アスペクト指向に基づくSOAの考察
– 非機能特性に着目したサービス抽出方法の提案 –

2008MI076 伊藤 広大　　2008MI086 亀山 怜希　　2008MI283 横山 脩二
指導教員　張 漢明

1　はじめに

ビジネス環境は急激に変化するようになり，それに応じてシステムには迅速かつ柔軟な対応が求められている．システムをサービスの集まりとして構築するサービス指向アーキテクチャ (以下，SOA) が，柔軟性や再利用性を向上する技術として注目されている．SOA に基づくシステムでは手続き指向やビジネスプロセスに基づき，機能的な単位であるモジュールをサービスと呼ぶ．Web サービスの普及にともない，SOA に関するアーキテクチャや設計プロセスなどの研究 [2][4] がおこなわれ確立されつつある．SOA に基づくシステム開発では，サービスを連携させる際の効率や信頼性などのシステムの品質が重視されている．

SOA に基づくシステム開発における問題として，非機能特性を満たすためのソースコードの散在による保守性の阻害があげられる．システムの品質は非機能特性によって大きく左右される．サービスは業務の単位やビジネスプロセスに着目し構築されている [1] ので，ひとつの機能を満たすまとまりになっている．これにより，非機能特性を実現するためのソースコードは各コンポーネントに散在している．散在した関心事により変更が容易ではなくなる．すなわち，SOA の利点である保守性の向上を妨げている．

本研究の目的は SOA に基づくシステムのサービス内やサービス間に散在する非機能特性を満たすためのソースコードを分離し，サービスとして部品化することである．これにより，SOA に基づくシステムのアプリケーション設計を支援することができる．本研究では，システムの機能に関してのアーキテクチャや設計プロセスは既に確立していることを前提とし，非機能特性に着目し考えていく．

本研究は非機能特性を満たすためのサービスを実現することで，横断的関心事を分離したシステムが構築できるのではないかという考えをもとに進める．ISO9126 を基に非機能特性について整理をおこなう．機能に関するサービスは適切に抽出されているものとする．既存研究で示された SOA の性質を考慮し，非機能特性に関するサービスについて考えていく．さらに，アスペクト指向技術を用いて，機能のサービスと非機能特性を満たすためのサービスの連携する仕組みを提案する．これにより，サービス内やサービス間の非機能特性に関するソースコードの冗長さを軽減でき，再利用性や保守性を高めることが可能である．実際に実現可能であるかを考察し確認した．

2　背景技術

2.1　SOA

SOA は，大規模なシステムをサービスの集まりとして構築する設計手法である．サービスとは，外部から標準化された手順によって呼び出すことができるひとまとまりのソフトウェアの集合である．ソフトウェアを部品化し，その組み合わせでシステムを構築していく手法は分散オブジェクト技術などがあり，従来から存在する．SOA での部品化の単位は従来の技術より粗い粒度のプログラム上の機能である．

2.2　アスペクト指向技術

アスペクト指向技術とは，ソフトウェアの横断的関心事をアスペクトととらえ，コンポーネント化可能にする技術である．アスペクトとは側面の意味であり，ソフトウェアにおけるコンポーネントやオブジェクトに横断して存在する関心事を横断的関心事という．アスペクト指向技術を用いることで，コンポーネントの把握・管理・変更が容易になり，再利用性や柔軟性の高いソフトウェアの実現を可能とする概念である．

2.3　AspectJ

AspectJ は代表的なアスペクト指向言語のひとつで，Java のアスペクト指向拡張言語である．AspectJ の重要な用語として，ジョインポイント (Join Point)，ポイントカット (Point Cut)，アドバイス (Advice) がある．ジョインポイントはアスペクトとアスペクトの接点であり，ポイントカットはジョインポイントの集合である．アドバイスはアスペクトからアスペクトへのメッセージ通信を記述する．

