SOA，OSS を利用した Web アプリケーション開発手法の紹介

稲沢事業所　情報システム技術部　情報システム技術課

・酒向　宏和
・三田　明宏
・須藤　篤志

要　旨

近年，社内業務における情報システムの役割はますます大きくなっている。その一方で，情報システムの開発・保守・運用に関するコスト削減や納期短縮が強く求められる状況にある。

このような限られた予算・期間の中で，高い要求を実現するため，サービス指向アーキテクチャ（SOA）とオープンソースソフトウェア（OSS）を利用して，システム開発を行った。

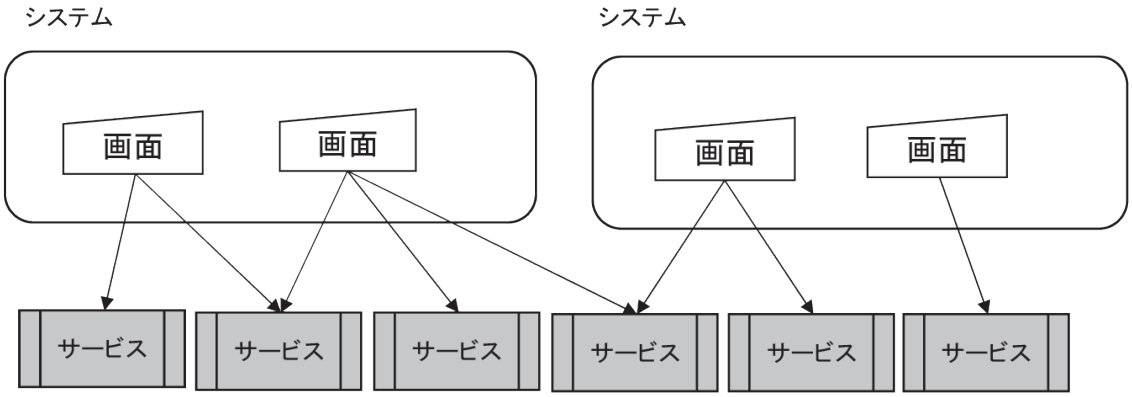
本稿では，SOA と OSS を利用した Web アプリケーション開発の事例と，その際に発生した課題と対策について紹介する。

SOA（Service Oriented Architecture）

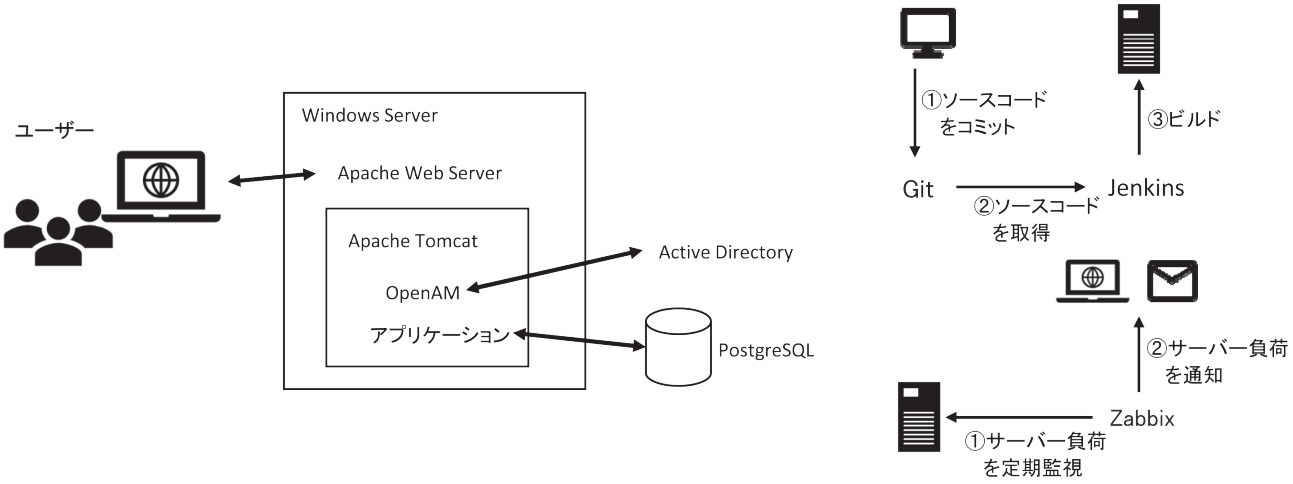
システムの開発手法の１つ。システムの各機能を１つの「サービス」としてとらえ，それを組み合わせることでシステム全体を構築する。

