社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE

SOA システム構築のための既存システムの再利用性評価

田中 秀一郎† 西澤 茂隆† 田中 章弘† 中村 匡秀† 松本 健一†

†奈良先端科学技術大学院大学情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5

E-mail: †{shuichiro-t, shigetaka-n, akihir-t, masa-n, matumoto}@is.naist.jp

あらまし レガシーシステムをサービス指向アーキテクチャ（SOA）化する場合の再利用性を性質付ける新たなメトリクスとそのメトリクスの計測手法を提案する．まず，ビジネスプロセスをトップダウンに分析することによって抽出されたサービス（必要サービス）と，ソースコードを分析することによってボトムアップに切り出されたサービス（候補サービス）を機能面から付き合わせる．その後，必要サービスの実現に現存の候補サービスがどれだけ再利用可能かを分析する．そして，再利用できるレガシーシステムの割合をサイズメトリクスで計測する．提案手法を酒在庫管理システムの一実装に適用した結果，既存システムの潜在的なサービスの再利用率を定量的に計測できることが確認できた．

キーワード サービス指向アーキテクチャ，レガシーシステム，再利用性

Evaluating Reusability of Legacy System for Migration to SOA-based System

Shuichiro Tanaka† Shigetaka Nishizawa† Akihiro Tanaka†

Masahide Nakamura† Ken-ichi Matsumoto†

† Graduate School of Information Science, Nara Institute of Science and Technology

8916-5, Takayama, Ikoma, Nara 630-0192 Japan

E-mail: †{shuichiro-t, shigetaka-n, akihir-t, masa-n, matumoto}@is.naist.jp

Abstract We propose a new metric that measures the reusability in case of migrating from legacy system to service-oriented system (SOA), and a method to measure the metric. First of all, we compare two services in functional aspects; the first one, we call this necessary service, is a service which is obtained by analyzing business process using top-down approach. The second one, we call this candidate service, is a service which is obtained by analyzing source codes using bottom-up approach. Then, we analyze how exiting candidate services are reusable to realize necessary services. Finally, we measure the ratio of the reusable legacy system based on size metrics. As a result of applying proposal technique to existing liquor management system, we could confirm that we are able to measure potential reusability ratio of the service in the existing system quantitatively.

Keyword Service-Oriented architecture, Legacy System, reusability

1. はじめに

ビジネス環境の変化に合わせて企業の業務システムを迅速かつ柔軟に対応させることが求められている．しかし，長年運用・保守され続けている既存システム（レガシーシステム）は，他のシステムとの相互運用性やデータの共有などが考慮されていない．このようなシステムでは，業務手順（ビジネスプロセス）が変わる度に，システムを大幅に修正する必要があり，保守コストが膨大になってしまう．

この問題を解決する手段として，サービス指向アーキテクチャ（SOA）[5]が注目されている．SOA は，システムを「サービス」という機能単位の集合として設計する手法である．各「サービス」は，システムのプラットフォームや内部実装に非依存で，共通のインターフェースを通じて疎結合され，高度なサービスは既存サービスの組み合わせによって実現される．そのため，サービスの組み替えや追加によって，ビジネスプロセスの変化に対応して，業務システムを迅速かつ柔軟に変更させることができる．SOA のそうした利点から，レガシーシステムを SOA 化することに関心が集まっている．レガシーシステムを SOA 化する場合，既存システムを最大限再利用し，できるだけ開発コストを節約したいというニーズがある．しかし，レガシーシステムは長年の運用・保守の過程で変更が繰り返され，保守性が大きく低下していることが多い[7]．そのため既存システムから業務機能単位でサービスを切り出すことが難しい．そのような状況の下，経営者は既存システムを再利用すべきか，新しくシステムを構築すべきかを意思決定する必要がある．しかし，現状では，既存システムの SOA 化しやすさを評価する尺度が存在せず，その判断を行うことが難しい．

