータモデルを把握している必要がある．

3. SaaS インテグレーションの問題

図 2 のモデルで考えられる問題を以下に示す．

(1) インタフェースの問題：SaaS アプリケーション毎に提供する API が異なる．このため，オンプレミスアプリケーション側はそれぞれの SaaS アプリケーションとのデータ連携で用いるコネクタに対応したインタフェースに変更する必要がある．

(2) 結合度の問題：データ連携する SaaS アプリケーションの API がオンプレミスアプリケーションの外部インタフェースの変更に影響を与える．このため，オンプレミスアプリケーションと SaaS アプリケーション間は密結合となり，拡張性が低いシステム構成となる．

(3) 機能粒度の問題：ハイブリッド型のシステム構成では，移行した SaaS アプリケーションの機能粒度と移行前に利用していたオンプレミスアプリケーションの機能粒度は異なる．このため，SaaS アプリケーションの機能粒度に依存した形で SaaS インテグレーションする必要がある．

(4) データモデルカスタマイズの問題：データモデルのカスタマイズが可能であるため，API を用いて間接的に SaaS アプリケーションのデータベースへアクセスする場合，利用者側が常に最新のデータモデルを把握している必要がある．

(5) シングルインスタンス・マルチテナントによる問題：API を用いてデータベースへアクセスする際，テナント毎にユーザを識別する必要がある．また，SaaS アプリケーション毎に識別方法が異なるため，SaaS アプリケーション毎に異なる識別情報が必要となる．

4. アプローチ

SaaS インテグレーションの問題はオンプレミスアプリケーションと SaaS アプリケーション間で，インタフェースの整合，機能の整合，テナントの整合を図ることが困難なことから起きる問題であると考える．本稿では，上記の整合を図るために SaaS インテグレーションブローカを提案する．さらに，SaaS インテグレーションブローカを用いた SaaS インテグレーションア

ーキテクチャを提案する.

3 章で指摘した インタフェースの問題, 結合度の問題は, インタフェースの整合に関連する問題である. 機能粒度の問題は, 機能の整合に関連する問題である. また, データモデルカスタマイズの問題, システムインスタンス・マルチテナントの問題は, テナントの整合に関する問題である. このため, SaaS インテグレーションブローカを用いて, それぞれの問題を解決することで, 関連する整合の実現を目指す. また, SOA を用いることでそれぞれの問題を解決する.

5. SaaS インテグレーションアーキテクチャ

5.1. SaaS インテグレーションブローカの定義

インタフェースの整合, 機能の整合, テナントの整合を実現するため, 以下に SaaS インテグレーションブローカのアーキテクチャを定義していく.

A) インタフェースの整合

(1) インタフェースの問題の解決方法: ブローカを拡張することで, 統一的なインタフェースから複数の SaaS アプリケーションとのデータ連携を実現する. また, ブローカは複数のコネクタの機能を提供する Web サービスとして定義する.

(2) 結合度の問題の解決方法: ブローカを用いることで, オンプレミスアプリケーションと SaaS アプリケーション間は疎結合となる. しかし, ブローカにインタフェースの複雑性を委譲した構成となり, SaaS アプリケーションとブローカ間が密結合となる. ブローカと SaaS アプリケーション間を疎結合にするため, コネクタサービスを拡張する. コネクタサービスは, 特定の SaaS アプリケーションとの間でデータ交換を実現する機能を提供する Web サービスとして定義する. コネクタサービスにブローカの機能の一部を移行することで, ブローカと SaaS アプリケーション間を疎結合にすることが可能となる. また, このような構成をとることで, システムの拡張性が高まる.

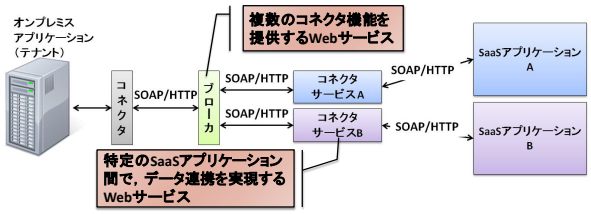

図 3 インタフェースの整合を実現するモデル

B) 機能の整合

(1) 機能粒度の問題の解決方法: この問題に対しては, ブローカにコネクタサービスの連携機能を持たせることで対応する. ブローカの実体はビジネスプロセスとして定義する. また, コネクタサービスの粒度は SaaS アプリケーションが提供する API 単位とする. ブローカを用いて, 複数のコネクタサービスを組み合わせることで, 機能粒度の違いを吸収する.