OSS（Open Source Software）

作成者がソースコードを無償で公開しており，利用や改変，再配布が自由に許可されているソフトウェア。



SOA を利用したシステムのイメージ



OSS の利用事例

1．まえがき

今後，サービス指向アーキテクチャ（SOA），オープンソースソフトウェア（OSS）を利用したシステム開発がさらに増えてゆくと予想される。同様のシステム開発の一助になればと考え，本テーマを選定した。

本稿では，SOA と OSS を利用した Web アプリケーション開発の事例と，その際に発生した課題と対策について紹介する。

2．サービス指向アーキテクチャ

2．1　サービス指向アーキテクチャとは

サービス指向アーキテクチャ（以降，SOA）とは，システム全体を，業務視点の機能単位（この１つ１つをサービスと称する）の組み合わせによって構築する開発（設計）手法のことである。

SOA は，業務プロセスをトップダウン型で分析し，サービスとしてシステム化する。各サービスは，互いに疎結合とし，サービス再活用の容易性及び保守性を確保する。図１に SOA を利用したシステム全体の構成を示す。

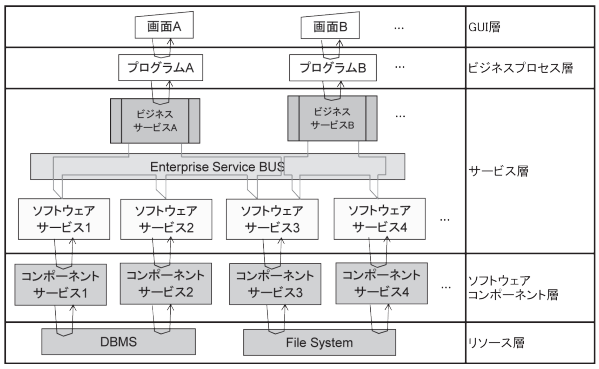


図１．SOA を利用したシステム全体の構成

- ビジネスサービス
  外部アプリケーションに公開するサービスインターフェース。ビジネスサービスが業務ロジックの単位。
- ソフトウェアサービス
  ビジネスサービスを構成するためのサービスインターフェース。
- コンポーネントサービス
  ソフトウェアサービスから利用され，繰り返し実行可能なロジックやデータを保持する。

サービス間の連携には，SOAP や XML などの標準化されたプロトコルやデータ形式を用いる。

2．2　ねらい

近年のアプリケーション開発では，コスト削減や納期短縮が求められている。これらの要求を１つのアプリケーションの開発だけで達成することは困難である。そこで，複数の Web アプリケーションの開発で総合的にコスト削減や納期短縮ができる SOA の技術を採用し，本課題に取り組んだ。

SOA を利用した Web アプリケーションを開発するにあたり，次の QCD 目標を定めた。

- 品質の向上（Quality）
  サービスを利用することでロジックやデータを疎結合型で共有する。デバッグ済みで品質確保された既存のサービスを再利用することで Web アプリケーションの信頼性を向上させる。
- コストの削減（Cost）
  個々の Web アプリケーションは，既存のサービスを再利用して実装工数を削減する。
- 納期の短縮（Delivery）
  Web アプリケーション間で共通する機能は SOA サービスを活用することで重複した設計・実装を避け，納期を短縮する。

2．3　システム開発

サービスの設計では，システムが取り扱う業務プロセス全体を分析し，サービス化する業務プロセスを抽出する。サービス化の設計は基本的にトップダウン型を推奨するが，既存アプリケーションで有益なプログラムがある場合は，ボトムアップ型で既存のプログラムコードを分析し，サービスへの切り出しを行う。

Web アプリケーションの設計では，業務ロジックをサービスの組み合わせで実装できるようにする。既存の Web アプリケーションは，改修が発生したタイミングで，段階的に SOA 方式に移行することで開発負荷を下げる。