そこで，本稿では，この判断を支援するために，レガシーシステムを SOA 化する場合の再利用性を性質付ける新たなメトリクスとそのメトリクスの計測手法を提案する．

具体的には，まずビジネスプロセスを一般的な SOA 設計に基づいてトップダウンに分析し，現行で必要な業務機能を，サ

ービスという単位で抽出する（**必要サービス**と呼ぶ）．次に，我々の先行研究で提案する手法[4]を用いて，レガシーシステムのソースコードを分析し，サービスとして成立しうる機能群をボトムアップに抽出する（**候補サービス**と呼ぶ）．その後，必要サービスと候補サービスとを機能面からつき合わせ，必要サービスの実現に現存の候補サービスがどれだけ再利用可能かを分析する．最後に，再利用可能な候補サービスのシステム全体に占める割合を，サイズメトリクスを用いて計算する．

提案手法を酒在庫管理システム（**酒屋問題** [10]）の一実装に適用し，ケーススタディを行った．その結果，提案手法が既存システムにおける潜在的なサービスの再利用率を，定量的に計測できることがわかった．

2. サービス指向アーキテクチャ（SOA）

サービス指向アーキテクチャ（SOA）[5]とは，ソフトウェアを「サービス」という機能単位で連携・統合することで構築する設計手法である．サービスの定義に関しては様々なものが存在するが，本稿ではサービスを以下の 3 つの条件を満たすソフトウェア処理（タスク，プロセス）の集合と定義する．

条件１：自己完結

サービスは他のサービスに依存せずに，単体で実行可能である．つまり，他の処理を実行したあと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，外部から利用可能なオープンなインターフェースを備えている．プラットフォーム依存の呼び出し方法を要求するような処理や，システム内部でしか利用されない一時データを入出力とするような処理は, サービスとして認められない．ただし，サービスの開発言語や動作環境などはサービスごとに異なっていても問題とされない．

条件３：粗粒度

サービスは，単体で業務機能に対応する粗粒度の処理である．複数のサービスを組み合わせるで，より高機能・粗粒度なサービスを実現できる．

これらの３つの条件は, SOA におけるサービスの必要条件である[3][6]． Web サービス[9]等の標準的な SOA フレームワークでラップすることで，３つの条件を満たすソフトウェア処理をサービスとして外部公開することができる．これら３つの条件を満たすサービスは互い疎結合[11]であるという特徴がある．SOA にはこれら特徴があるため，ビジネスプロセスの変化に対応して，サービスを組み替えることで柔軟にシステムを変更できる．今後の議論のために、各サービス s が含むソフトウェア処理（機能）の集合を

$$feature(s) = \{ f_1, f_2, \cdots, f_n \}$$ で表す。

3. SOA 開発における既存のサービス抽出手法

3.1. トップダウンアプローチ

SOA に基づくシステム開発手法として，ビジネスプロセスを

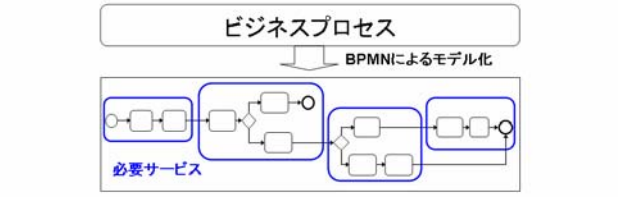

図 1 トップダウンアプローチ

トップダウンに分析し，システムを構築するアプローチが一般的である[6]．トップダウンアプローチでは，まず，事業レベルから担当者が行う作業レベルまで，階層的に業務を単純・抽象化する．そして，業務手順をフロー分析して，BPMN（Business Process Modeling Notation）[2]で表す．BPMN とは，業務手順を分かりやすく図示して可視化するための表記ルールを定めたものである．

次に，適切な粒度のビジネスプロセスをサービスとして切り出す．ビジネス環境の変化に強い柔軟なシステムを実現するためには, サービスの範囲と粒度が適切である必要がある. BPMN で可視化されたビジネスプロセスを，ビジネス環境の変化に応じて適切に組み換えや追加が可能な程度に小さくする. ただし，その粒度は業務視点でサービスが把握可能な程度に大きくなければならない．また，サービスは業務的に独立性が保たれてなくてはならない．

最後に，サービスにシステムの機能を対応付ける．サービスに対応する機能をもったプログラムやコンポーネントを適切な粒度のモジュールにまとめることでサービスを実装する．本稿では，与えられた業務プロセス p に対し，トップダウンアプローチで抽出されたサービス（**必要サービス**）s_n と呼び，m 個の必要サービス s_n の集合を

$$S_{necessay}(p) = \{ s_{n1}, s_{n2}, \cdots, s_{nm} \}$$ と表す.