C) テナントの整合

(1) データモデルカスタマイズの問題の解決方法: データモデルのカスタマイズに対しては, メタデータモデルを定義することで対応する. 定義したメタデータモデルのインスタンスとして, それぞれの SaaS アプリケーションのデータモデルを作成する. オンプレミスアプリケーション側は, メタデータモデルを基にして SaaS インテグレーションを行う. また, メタデータモデルからそれぞれの SaaS アプリケーションのデータベースで保持しているデータモデルへのデータ変換を行う機能を提供するサービスを拡張する. 拡張したサービスは, データ変換サービスとし, 各 SaaS アプリケーションのテナント毎に提供する Web サービスとして定義する. データ変換サービスは, ブローカを用いて他のサービスと組み合わせて利用する.

(2) シングルインスタンス・マルチテナントによる問題の解決方法:テナント毎の識別情報を提供する識別サービスを拡張することで対応する. 識別サービスは, 入力値にユーザ名と SaaS アプリケーション名を持ち, 入力された値から適切なテナント情報を応答する機能を持つ Web サービスとして定義する.

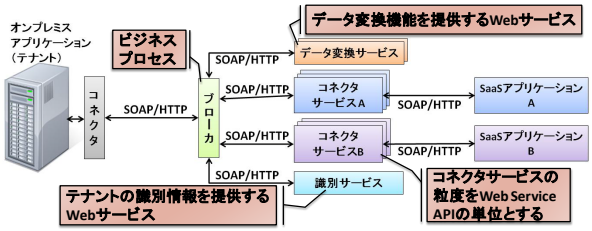

図 4 機能の整合, テナントの整合を実現するモデル

図 3 のモデルで示す要素を拡張することで, インタフェースの整合を実現することが可能となる. また, オンプレミスアプリケーションと SaaS アプリケーション間が疎結合となる. 疎結合になることで, 通信プロトコルの非依存性, 位置透過性, 実装非依存性, アプリケーションの独立性など SOA の持つべき性質を満たした拡張性が高いシステム構成となる[3].

図 4 のモデルで示す要素を拡張することで, 機能の整合, テナントの整合を実現することが可能となる. また, ビジネスプロセスで様々なサービスを組み合わせることで, SaaS アプリケーションの特性に対して柔軟に対応することが可能となる. このため, 環境が異なる様々な SaaS アプリケーションとのデータ連携を容易に実現するシステム構成となる.

5.2. SOA に基づく SaaS インテグレーションアーキテクチャ

SaaS インテグレーションブローカを用いた, SaaS インテグレーションアーキテクチャを提案する. 図 5 に SOA に基づく SaaS インテグレーションアーキテクチャを示す.

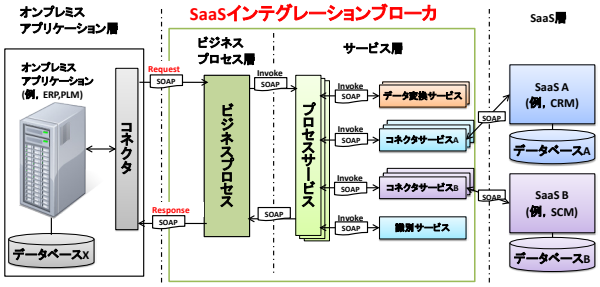

図 5 SOA に基づく SaaS インテグレーションアーキテクチャ

提案するアーキテクチャは, オンプレミスアプリケーション層, ビジネスプロセス層, サービス層, SaaS 層の 4 層で構成する. 以下, SaaS インテグレーションブローカを構成するビジネスプロセス層とサービス層の役割について定義する.

(1) ビジネスプロセス層: オンプレミスアプリケーションに対して統一的なインタフェースを提供するとともに, SaaS インテグレーションの詳細を隠蔽する. そうすることで, オンプレミスアプリケーションと SaaS アプリケーション間の疎結合を実現し, インタフェースの整合を図ることが可能となる.