2．4　開発時における課題と対策

SOA を活用したシステム開発時に発生した課題と実施した対策を紹介する。

課題は以下である。

- サービスの粒度によるオーバーヘッド／再利用性
- リソースへのアクセス順序によるデータ不整合
- 他社製品活用時のインターフェースの違い

２．４．１　サービスの粒度によるオーバーヘッド／再利用性

サービスの設計にあたり，サービスの粒度が課題となった。サービスの粒度を小さくすると，サービスの呼び出しにかかるオーバーヘッドが大きくなる。

逆にサービスの粒度を大きくすると，サービスの再利用性が低くなる。

そこで，下記の単位でサービスを設計することで解決した。

- 他のサービスとロジックが重複せず，１サービスで完結する最小の単位とする。
- 扱うデータの種類よって切り分ける。

２．４．２　リソースへのアクセス順序によるデータ不整合

リソースへのアクセスに関するエラーが発生すると，ファイルの更新やデータベースの更新をエラー前に戻す必要がある。データベースの更新はロールバックによって更新前の状態に戻せるが，ファイルは更新前の状態に戻せない。

また，各コンポーネントサービス間で互いに処理の終了待ちとなるデットロックによる処理不能状態とならないよう設計する必要がある。

上記を踏まえて，下記の方針をサービス設計に追加することで解決した。

- データの更新前チェックは，更新処理を行うサービス内で行う。
- データ操作は，データベース，ファイルの順とする。
- データベースの更新順序を統一する。

２．４．３　他社製品活用時のインターフェースの違い

開発コストや開発期間の関係で実装することが難しい機能は，他社の製品を活用した。

他社製品を活用する場合次の課題がある。

- インターフェースの違い

  製品毎にインターフェース仕様やデータ形式が異なる。この違いはサービスを利用するWebアプリケーションに意識させたくない。

上記課題の対策を，日・英翻訳サービスの実例とともに紹介する。

日・英翻訳サービスでは，Microsoft AzureのTranslator Text APIを活用して開発した。

他のサービスがXML形式でサービス連携することに対して，Translator Text APIはJSON形式で連携するインターフェース仕様となっている。この違いを次の仕組みで解決した。

- Webアプリケーションとビジネスサービスの連携は他のサービスと同じXML形式のインターフェースとする。
- コンポーネントサービスより，Microsoft Translator Text APIが定めるJSON形式のインターフェースで連携する。

上記の設計を踏まえた翻訳サービスのプログラム構成を図２に示す。

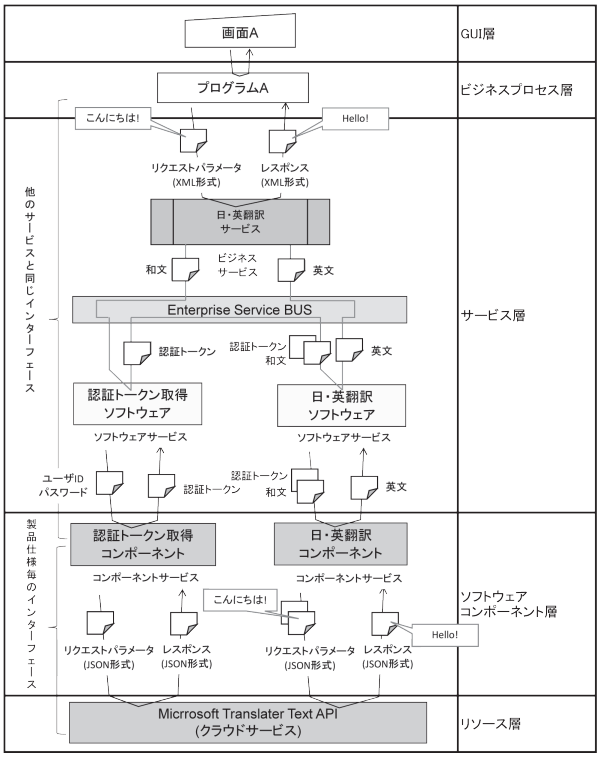


図２．翻訳サービスのプログラム構成

３．オープンソースソフトウェア

３．１　オープンソースソフトウェアとは

オープンソースソフトウェア（以下，OSSという）は，作成者がソースコードを無償で公開しており，利用や改変，再配布が自由に許可されているソフトウェアのことである。

OSSとして公開されたソフトウェアは，その全体を，インターネットで公開されており，誰でも無償で自由に使用，複製，改変，再配布及び自ら開発したプログラムへの組み込みを行なうことができる。