3.2. ボトムアップアプローチ

もう 1 つの SOA に基づくシステム開発手法として，我々は先行研究[4]において，ソースコードからボトムアップにサービスを抽出する手法を提案している．このアプローチのキーアイデアは，ソースコードをデータフローダイアグラム（DFD）に変換し，処理間のデータ依存関係から条件 1,2,3（2 章参照）を満たす処理群をくくり出すことである．このサービス抽出法は以下の４つのステップで構成される．

Step 1.　ソースコードを DFD に変換する
Step 2.　DFD 上の処理間を流れるデータを分類する
Step 3.　DFD 上の処理間に依存関係を設定する
Step 4.　DFD に処理結合ルールを適応する

なお，ステップ 1 は，手続き型のソースコードをリバースエンジニアリングすることによって階層 DFD を導出する[1][8]. 導出された DFD の各レイヤに対して，ステップ 2 以降を適用する．

Step 2：データの分類

サービスのオープンなインターフェースを実現するために

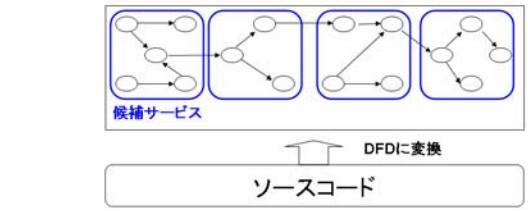

図 2 ボトムアップアプローチ

は，プロセス間を流れるデータがどのような性質のデータかを把握する必要がある．ここでは，DFD 上のデータを外部データ，システムデータ，モジュールデータの 3 種類に分類する．外部データとは，システムの外部要素とシステム内部のプロセスとの間でやり取りされるデータのことである．コード上では，ファイルや標準入出力に該当する．システムデータとは，システム内部の全てのプロセスが共通して利用するデータのことである．コード上では，データベースに入出力するデータやグローバル変数などが該当する．モジュールデータとは，システム内部の特定のプロセスが一時的に使用するデータのことである．コード上では，ローカル変数などに該当する．

Step 3：依存関係の設定

サービスの自己完結性を実現するために，DFD のプロセス間の依存関係を解析する．その結果，依存関係は強い順に，モジュールデータ依存＞処理依存＞システムデータ依存＞制御依存の 4 種類に分類できる．モジュールデータ依存とは，モジュールデータの入出力により発生する依存関係のことである．処理依存とは，プロセス間で同期をとって実行しなければならない処理内容により発生する依存関係のことである．システムデータ依存とは，システムデータの入出力により発生する依存関係のことである．制御依存とは，あるプロセス出力が他のプロセスの実行の有無を決定する依存関係のことである．

Step 4：サービス抽出

ステップ 3 で設定した依存関係を利用し，依存の強いプロセス群を結合し，自己完結したサービスとして抽出する．以下の 6 つのルールを用いてプロセスの結合要否を判断する．

（ルール 1）モジュールデータ依存プロセスの結合
（ルール 2）システムデータ依存プロセスの分割
（ルール 3）処理依存プロセスの結合
（ルール 4）制御依存プロセスの分割
（ルール 5）結合プロセスの合流
（ルール 6）結合プロセスの連鎖

本稿では，与えられたソースコード c に対し，ボトムアップアプローチで切り出されたサービスを**候補サービス** s_c と呼び，m 個の候補サービス s_c の集合を便宜上，

$$S_{candidate}(c) = \{s_{c1}, s_{c2}, \cdots, s_{cm}\}$$ と表す．

4. 提案手法

4.1. キーアイデア

SOA に基づくシステム開発手法として，トップダウンアプローチとボトムアップアプローチの 2 つの手法があることを示した．しかし，どちらのアプローチも既存システムを最大限再利用して，システムを SOA 化したいというニーズを満たすには不十分である．