(2) サービス層: プロセスサービスを用いて複数のコネクタ

サービスを組み合わせ，機能粒度の違いを吸収する．また，データ変換サービスと識別サービスを組み合わせることでテナントの違いを吸収する．そうすることで，機能粒度の整合，テナントの整合を図ることが可能となる．

また，サービス層で提供するコネクタサービスの粒度は，SaaS アプリケーションの API 単位とする．これにより，API の機能に適したインタラクション(同期型，非同期型)を選択することが可能となる．

5.3. SaaS インテグレーションメタモデルの定義

SOA に基づく SaaS インテグレーションアーキテクチャを構成する要素とその関係を SaaS インテグレーションメタモデルとして定義し，図 6 に示す．以下，SaaS インテグレーションメタモデルを構成する各要素について定義する．

(1) SOA に基づく SaaS インテグレーションアーキテクチャ：SaaS インテグレーションを実現するための要素．オンプレミスアプリケーション構成要素，コネクタ要素，拡張要素，SaaS アプリケーション構成要素からなるアーキテクチャとする．

(2) オンプレミスアプリケーション構成要素：オンプレミスアプリケーションを構成する要素群．

(3) コネクタ：オンプレミスアプリケーションとビジネスプロセス間のデータ交換機能を提供する要素．

(4) SaaS インテグレーションブローカ構成要素：(a)～(e)の要素から構成される．階層的な構造をとり，機能を段階的に詳細化することでシステムの拡張性を高めることが可能となる．

(a) ビジネスプロセス：受信した値から適切なプロセスサービスを呼び出す機能を提供する要素．統一的なインタフェースを実現する．複数のプロセスサービスから構成される．

(b) プロセスサービス：データ変換サービス，識別サービス，コネクタサービスを組み合わせることで，オンプレミスアプリケーションで提供していた粒度の機能を提供する要素．組み合わせる複数の下位サービスから構成される．

(c) データ変換サービス：メタデータモデルの値からテナント毎のデータモデルに適した値にデータを変換し，API コールに必要な情報を提供する要素．

(d) 識別サービス：SaaS アプリケーション毎にテナントの識別情報を提供する要素．複数の SaaS アプリケーション間でのデータ連携を実現する．

(e) コネクタサービス：SaaS アプリケーションの API 単位の粒度で機能を提供する要素．最小の機能粒度とする．

(5) SaaS アプリケーション構成要素：SaaS アプリケーションを構成する要素群．SaaS 特有の要素は，アイデンティティ管理，プロビジョニングサービス，メタデータサービス，メタデータリポジトリの要素から構成する．SaaS 特有の要素を用いることで，シングルインスタンス・マルチテナントを実現する．

5.4. SaaS インテグレーションアーキテクチャの振る舞い

SaaS インテグレーションメタモデルに基づく提案アーキテクチャの振る舞いを図 7 に示す．

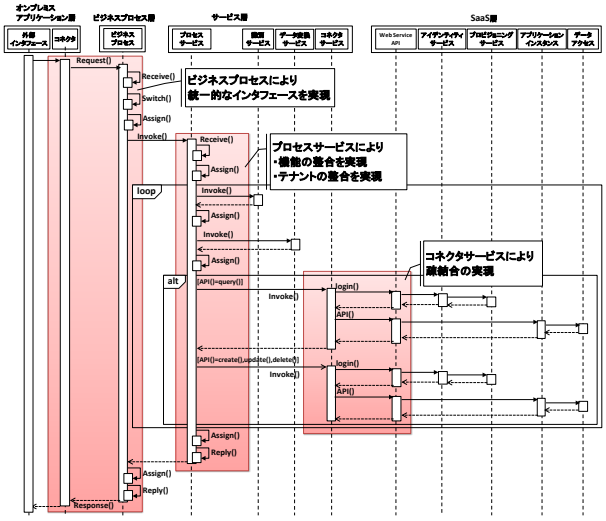

図 7 SaaS インテグレーションアーキテクチャの振る舞い

(1) ビジネスプロセスの振る舞い：受信した値を基に，適切なプロセスサービスを呼び出す．また，プロセスサービスで統合した一連の SaaS アプリケーションの処理結果を応答する．

(2) プロセスサービスの振る舞い：識別サービス，データ変換サービス，コネクタサービスを組み合わせ，オンプレミスアプリケーションの機能粒度に合致した値を応答する．

(3) 識別サービスの振る舞い：受信した値を基に，テナント毎の識別情報を応答する．同期呼出しとする．

(4) データ変換サービスの振る舞い：受信した値を基に，API コールに必要な情報を応答する．同期呼出しとする．

(5) コネクタサービスの振る舞い：識別サービスから取得した情報を基にテナントの認証を行い，SaaS アプリケーションとのセッション情報を得る．セッション情報とデータ変換サービ