3.2　OSS利用におけるメリット・デメリット

OSSは商用，非商用の目的を問わず無料で利用し，自由にカスタマイズ可能なため，メリットのみに思える。しかし，実際にはデメリットも存在する。

3.2.1　メリット

●ライセンス費用が不要

オープンソースは，上述したとおり商用，非商用の目的を問わず無料で利用が可能である。

例えば，商用データベースで有名なOracleは，利用するには高額なライセンス費用や保守費用を支払う必要がある。

一方，オープンソースのPostgreSQLを利用するには，費用は一切不要である。

●開発の効率化による開発費の抑制と短納期

開発者が必要とする機能が既にOSSに実装されている場合，それを活用することで開発費の抑制や短納期が可能である。

●カスタマイズが自由

商用，非商用の目的を問わず修正が可能なため，カスタマイズも自由である。

そのため，オープンソースのソフトウェアにない機能があっても，自由に機能を修正したり，機能を作って追加したりすることが可能となる。

これに対し，有償で販売されるソフトウェアの場合には，ソフトウェアにない機能があっても，そもそもカスタマイズが許されておらず，開発元や提携ベンダーに依頼する必要がある。また，カスタマイズを受けてもらえない場合もある。

●不具合も自分で修正が可能

オープンソースのソフトウェアに不具合があった場合，ソースコードが公開されているため，自分で問題個所を調査して修正までできる。

コミュニティの活動が活発なオープンソースのソフトウェアの場合は，優秀な技術者が常に検証を行って報告や対応が行われており，常時アップデートされている。

これに対して，有償で販売されるソフトウェアの場合には，問題個所を調査すること自体も難しく，たとえ問題個所がわかっても，開発元の対応を待つ必要がある。

3.2.2　デメリット

●ドキュメント類が揃っていない場合がある

オープンソースのソフトウェアは，ライセンス費用が無料である代わりに，ドキュメント類が揃っていない場合がある。

有償で販売されるソフトウェアは，マニュアルが購入時に同梱されていたり，Webに掲載されていたりするが，オープンソースの場合，詳細なドキュメントが用意されていないことがある。

また，オープンソースはカスタマイズが自由にできるが，基本的には「自己責任」で調査・改造する必要がある。

●日本語の情報が少ない

オープンソースのソフトウェアは，海外発祥のものが多いため，日本語での情報が少ない場合もある。

有償で販売されるソフトウェアの場合，日本に入ってきたばかりの場合は，日本語の情報が少ない事も多いが，日本法人ができると，すぐに日本語対応が行われる。オープンソースのソフトウェアを利用する場合には，日本語だけでなく，英語も活用した方が入手できる情報が多くなる。

●開発元のサポートが用意されていない

オープンソースのソフトウェアでは，開発元のサポートは基本的には用意されていない。

有償のソフトウェアでは，開発元には契約によって定められたサポートの責任が発生するが，前述しようにオープンソースの開発元にはサポートを行う責任がない。そのため，何らかの不具合が発生したとしても，「自己責任」で対応をする必要がある。

これに対して，Microsoft OfficeのWordやExcelのような有償のソフトウェアでは，Microsoftが不具合の修正を行う。

3.3　活用事例

（1）Webシステム構築

OSSと有償ソフトウェアを組み合わせ，図3のシステム構成で動作するWebアプリケーションを構築した。全てをOSSとするのではなく，既存インフラの活用，保守のしやすさの観点から一部は有償ソフトウェアを利用する構成とした。

ユーザー認証はOpenAMを採用した。OpenAMはActive Directry，LDAPなどの複数のデータソースに対応しいる。導入先環境毎に異なるユーザー情報の管理方法が異なる場合でも，アプリケーション側の改修を行わず，OpenAMの設定変更を行うことで対応することが可能となる。また，RDBMSにはPostgreSQLを採用した。ライセンス費用が不要のため，導入コスト，ランニングコストを削減することができる。