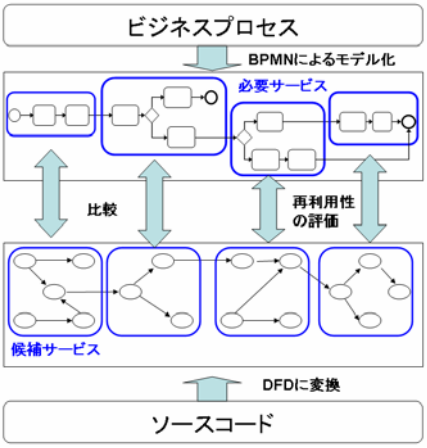

図 3 ハイブリッドアプローチ

トップダウンアプローチは新規に SOA に基づくシステムを構築するには適している．しかし，レガシーシステムを SOA 化する場合，この手法を直接適用することは難しい．なぜなら業務手順のプロセスモデル化はレガシーシステムの実装を考慮せずに行われるため，トップダウンに抽出したサービスがレガシーシステムからどれだけ切り出せるのかわからないからである．

ボトムアップアプローチはソースコードを分析してサービスを抽出する．そのため，抽出されたサービスは，レガシーシステムのプログラムから実際にサービス単位で切り出すことが可能である．しかし，ソースコードのみを情報源するため，ビジネスプロセスの意味解析は不可能である．つまり，抽出されたサービスはあくまでサービスの必要条件を満たすサービス候補でしかない．

以上の理由から，レガシーシステムを SOA 化する場合の既存システムの再利用性を計測するためには，ボトムアップアプローチとトップダウンアプローチを組み合わせた**ハイブリッドアプローチ**が必要だと考える．具体的には，それぞれのアプローチから抽出される必要サービスと候補サービスとをつき合わせることによってレガシーシステムの再利用性を分析する．

提案手法は以下の通り定式化される．

入力：ビジネスプロセス p，レガシーシステム LS のソースコード c

出力：LS のサービス再利用率

手続き：

Step 1（ハイブリッドアプローチによるサービス抽出）：p に対してトップダウンアプローチ，c に対してボトムアップアプローチを適用し，それぞれ $S_{necessary}(p)$, $S_{candiadate}(c)$を得る．

Step 2（サービス再利用性判定）：各必要サービス $s_n \in S_{necessary}(p)$ と各候補サービス $s_c \in S_{candiadate}(c)$を機能面から比較し，$s_n$ の実現に s_c が再利用可能かを判定する．

Step 3（サービス再利用率定量化）：再利用可能なサービス候補(群)の LS に占める割合をサービス再利用率として計算する．

以降の章で，各ステップの詳細を述べる．

4.2. ハイブリッドアプローチによるサービス抽出

本ステップでは，まず与えられたビジネスプロセス p をトップダウンに分析することで，業務の視点から把握可能かつビジネス環境の変化に柔軟に対応可能である適切な粒度の必要サービスの集合 $S_{necessary}(p)$を抽出する．

次に，与えられたソースコード c をボトムアップに分析することによって，レガシーシステムから実際に切り出すことができる候補サービスの集合 $S_{candiadate}(c)$を抽出する．このとき，ボトムアップアプローチはボトムアップアプローチで注目するDFD の階層によって切り出せるサービスの粒度が異なる．一般的に，高機能・粗粒度なサービスは，細粒度なサービスを複数組み合わせることで実現できる．したがって，ボトムアップアプローチの適用においては，できるだけ細粒度のサービス抽出を行い，より再利用性が高い候補サービスを切り出す．

4.3. サービス再利用性判定

本稿で着目するサービスの再利用とは，$S_{necessary}(p)$各必要サービスの実現において，$S_{candiadate}(c)$の候補サービスをどの程度そのまま利用できるか，ということである．再利用においては，必要サービスが要求する機能を候補サービスが提供する機能を用いて実現する．この目的のため，本稿では，サービスの再利用性を以下のように定義する．

定義（サービス再利用性）：必要サービス $s_n \in S_{necessary}(p)$と候補サービス $s_c \in S_{candiadate}(c)$が与えられたとする．**$s_n$ に対して s_c が再利用可能である**とは，$feature(s_n) \supseteq feature(s_c)$が成り立つ，かつ，そのときのみであると定義する．このとき，s_c を s_n の**再利用サービス**と呼ぶ．s_n のすべての再利用サービスの集合を $S_{reuse}(s_n)$と書く．ここで，$S_{reuse}(s_n) \subseteq S_{candiadate}(c)$である．