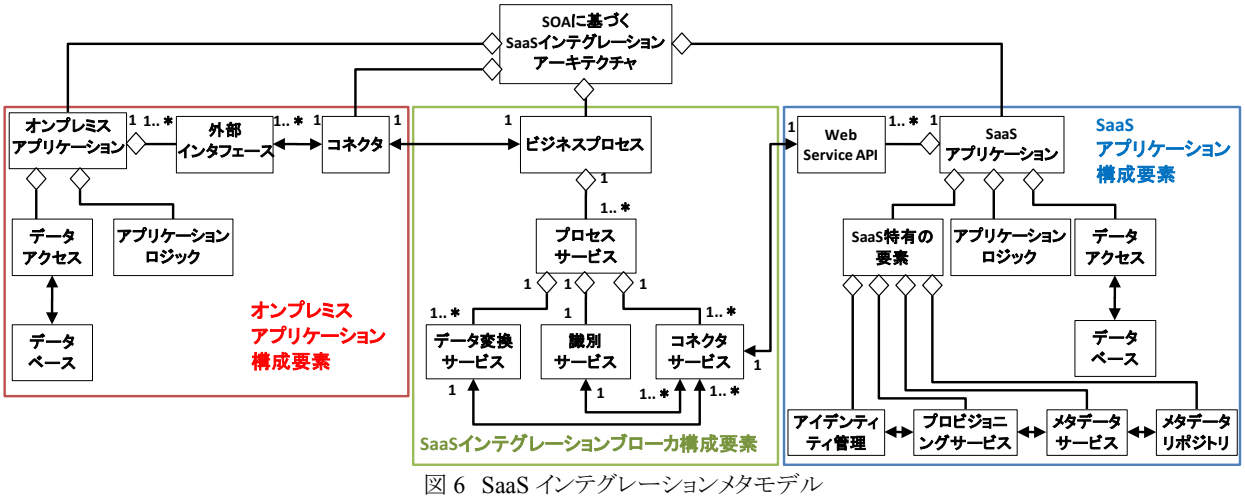

図 6 SaaS インテグレーションメタモデル

スから取得した情報を基に API コールを用いて SaaS アプリケーションのデータベースへアクセスする．SaaS アプリケーションの API コールの振る舞いに応じて，同期，または，非同期で呼び出す．

6. プロトタイプによる検証

6.1. プロトタイプの構成

本稿で開発したプロトタイプの構成を図 8 に示す．

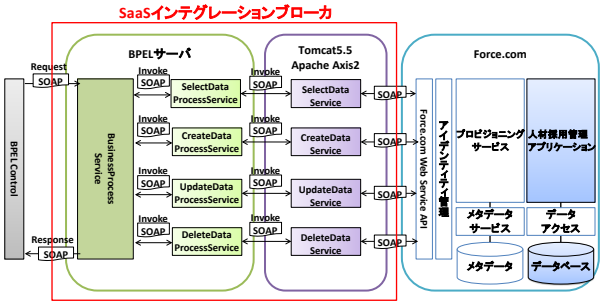

図 8　プロトタイプの構成

ビジネスプロセス，プロセスサービス，コネクタサービス，SaaS アプリケーションの開発をした．ビジネスプロセスとプロセスサービスは，BPEL サーバに配置する．コネクタサービスは，Tomcat5.5 上で動作する Apache Axis2 に配置する．また，SaaS アプリケーションは，Force.com を利用して開発した[2].

6.2. 開発実績

プロトタイプの開発実績を表 1　に示す．

表 1　プロトタイプの開発実績

プラットフォーム	プラットフォームの性能	プロトタイプの構成要素	要素数(個)	規模 (LOC)
Force.com	---	SaaS アプリケーション	1	構成メタデータの定義
Apache Axis2	CPU:Core2 Duo　2.6 GHz メモリ:2 GB	コネクタサービス	4	179
Oracle BPEL Process Manager	CPU:Pentium4　2.67GHz メモリ:1 GB	ビジネスプロセス	1	279
		プロセスサービス	4	437

6.3. 実行結果

図 9 に示すパターン A～C の応答時間，SaaS アプリケーションによる処理時間をそれぞれ 10 回計測した．また，それぞれのパターンの応答時間と SaaS アプリケーションの処理時間との差から SaaS インテグレーションブローカの構成要素によるオーバヘッドを導出した．