3　関連研究

木村隆洋らにより，レガシーシステムに SOA を適応させる際のサービス抽出方法に関する研究 [4] がされている．この研究では，レガシーソフトウェアに SOA を適用するための手段を述べている．サービスを決定するためにはサービスの定義が必要になってくるが，サービスの定義に関しては様々なものが存在する．この研究ではサービスを以下の 3 つの条件を満たすモジュールと定義している．条件は自己完結，オープンなインターフェース，粗粒度の 3 つである．

条件 1:自己完結

サービスは他のサービスに依存せずに実行可能である．すなわち，他の処理を実行した後でなければ実行できない処理や，入力として他の処理の出力が必須となるような処理はサービスとして認められない．

条件 2:オープンなインターフェース

サービスは外部から利用可能であるオープンなインターフェースを備える．プラットフォームに依存した呼び出し方法や，システム内部のみでしか利用されない一次データを入出力とするような処理は，サービスとして不適切である．

条件 3:粗粒度

サービスは単体でビジネスの構成単位として機能する処理である．複数のサービスを組み合わせて利用することで，より高機能・粗粒度なサービスを実現できる．

上記の条件は機能に着目した際のサービスを決定するための条件である．本研究では，非機能特性を満たすためのサービスと機能を満たすためのサービス (以下，機能のサービス) はどこが異なるのかを判断する基準としてこの 3 条件を用いる．

4 非機能特性を満たすためのサービス

4.1 非機能特性

非機能特性はシステムの機能が達成すべき性能や制限の要求を満たす特性であり，性能や信頼性，拡張性，セキュリティなど，機能特性以外のもの全般を指す．非機能特性を満たすことがシステムの品質の向上につながる．非機能特性はシステム全体に影響していることが多く，横断的関心事となりやすい．SOA に基づくシステムの場合，サービス間に横断して同一の非機能特性が求められることがある．

4.1.1 非機能特性の整理

ISO9126 を参考に非機能特性の整理をおこなった．ISO9126 はソフトウェアの品質を表す特性を定めた国際規格である．品質を表す特性を品質特性と呼び，6 つの品質特性に分類され，品質特性はさらに 27 の副特性に分類されている．サービスによって満たすことができる非機能特性として以下の 3 つがあると考えた．

- 機能性
  - セキュリティ
- 信頼性
  - 障害許容性
- 効率性
  - 時間効率性
  - 資源効率性

4.2 非機能特性を満たすためのサービスの実現方法

非機能特性を満たすためのサービスとは，単体でエンドユーザ向けのサービスになるのではなく，SOA に基づくシステムの開発を支援するためのサービスである．非機能特性を満たすためのサービスは機能のサービスと組み合わせてシステムとして構築する．非機能特性のサービスを利用することで開発者は品質を意識せずに開発をおこなうことができる．機能のサービスを実装した後に非機能特性を満たすためのサービスと連携させることで，品質を確保することが可能となる．

各々の品質副特性を満たすためのサービスの入出力とサービスの概要を図 1 に示す．

満たす非機能特性	サービスの概要	入力	出力
セキュリティ	暗号化	SOAPメッセージ	暗号化したSOAPメッセージ
	複合化	暗号化したSOAPメッセージ	SOAPメッセージ
障害許容性	再入力	入力データ	エラーメッセージ
時間効率性	応答時間，ターンアラウンドタイムを監視	指定した数値	数値超過報告
	スループットを監視	指定した数値 スループット	数値超過報告
資源効率性	CPU使用率を監視	指定した数値 CPU使用率	数値超過報告
	メモリ使用率を監視	指定した数値 メモリ使用率	数値超過報告

図 1　サービスの概要と入出力

4.2.1 セキュリティを満たすためのサービス

セキュリティとはソフトウェアへの不正なアクセスを排除する機能や，ミスによる資源破壊の防止の度合いである．SOA に基づくシステムでは，ネットワークを介したメッセージ通信を頻繁におこなう．したがって，ネットワークを通じた組織外部からの不正なアクセスによる情報の漏洩，破壊等がおこなわれる可能性がある．その防止策としてメッセージ通信をおこなう際にメッセージの暗号化，復号化をする必要がある．セキュリティを満たすために暗号化サービス，復号化サービスの 2 つをサービスとする．