上記再利用性の定義において，候補サービス s_c は必要サービス s_n と比べて，機能的に小さくなくてはならないことに注意されたい．この場合，s_n の実現において，s_c をそのまま(手を入れることなく)再利用でき，不足する機能については他の候補サービスや新規サービスで補足すればよい．

一方，s_c が s_n で必要とする以上の機能を含んでいる場合は，再利用の定義から外れる．一般に肥大化したレガシーシステムをコンパクトかつ疎結合なサービス群に分離･分割することは，多大なコストがかかる問題だと知られている．したがって，s_c から s_n で必要とされる機能のみを切り出してサービスとして利用することは，少なからずコードの改修が必要である．この場合，本稿ではサービスの再利用と呼ばないこととする．

4.4. 再利用性の定量化

再利用性を定量化するため，必要サービス s_n が必要とする機能のうち，再利用サービスが提供可能な機能の割合を，**サービス再利用率**として定量化する．割合の算出においては，機能の大きさを性質付ける何らかの**サイズメトリクス**を用いる．ソフトウェア工学の分野では，代表的なサイズメトリクスとして，LOC（Lines of Code）や FP（Function Point）等，様々なものが知られている．しかし，提案手法においては，あえて特定のものに固定せず，再利用性評価者が何らかのサイズメトリクスを自ら選択して適用するものと仮定する．いま，機能 f に対し，何らかのサイズメトリクスで定量化した f のサイズを $size(f)$と書くことにする．そして，サービス s のサイズをそのサービスが持つ機能を元に以下のように表す．

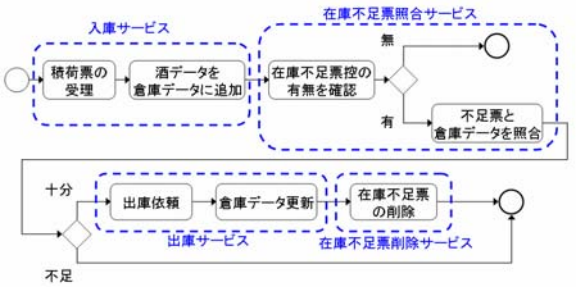

図 4 積荷票処理の BPMN

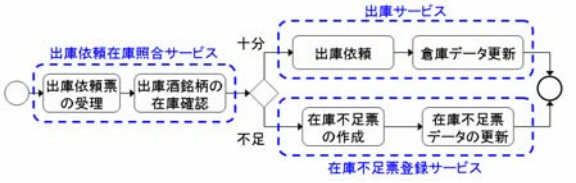

図 5 出庫依頼処理の BPMN

$$size(s) = \sum_{f \in feature(s)} size(f)$$

定義（サービス再利用率）：必要サービス $s_n \in S_{necessary}(p)$が与えられたとする．このとき，s_n に対する候補サービスの再利用率 $Reuse(s_n)$を以下のように定義する．

$$\mathrm{Re}use(s_n) = \frac{size(\bigcup_{s_{ci} \in S_{reuse}(s_n)} s_{ci})}{size(s_n)}$$

とする．また，レガシーシステム全体の再利用性は，必要サービスの再利用率をそのサイズで重み付けして足し合わせたものとする．

$$\mathrm{Re}use(LS) = \sum_{s_n \in S_{necessary}} \frac{size(s_n)}{\sum_{s_n \in S_{necessary}} size(s_n)} \mathrm{Re}use(s_n)$$

$$= \frac{\sum_{s_n \in S_{necessary}} size(S_{reuse}(s_n))}{\sum_{s_n \in S_{necessary}} size(s_n)}$$

5. ケーススタディ

5.1. 酒在庫管理システム

ケーススタディとして，酒屋問題[10]におけるビジネスプロセスと，その一実装である酒在庫管理システム（C 言語，797 行）に対して提案手法を適用し，サービス再利用性の評価を行った．

酒在庫管理システムは，大きく分けて積荷票処理と出庫依頼

処理の２つの機能から構成されている．積荷票処理は，酒を倉庫に入庫した後，在庫待ち状態になっている酒を出庫する処理である．出庫依頼処理は，依頼された酒を倉庫から出庫する処理と在庫不足登録処理の２つからなる．

