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

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

木村 隆洋† 中村 匡秀† 井垣 宏† 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: †{taka-kim, masa-n, hiro-iga, matumoto}@is.naist.jp

あらまし レガシーソフトウェアにサービス指向アーキテクチャ（SOA）を適用するための一手段として，本論文では，手続き型言語で書かれたソースコードからサービスを抽出する方法を提案する．具体的には，まずソースコードから，データフローダイアグラム（DFD）を取得する．次に，DFD 上のデータを 3 種類に分類し，プロセス間の依存関係を性質付ける．この依存関係に基づき，DFD 上の複数のプロセスを，自己完結したサービスとしてくくりだす 6 つのルールを提案する．既存のアプリケーション（酒在庫管理システム）に対してサービスを抽出する実験を行い，ソースコードから様々な粒度のサービスを抽出することができた．

キーワード サービス指向アーキテクチャ，レガシーソフトウェア，DFD，リバースエンジニアリング，サービス抽出

Extracting Services from Legacy Software Based on Dependency Analysis

Takahiro KIMURA† Masahide NAKAMURA† Hiroshi IGAKI† and Ken-ichi MATSUMOTO†

†Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192 Japan

E-mail: †{ taka-kim, masa-n, hiro-iga, matumoto}@is.naist.jp

Abstract To facilitate adaptation of legacy software for the service-oriented architecture (SOA), this paper presents a method that systematically extracts services from source codes of a procedural system. We first obtain data flow diagrams (DFDs) from the given source codes with the existing reverse-engineering technique. Then, we define dependencies among processes on the DFDs by classifying the data into three categories. Based on the dependencies, we propose six rules that aggregate several processes on the DFDs as a self-contained service. We have conducted an experiment for an existing legacy application (liquor management system). As a result, it was shown that services with various granularities were identified reasonably from the source codes.

Keywords Service-Oriented Architecture, Legacy Software, DFD, Reverse Engineering, Service Extraction

1. はじめに

今日，ビジネス環境はますます急激に変化するようになり，企業の業務システムもその変化に対して，迅速かつ柔軟に対応することが求められている．しかし，既存のシステムの多くは，業務毎に高度に専門化されたモノリシック（一枚岩）なシステムが多く，任意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の大幅な修正が必要となり，保守コストが膨大になることが大きな問題とされている．

このような背景の下で，サービス指向アーキテクチャ(SOA)[3]というアーキテクチャパラダイムが注目されている．SOA では，システムの機能を「サービス」という単位でくくりだし，ビジネスプロセスの基本構成要素（要素プロセスと呼ぶ）とする．これらの「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なインターフェースを通じて疎結合される．ビジネスプロセスは既存サービスの緩い結合により構成され，個々のサービスの組み換えや更新は容易である．結果的に，ビジネス環境の変化に迅速・柔軟に対応できる．このような利点から，最近では特に，長年に渡り保守されてきた現役の「レガシーシステム」に SOA を適用する（SOA 化する）話題に大きな関心が集まっている．

SOA に基づくシステム開発では，まずビジネス環境を分析して要素プロセスを決定した後，これらの要素プロセスに合わせて，システム機能をサービスとしてマッピングする方法が知られている[4]．しかしながら，レガシーシステムには，長年の保守による保守性の低下や，変更による業務への影響が認められない，などの問題がある．このため，ビジネス環境に合わせた大幅な修正を行うことが困難になっている．

従って，レガシーシステムを SOA 化する際には，既

存システムの機能を最大限再利用することを考慮し，ビジネスの要素プロセスを決定することが必要である．

これを支援するため，本論文では，システムのソースコードを分析し,システム内の潜在的なサービス(候補）を抽出する手法を提案する．具体的には，まずソースコードに対してリバースエンジニアリングを適用し，データフローダイアグラム（DFD）を取得する．次に，DFD 上のデータを３つのカテゴリに分類し，プロセス間の依存関係を性質付ける．この依存関係に基づき，DFD 上の複数のプロセスを自己完結したサービスとしてくくりだす 6 つのルールを提案する．

提案手法を酒在庫管理システムに適用し，サービス抽出実験を行った．その結果，ソースコードからサービスを抽出できることを確認した．

2. 準備

2.1. 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）[3]とは，ソフトウェアの機能を「サービス」という単位でくくり出し，複数のサービスを連携・統合することでシステムを構築するソフトウェアアーキテクチャである．ここで，サービスとはソフトウェアで実行される処理の集合を指し，サービス提供者によってサービス利用者に対して提供される．サービスの定義に関しては様々なものが存在するが，本論文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）の集合と位置づける．