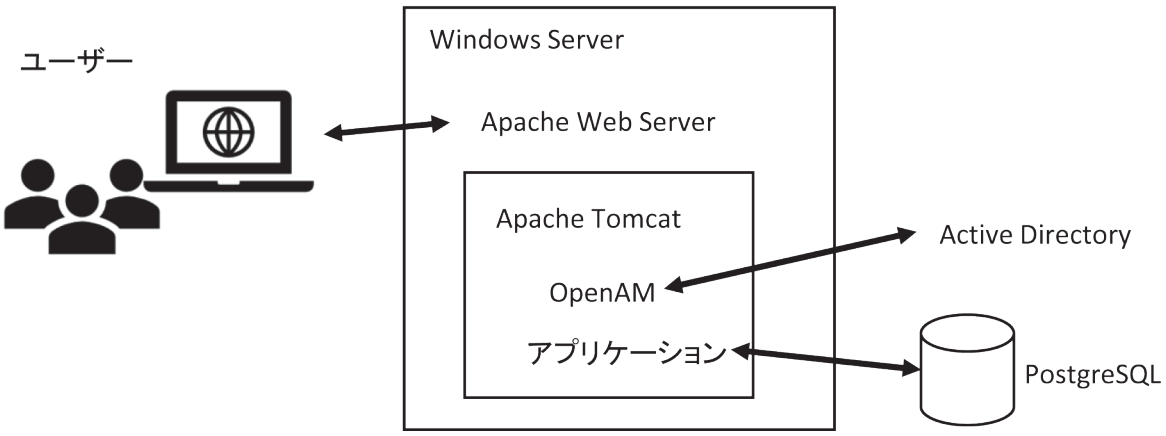


図３．OSS を活用したシステム構成

（2）ソースコード管理・ビルド

以前はバージョンごとにフォルダを作成してソースコードを管理していたが，変更点を追跡しにくい，バージョン管理が煩雑になるなど課題があった。そこで，ソースコードの変更履歴を記録・追跡するためにバージョン管理システムである Git を採用した。また，Jenkins と連携させることにより，Git へコミットを行うことでビルドまで連動して行うことが可能となった。

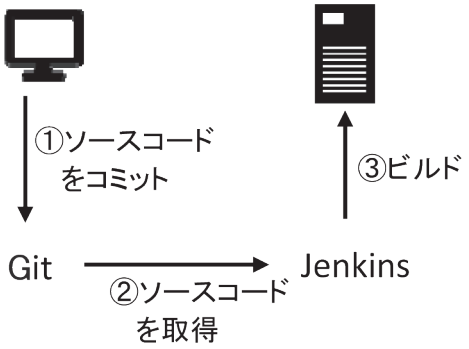


図４．Git を活用したソースコード管理・ビルド

（3）リリース後のサーバー監視

システム稼働後のサーバー監視については，Zabbix を利用する。予め閾値を設定しておくことで，閾値を超えたタイミングでアラートメールを送信することが可能となる。サーバーのロードアベレージの他，死活監視，ディスクの空き容量等の監視が可能なため，運営作業にかかる手間を削減できる。

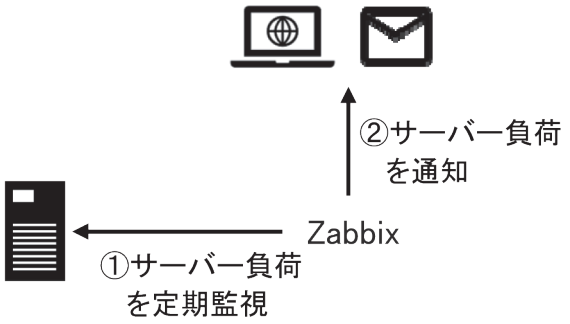


図５．Zabbix を活用したサーバー負荷監視

４．むすび

SOA は，概念としては有用だが，実際のシステム開発に適用するのは難しいという意見もある。

OSS は，無償のソフトウェアであるが，利用規約や条件が存在し，メリットだけでなく，デメリットもある。

本稿では，実際のシステム開発で，コスト削減，納期短縮に繋がった事例を紹介しているので，今後のシステム開発の参考になれば幸いである。

また，OSS を利用するかは，システムの開発時に都度検討しているため，その判断基準となるガイドラインの作成に取組んでいきたい。

執筆者紹介

酒向　宏和

2018年入社　情報システム開発に従事。
現在，情報システム技術部　情報システム技術課
グループリーダー

三田　明宏

2012年入社　情報システム開発に従事。
現在，情報システム技術部　情報システム技術課

須藤　篤志

2016年入社　情報システム開発に従事。
現在，情報システム技術部　情報システム技術課