5.2. 必要サービスの抽出

まず，トップダウンアプローチによって，酒屋問題における積荷票処理と出庫依頼処理の業務手順を BPMN で表した（図4,5）．積荷票処理では，ユーザが積荷票を入力すると，記載されている酒銘柄のデータを倉庫データに追加する．その後，在庫不足票の有無を確認し，入庫した酒銘柄が在庫不足票にあれば，出庫処理を行う．出庫した銘柄については，在庫不足票から削除する．出庫依頼処理では，ユーザが出庫依頼票を入力すると，出庫依頼酒銘柄が倉庫にあるか検索する．倉庫に十分あれば出庫処理を行い，出庫した分だけ倉庫データから該当酒銘柄の在庫を減らす．倉庫に十分なければ，在庫不足票に出庫できない銘柄のデータを追加する．

次に，業務手順を表した BPMN に基づき，業務の視点から把握可能かつ環境変化にも対応可能な適切な粒度でサービスを抽出した．積荷処理から抽出された必要サービスは，「s_{n1}:入庫」，「s_{n2}:在庫不足票照合」，「s_{n3}:出庫」，「s_{n4}:在庫不足票削除」の 4 サービスであった．出庫依頼処理から抽出された必要サービスは，「s_{n5}:出庫依頼在庫照合」，「s_{n6}:在庫不足票登録」，「s_{n7}:出庫」の 3 サービスであった．なお，s_{n3} と s_{n7} は同じ機能を持つ同一のサービスである．以上をまとめると，必要サービスの集合は，

$S_{necessary}$(酒屋問題) = {s_{n1}, s_{n2}, s_{n3}, s_{n4}, s_{n5}, s_{n6}}

となった．

5.3. 候補サービスの抽出

本実装から得られた DFD の上位レイヤに対しサービス抽出を行うと，「s_{c1}:在庫不足票作成」，「s_{c2}:入庫」，「s_{c3}:在庫待ち解消」，「s_{c4}:出庫依頼」，「s_{c5}:空き倉庫削除」の基本業務として成立する 5 つの候補サービスが抽出された（図 6）．

さらに，「s_{c2}:入庫」，「s_{c3}:在庫待ち解消」，「s_{c4}:出庫依頼」の 3 サービスに対しては下位レイヤより細粒度のサービス抽出を行った（図 7,8,9）．「s_{c2}:入庫」からは「s_{c21}:パラメータチェック」と「s_{c22}:倉庫更新」の 2 つの候補サービスが抽出できた．「s_{c3}:在庫待ち解消」からは「s_{c31}:不足票解消」と「s_{c32}:処理済控消去」の 2 つの候補サービスが抽出できた．「s_{c4}:出庫依頼」からは「s_{c41}:パラメータチェック」，「s_{c42}:酒在庫チェック」，「s_{c43}:在庫不足登録」，「s_{c44}:出庫」，「s_{c45}:不足票照合」の 5 つの候補サービスを抽出できた．入庫プロセス，在庫待ち解消プロセスに関してはお互いのプロセス依存が強く，これ以上細粒度のサービスは切り出せなかった．以上，最も小さい粒度のサービスを合わせて，

$S_{candidate}$(酒在庫管理システム) = {s_{c1}, s_{c21}, s_{c22}, s_{c31}, s_{c32}, s_{c41}, s_{c42}, s_{c43}, s_{c44}, s_{c45}, s_{c5}}