4.2.2 時間効率性を満たすためのサービス

処理の速度が低下することは品質を損なうことになる．したがってシステム開発において考慮するべき点といえる．サービスは利用者の数によりマシンの負荷が変動するので，処理の速度も変動する．開発者は各々の処理についやす時間を把握することで，意図した時間をこえている部分の修正が可能となる．また，対策を講じることにより処理時間の向上をはかることができる．時間効率性の指標として応答時間，ターンアラウンドタイム，スループットがあげられる．時間効率性を満たすために処理時間を計測して報告する処理をサービスとする．

4.2.3 資源効率性を満たすためのサービス

システムを実行する際に，どの程度の資源を使用するかを示すのが資源効率性である．資源使用率の指標としてメモリ使用率やファイル使用率などがある．サービス利用者数の変化にともない，資源使用率も変動する．資源使用率が高い数値になるほど，実行効率は悪化する．したがって，開発者がサービスのメモリ使用率，CPU 使用率を知ることは重要である．これらの値が期待する値をこえている場合，修正や対策をすることは効率性の向上につながるといえる．資源効率性を満たすために，指定した値をこえた場合に開発者側に報告する処理をサービスとする．

4.2.4 障害許容性を満たすためのサービス

システムはデータの入力がおこなわれた際に，データが入力形式に背いていた場合は，利用者に再入力やアクセス拒否通知等をおこなう必要がある．再入力の場合に

は，入力されたデータを削除しインプットボックス等を初期状態に戻す必要がある．アクセス拒否等は利用者に拒否通知をメッセージとして表示する必要がある．再入力やアクセス拒否通知はサービス間やサービス内に散在していることが多い．障害許容性を満たすためのソースコードを横断的関心事ととらえサービスとする．

5　サービス連携

5.1　機能のサービスと非機能特性を満たすためのサービスの連携

1 つの機能のサービス，または複数のサービスに散在する非機能特性に関するソースコードを横断的関心事ととらえる．横断的関心事を AspectJ を用いてモジュール化し，サービスとする．すなわち非機能特性を満たすためのサービスは，アドバイスとしての非機能特性を満たすためのソースコードとポイントカットからなるアスペクトである．記述されたポイントカットにより非機能特性をに関するソースコードを織り込むことで連携する．連携のイメージを図 2 に示す．

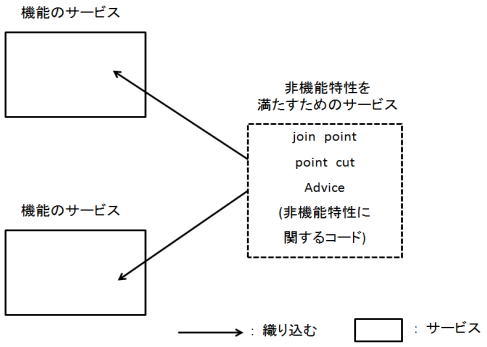

図 2　アスペクト J を用いたサービス連携のイメージ図

アスペクトをサービス化すると，連携する際にいくつかの問題点が生じる．

インターフェースにおける問題点

非機能特性を満たすためのサービスを実現しているのはアスペクトである．非機能特性に関するソースコードをアドバイスとして機能のサービスに織り込む．これにより非機能特性を満たすためのサービスは，インターフェースを備えていない．すなわち，非機能特性を満たすためのサービスを呼び出すことができない．

再利用における問題点

再利用する際は，機能のサービスに非機能特性を満たすためのソースコードを織り込まなければならない．すなわち，アスペクトである非機能特性を満たすためのサービスにジョインポイントを追加する必要がある．機能のサービスを開発する側は，非機能特性を満たすためのサービスを利用し，連携させることで新しいサービスを提供する．開発者側は非機能特性を満たすためのサービスの中身を変更することができないので，ジョインポイントを追加することができない．追加することが不可能であるので再利用できないといえる．