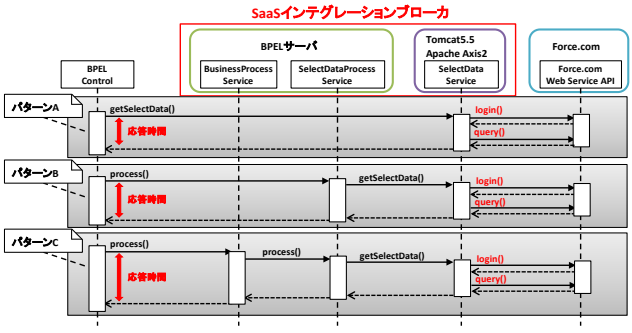

図 9　計測パターンの振る舞い

パターン A～C ともに，SaaS アプリケーションのデータベースへアクセスし，データを取得する処理を行う．

表 2　プロトタイプによる計測結果

パターン	メッセージ交換回数 (回)※	平均応答時間 (ms)	標準偏差 (ms)	SaaS の平均処理時間(ms)	ブローカによるオーバヘッド(ms)
パターン A	3	1425	121	login コール:706	55
パターン B	4	1979	237	query コール:664	609
パターン C	5	2578	433	合計:1370	1208

※要素間で行われるメッセージ交換回数．リクエスト/レスポンスで 1 回とする．

7. 関連研究

複数のコネクタ機能を提供する Integration Service を利用して SaaS インテグレーションを実現する研究がある[1]．Integration Service は SaaS アプリケーションとして提供され，ブラウザから操作やカスタマイズが可能となる．また，種類も複数ある．

8. 評価と考察

(1) プロトタイプによる評価と考察

表 2 で示した結果から提案アーキテクチャを考察する．

(a) ブローカによるオーバヘッドについて考察:SaaS インテグレーションブローカのビジネスプロセス要素とプロセスサービス要素の処理によるオーバヘッドが大きいと分析することができる．このため，これらの要素は性能が高いマシンに配置するなどの対策が必要となる．

(b) 応答時間のばらつきについての考察:ブローカを用いることで，標準偏差の値が増加する．この問題に対しては，非同期型のサービス呼出しで対応することが可能である．提案アーキテクチャでは，ビジネスプロセスを利用して複数のサービスを段階的に詳細化している．このため，適切な粒度のサービスに対して，メッセージ交換パターンを選択することが可能となる．

(2) 関連研究との比較による評価と考察

Integration Service は，1 つの SaaS アプリケーションとして開発されている．このため，本稿で提案するアーキテクチャに比べ，拡張性が低いシステム構成となる．

9. 今後の課題

(1) 複数の SaaS アプリケーションを用いたプロトタイプ開発:提案アーキテクチャでは，ハイブリッド型のシステム構成を想定する．このため，複数の SaaS アプリケーションを用いたプロトタイプを開発し，提案アーキテクチャの妥当性を評価する必要がある．

(2) プロセスパターンの定義:　オンプレミスアプリケーションの機能をサービス層で実現するためには，各サービスの開発やサービスの組み合わせ方を設計する必要がある．このため，再利用可能な粒度でサービスの組み合わせを，プロセスパターンとして定義する．定義したプロセスパターンを再利用することで，開発コストを抑えることが可能となる．

10. まとめ

本稿では，SaaS インテグレーションにおける SaaS アプリケーションの特性を定義し，SaaS インテグレーションの問題について指摘した．インタフェースの整合，機能の整合，テナントの整合の観点からアプローチを示し，SaaS インテグレーションアーキテクチャを提案した．また，SaaS インテグレーションメタモデルを用いて，提案アーキテクチャを構成する要素間の関係を明らかにした．さらに，プロトタイプと関連研究との比較から提案アーキテクチャの妥当性を評価した．

参考文献

[1] H. Hai, et al., SaaS and Integration Best Practices, FUJITSU Sci. Tech. J., Vol. 45, No. 3, Jul. 2009, pp. 257-264.
[2] Salesforce.com, Force.com Web Services API Developer's Guide, Salesforce.com, Sep, 2009, http://www.salesforce.com/us/developer/docs/api/apex_api.pdf.
[3] T. Erl, Service-Oriented Architecture, Prentice Hall, 2005.