が得られた．

5.4. 再利用性の計測

2 つのアプローチで得られた必要サービスと候補サービスを

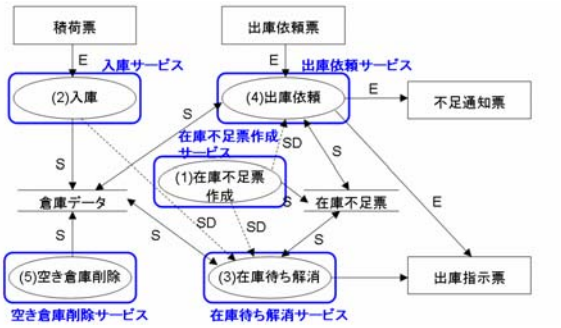

図 6 酒屋システム全体の DFD

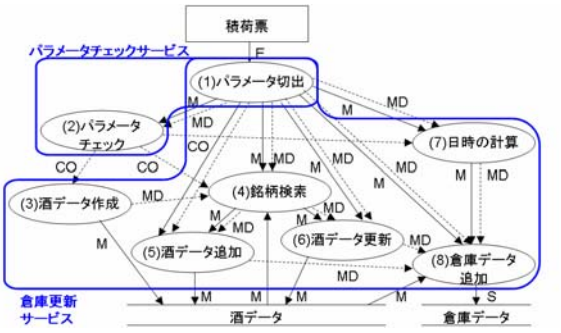

図 7 入庫プロセスの詳細 DFD

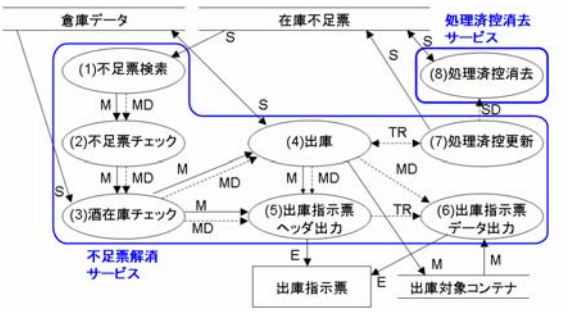

図 8 在庫待ち解消プロセスの詳細 DFD

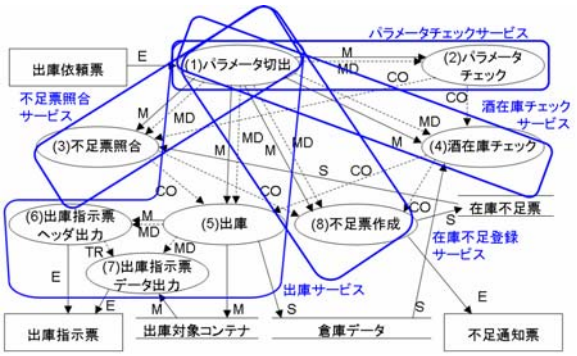

図 9 出庫依頼プロセスの詳細 DFD

機能面からマッチングした．すると，「s_{n2}:在庫不足票照合」必要サービス以外の必要サービスには，それぞれ過不足なく同等の機能を持つ候補サービスが得られた（表 1）．「s_{n2}:在庫不足票照合」が要求する機能は，「s_{c31}:不足票解消」候補サービスに含まれている．しかし，図 8 からわかるように，「s_{c31}:不足票解消」

表 1 必要サービスの再利用性

必要サービス	候補サービス	再利用率
s_{n1}:入庫	s_{c22}:倉庫更新	100%
s_{n2}:在庫不足票照合	なし	0%
s_{n3}:出庫	s_{c4}:出庫	100%
s_{n4}:在庫不足票削除	s_{c32}:処理済控消去	100%
s_{n5}:出庫依頼在庫照合	s_{c42}:酒在庫チェック	100%
s_{n6}:在庫不足票登録	s_{c43}:在庫不足票登録 s_{c1}:在庫不足票作成	100%

には，別のサービスとして切り出すべき出庫機能まで含まれているため，そのまま再利用できない．また，「s_{c31}:不足票解消」以外に「s_{n2}:在庫不足票照合」サービスの機能を充足する候補サービスは存在しなかった．

次に，サイズメトリクスによって再利用率を定量化する．本ケーススタディでは，サイズメトリクスとしてコード行数を用いた．理由は，単純で計測が容易なことと，自動生成コードなどの影響がないレガシーシステムでは，コード行数と開発コストに相関が高いと考えたためである．ただし，必要サービスの機能は実装されていないため，同等の機能に対応する既存システムのコード行数とする．以下にソースコードを 3 種類に分類した．

1．再利用できる機能に関するコード（α）
2．再利用できない機能に関するコード（β）
3．機能と直接関係のない実装依存のコード（γ）

そして，再利用率を以下のように計算した．

$$\mathrm{Reuse}(s_n) = \frac{size(\bigcup_{s_{ci} \in s_{reuse}(s_n)} s_{ci})}{size(s_n)} = \frac{\alpha}{\alpha + \beta}$$

各必要サービスの再利用率は（表 1）に示す．次に，レガシーシステム全体の再利用率を計測した．797 行のソースコードのコメントと空行を削除し，1 行 1 コマンドとすると 400 行となった．そのうち，再利用できる機能に関するコードは 270 行，再利用できない機能に関するコードは 68 行, 機能と直接関係のない実装依存のコードは 62 行だった. システム全体の再利用率は 79.9%だった.