5.2　アスペクト間記述を用いた連携

5.1 節で述べた問題を解決するために，機能のサービスと非機能特性を満たすためのサービスの間にアスペクト間記述を用いる．アスペクト間記述に非機能特性を満たすためのサービスを呼び出すソースコードとそのソースコードを織り込む箇所を記述することで連携をはかる．これにより，機能のサービスと同様に非機能特性を満たすためのサービスにインターフェースを備えることができる．アスペクト間記述には AspectJ を用い，非機能特性を満たすためのサービスを呼び出すソースコードとしてポイントカット，ジョインポイント，アドバイスを記述する．これにより，機能のサービス内に非機能を満たすためのサービスを呼び出すソースコードを記述する必要がなくなる．アスペクト間記述を用いた連携のイメージを図 3 に示す．

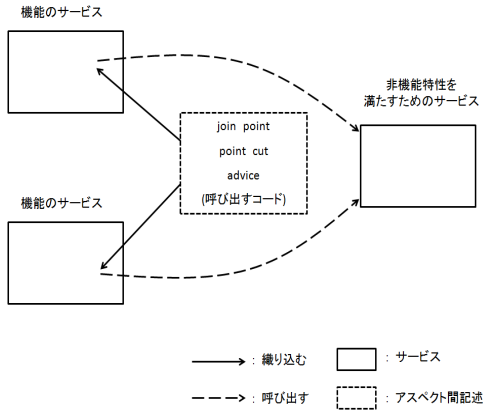

図 3　アスペクト間記述を用いたサービス連携のイメージ図

6　考察

6.1　非機能特性を満たすためのサービスについて

開発者は，非機能特性を満たすためのサービスを用いることで機能の開発に集中することができる．機能のサービスの開発を終えた後に，そのサービスに必要な非機能特性を考える．非機能特性を満たすためのサービスと連携させることで，非機能特性を満たしたシステムを構築できる．これにより再利用性が向上し，開発工数を大幅に減らすことができると考えている．非機能特性を満たすためのサービスを利用し，機能に関するソースコードだけのシンプルな構造を実現できる．システムを保守していく上で，非機能特性を満たすためのサービスに変更が生じる場合がある．サービス内に非機能特性に関するソースコードが散在している場合には多数の変更が必要となる．非機能特性に関するソースコードをサービスとしてひとまとまりにしておくことにより何度も変更をおこなう必要がなくなる．4.2.2 小節の時間効率性を満たすためのサービスのように，サービスにより異なる値が求められるものは呼び出し時の引数を利用する．

これにより，あらゆる時間を指定することができる．セキュリティに関するサービスについて，暗号化を行う前にネットワーク通信をしてしまうことがある．セキュリティを満たすためのサービスは機能のサービスと同一のマシン上に配置するという限定の下に利用する．資源効率性に関するサービスについて，CPU 使用率監視などは既に提供されているソフトウェアとしてを用いることで可能である．しかし，SOA はソフトウェア機能を独立したサービスという単位で実装し，それらをくみあわせてシステムを作り上げる考え方 [1] なのでサービスにする必要性があると考えた．さらに，非機能特性を満たすためのサービスの数や種類を増やし標準化をおこなうことによって，あらゆる SOA に基づくシステムに利用することができる．

6.2 サービス連携について

アスペクト間記述に非機能特性を満たすためのサービスを呼び出すソースコードとそのソースコードを織り込む箇所を記述することで連携をはかる．アスペクト間記述を用いることで，機能のサービスの中に非機能特性に関するソースコードを記述する必要がなくなる．ソースコードが整理され，システム全体の構造が把握しやすくなる．非機能特性を満たすためのサービスの必要性がなくなったときや呼び出す箇所を増やしたいときは，アスペクト間記述を更新するだけでよい．これにより保守性が向上するといえる．