(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．つまり，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，外部から利用可能な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（要素プロセス）として機能する粗粒度の処理である．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．

これらの条件は，SOA における一般的なサービスに要求される必要条件である[2][4]. これらの条件を満たすソフトウェア処理は，Web サービス[8]等の標準的な SOA フレームワークでラップされ，正式にサービスとして外部公開される．これらの条件から，サービス同士の結合度は弱く保たれる(**疎結合**[10])．よって，サービス内部の変更がシステム全体に波及せず，サービスの組み換えも容易となる．その結果，ビジネス環境の変化に対応しうる柔軟なシステム環境を実現することが可能となる．

2.2. レガシーシステムの SOA 化

文献[4]において, SOA に準拠したシステム開発手法が提案されている．この手法では，まずビジネス環境のフロー分析を行い，業務の流れを抽象化・単純化したプロセスモデルとして表現する．さらに，プロセスをそれ以上細分化できない「要素プロセス」まで詳細化する．次に，システム化の側面から分析を行い，システムを構成する細かいコンポーネントやプログラムを適切な粒度のモジュールにまとめ，サービスとして要素プロセスに対応付ける．

しかし，新規開発の場合と異なり，レガシーシステムを SOA 化する場合には, この方法を直接適用することは難しいと考えられる．ビジネス環境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，対応するサービスをレガシーシステムから抽出するためにはシステム側の大幅な修正が必要となる可能性がある．しかし，レガシーシステムには，(a)長年に渡る保守作業の結果，更なる修正が難しい状態になっている[5], (b)現在も業務で使用されているため大幅な改修を加えることが難しい, といった問題がある．

従って，レガシーシステムを SOA 化する場合には，まずシステム側からの分析を行い，現行のシステム機能をできるだけ再利用する形で，要素プロセスとシステム機能とのすり合わせを行うボトムアップなアプローチが現実的であると考えられる．

2.3. レガシーシステムからのサービス抽出

レガシーシステムの SOA 化を支援するために, 本論文では以下の問題に取り組む．

サービス抽出問題

入力：システムのソースコード C．C は手続き型言語で書かれるものとする．

出力：サービスの集合 S={s1,s2,...,sn}．ただし，各 si は C で実装される処理の部分集合で，2.1節の条件 1 ～ 3 を満たす．

この問題の意義は，以下のとおりである．まず，入力として，レガシーシステムでは保守が滞りがちな関連ドキュメントは不要である．また，現行のコードにおける（潜在的な）サービスの候補を抽出できる．得られたサービス（候補）を基に，現行のシステムを最大限考慮した無理の無い要素プロセスの決定・対応付けが可能となる．

3. 提案サービス抽出法

3.1. キーアイデア

提案手法のキーアイデアは，データフローダイアグラム（DFD）を用いて，ソースコード C における処理

間のデータ依存解析を行い，サービスとして成立する処理群をくくり出すことである．提案するサービス抽出法は以下の４つのステップで構成される．

（STEP1）ソースコード C を DFD に変換する．
（STEP2）DFD 上のデータ（フロー）を分類する．
（STEP3）DFD 上の処理間に依存関係を設定する．
（STEP4）DFD に処理結合ルールを適用する．

なお，（STEP1）については，手続き型のソースコードをリバースエンジニアリングし，階層的 DFD を導出する既存の方法[1][6]を用いるため，詳細は本稿では割愛する．導出された DFD の各レイヤに対して，（STEP2）以降を適用する． DFD の表記法として，処理（以降プロセスと呼ぶ）を円で，プロセス間のデータの流れ（データフロー）を実線矢印，データストアを平行線で表す．

3.2. データの分類（STEP2）

サービスのオープンなインターフェース（2.1節，条件 2 参照）を実現するため，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 上のデータを以下の 3 種類に分類する．

(1)外部データ

システムの外部要素（アクター）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．コード上では，ファイルや標準入出力が該当する．DFD 上のデータフローに E とラベル付けする．

(2)システムデータ

システム内部で複数のプロセスが共通して利用するデータ．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．プロセスとデータストア間のデータフローのうち，複数のプロセスが共通利用するものはシステムデータに分類が可能で，DFD 上では S とラベル付けする．

(3)モジュールデータ

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．コード上ではローカル変数などが該当する．プロセスとデータストア間のデータフローのうち特定のプロセスのみが用いるものと，プロセス間で直接受け渡しされるものはモジュールデータに分類が可能で，DFD 上では M とラベル付けする．

3.3. 依存関係の設定（STEP3）

サービスの自己完結性（2.1節，条件 1 参照）を実現するため），DFD のプロセス間の依存関係を解析する．本稿では，以下の 3 種類の依存関係を設定する．表記上，プロセス間の依存関係を破線矢印で表す．以降，P1, P2 を任意のプロセスとする．