5.5. 考察

酒在庫管理システムの一実装に提案手法を適用した結果，SOA 化するときの再利用率が高いことがわかった. その理由として，まず酒在庫管理システムの実装が 1 機能ごとにメソッドとして細かく分けられており，細粒度の候補サービスを容易に切り出せたからだと考えられる．もう一つの理由として，酒在庫管理システムは，手続き型言語で実装されていたが，長年の保守を経ていたわけではなかったため，業務機能単位に対応した候補サービスを切り出しやすかったことがあげられる．実際のレガシーシステムでは，長年の運用・保守の過程で，小さな変更が繰り返されシステム内部のプログラムやコンポーネントの結合が複雑になってしまう[7]．そのため，ボトムアップアプローチによる細粒度のサービス抽出が困難になり再利用性は低下すると考えられる．

また，本稿では提案した再利用性は，余分な機能の分離を必要とするサービスの再利用性を認めない．その理由は，ボトムアップアプローチでサービスとして分割できなかった機能を切り離すのには，膨大なコストがかかると考えられるため現実的に再利用が難しいと考えたためである．このことから，提案した再利用性メトリクスは確実に再利用できる割合を表していると言える．しかし，実際にレガシーシステムを SOA 化する場合には，不要な機能の切り離し再利用するケースもある．今後の研究として，モジュールの凝集度と結合度から，不要な機能の切り離しコストを見積もり，切り離しコストを考慮に入れた機能分離型再利用性を検討したい.

6. おわりに

本稿では，レガシーシステムを SOA 化する場合の再利用性を性質付ける新たなメトリクスとそのメトリクスの計測手法を提案した. また, 提案手法を酒在庫管理システムに適用し, SOA 化するときの潜在的な再利用性を定量的に計測した.

今後の研究として，提案した再利用性メトリクスの評価実験を計画している．再利用性が異なる複数のソフトウェアを，実際に SOA 化し，そのときにかかった開発コストを計測することで提案メトリクスの有用性を検証したい.

7. 謝辞

この研究は，科学技術研究費(若手研究 B 18700062)，21 世紀 COE プログラム「NAIST-IS:ユビキタス統合メディアコンピューティング」の助成を受けて行われている.

文　献

[1] A.B.O'Hare, E.W.Troan, "RE-Analyzer: From source code to structured analysis", IBM Systems Journals, Vol.33, No.1, 1994.

[2] Business Process Management Initiative -http://www.bpmn.org/

[3] Hao He, What is Service-Oriented Architecture? – http://webservices.xml.com/pub/a/ws/2003/09/30 /soa.html

[4] 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,” 信学技報 ソフトウェアサイエンス研究会，Vol.SS2005-42, pp.013-018, October 2005.

[5] M.P.Papazoglow, D.Georgakopoulos, "Service Oriented Computing", In Communications of the ACM, Vol.46, No.10, pp.25-28, October 2003.

[6] 牧野友紀，”ビジネス環境と実装システムを繋ぐ BPM と SOA”，情報処理, Vol.46, No.1, pp60-63, January 2005.

[7] 門田暁人，佐藤慎一，神谷年洋，松本健一,”コードクローンに基づくレガシーソフトウェアの品質の分析,” 情報処理学会論文誌, Vol.44, No.8, pp.2178--2188, August 2003.

[8] P. Benedusi, A. Cimitile, and U. De Carlini, "A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance”, Proc. of Int'l Conf, on Software Maintenance (ICSM’89), pp.180-189, Oct. 1989.

[9] W3C Web Service Activity - http://www.w3.org/2002 /ws/

[10] 山崎利治，“共通問題によるプログラム設計技法解説”, 情報処理，Vol.25, No.9, pp.934-935, 1984.

[11] 吉松史彰，“XML Web サービスにおける疎結合とは何か”, http://www.atmarkit.co.jp/fdotnet/opinion/yoshimatsu/onepoint03.html, July 2002.