6.3 機能サービスと非機能を満たすサービスの比較

3 章で述べた木村隆洋らが定めている 3 つの条件について，われわれが考えた非機能特性を満たすためのサービスと比較した．

条件 1 の自己完結について

時間効率性を満たすためのサービスと資源効率性を満たすためのサービスについては，条件を満たしている．セキュリティを満たすためのサービスと障害許容性を満たすためのサービスは条件を満たしていない．セキュリティを満たすためのサービスはメッセージ通信の前後に利用するサービスなのでメッセージ通信という処理に依存していると考える．障害許容性を満たすためのサービスについては範囲外入力があった際に再入力をさせるサービスであり，入力という処理に依存していると考える．すなわち非機能特性を満たすためのサービスは条件を完全に満たしているとはいえない．

条件 2 のオープンなインターフェースについて

5.2 節で述べた，アスペクト間記述にサービスを呼び出すソースコードを記述することで条件を満たすことができる．オープンなインターフェースを持つことでサービスの中身を知る必要がなくなり，機能のサービスと同じように利用できる．

条件 3 の粗粒度について

粗粒度の条件に関して満たしていない．非機能特性はビジネスの構成単位を満たすものではなく，システムの品質など機能を補佐するものが多いので粒度は小さい．

以上のことから，木村隆洋らが定めた機能についてのサービスの条件は，非機能特性を満たすためのサービスに用いることができない．機能のサービスと非機能特性を満たすためのサービスは本質的に違うものである．機能のサービスと連携することでエンドユーザが利用するサービスとして成り立つものであると考える．非機能特性を満たすためのサービスを定義するためには，新たなサービスの条件を定義する必要がある．

6.4 動的なサービス連携

本研究では，機能のサービスと非機能特性を満たすためのサービスの連携についてアスペクト指向を用いた連携方法を提案した．この連携方法はアスペクト指向プログラミングで実現している．アスペクトは織り込む場所が決まっているので，静的な連携方法といえる．サーバが壊れた場合やアクセス数が増えてきて負荷分散したい場合等は静的なアスペクトでは解決できない．そこで，動的なサービス連携についても考慮しなければならない．われわれは，このような動的な場合にもアスペクト指向技術を用いることで解決できると考えている．提案したサービス連携方法のアスペクト間記述にアスペクトを織り込む．使用しているサーバの負荷を分散したい時等に織り込まれるようにしておく．これにより，別のサーバに配置してある同じ内容のサービスに移行することができると考える．

7 おわりに

本研究では，サービス内，または複数のサービスに散在する非機能特性に関するソースコードについて，アスペクト指向を用いて分離した．非機能特性を満たすためのサービスについて提案，考察をおこなった．ISO9126 を参考に非機能特性について整理し，サービスによって実現できる非機能特性を特定した．アスペクト間記述を用いた連携の仕組みを考えた．非機能特性を満たすためのサービスを実現することによって再利用性，保守性が向上しシステムのアプリケーション設計支援することができることを確認した．

今後の課題として，意図しない状況においても正常に動作するために，6.4 章で述べたような動的なサービスの連携について考慮する必要がある．

参考文献

[1] IBM, “ *UdeveloperWorks*, ” http//www.ibm.com/developerworks/jp/websphere/library/soa/.

[2] M. P. Papazoglou, and W. Heuvel, “ Service-Oriented Design and Development Methodlogy, ” *IJWET*, no. 4, pp. 1-17, 2006.

[3] 木村隆洋，中村匡秀，井垣宏，松本健一, “ データ依存解析に基づくレガシーソフトウェアからのサービス抽出法, ” 電子情報通信学会，2005.

[4] 南波幸雄, “ 情報システムアーキテクチャとモデリングに基づく SOA の設計, ”産業技術大学院大学紀要, vol.58，no.2，pp.44-60，2008.