(1)順序依存

P2 を実行するための前提として， P1 を実行する必要がある場合には，処理順序による依存関係が認められる．これを P1 から P2 への順序依存と定義し，P1 から P2 へラベル O を付与した破線矢印を引く．

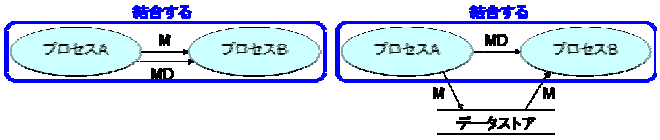

図 1　モジュールデータ依存プロセスの結合

図 2　システムデータ依存プロセスの分割

(2)データ依存

P1 の出力データが P2 の入力データになっている場合，P1 から P2 へのデータ依存を定義する．モジュール（又はシステム）データのやり取りによるデータ依存をモジュール（システム）データ依存と呼び，P1 から P2 へラベル MD（SD）付与した破線矢印を引く．ただし，P1 がデータストアにデータを出力しており，P2 が同じデータストアから入力を受けている場合，その入力が P1 の直接的な出力で無ければ，データ依存を設定しない．

(3)制御依存

P1 の出力によって，P2 の実行の有無が決定する（つまり，P1 が P2 の制御フラグとなる）場合，P1，P2 間に制御による依存関係が認められる．これを制御による依存と定義し，P1 から P2 へラベル C を付与した破線矢印を引く．

3.4. サービスの抽出（STEP4）

STEP3 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．P1 と P2 を 1 つのサービス内に結合する必要がある場合，両者の結合を結合プロセスと呼び，P1+P2 と書く．P1 と P2 が分割可能な場合，これらを分割プロセスと呼び，P1 | P2 と書く．なお，分割プロセスは結合することも可能である．以下に，プロセスの結合要否を判断する 6 つのルールを挙げる．

（ルール 1）モジュールデータ依存プロセスの結合

図 1 に依存関係 MD を有するプロセスからのサービス抽出ルールを示す．モジュールデータはシステム内の特定のプロセスが使用する，実装に極めて依存したデータである．プロセス A とプロセス B を別々のサービスに分割してしまうと，利用者がプロセス A,B をこの順に実行して，両者間の入出力データの受け渡しを代行しなければならない．このことはサービスの条件 1 に反する．また，この場合の入出力データはシステムの内部データであるため条件 2 にも反する．以上より，プロセス A とプロセス B は結

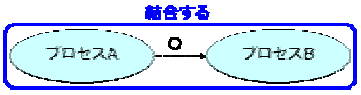

図 3　順序依存プロセスの結合

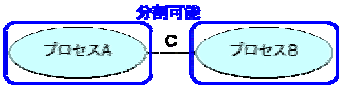

図 4　制御依存プロセスの分割

合する必要があり，（プロセス A）＋（プロセス B）が成立する．

（ルール 2）システムデータ依存プロセスの分割

図 2 に依存関係 SD を有するプロセスからのサービス抽出ルールを示す．システムデータはシステム内の複数のプロセスから参照および更新が可能である．中間のデータストアに適切なデータが格納されていれば，プロセス A とプロセス B とは非同期に実行可能である．そのため，プロセス A とプロセス B を別個のサービスに分割しても，サービスの条件 1，2 を充足可能である．以上より，プロセス A とプロセス B は分割が可能であり，（プロセス A）|（プロセス B）が成立する．なお，必要に応じて，両者を結合プロセスとすることも可能である．

（ルール 3）順序依存プロセスの結合

図 3 に依存関係 O を有するプロセスからのサービス抽出ルールを示す．処理依存が設定されている場合には，プロセス B はプロセス A が実行されていることを前提としており，プロセス A，プロセス B ともに単独で実行することは不可能である．このため，プロセス A とプロセス B を分割して別個のサービスとしてしまうと，サービス利用者側で実行順序を付ける必要があり，サービスの条件 1 に反する．以上より，プロセス A とプロセス B は結合する必要があり，（プロセス A）＋（プロセス B）が成立する．

（ルール 4）制御依存プロセスの分割

図 4 に依存関係 C を有するプロセスからのサービス抽出ルールを示す．プロセス A の出力はプロセス B を実行するか否かを判断する制御フラグであり，プロセス B の入力データにはならない．つまり，プロセス A はプロセス B を実行するための条件を満たすための処理を行っているわけではなく，実行の可否を判断しているだけである．このため，実行するための条件さえ整っているのであれば，単独でプロセス B を実行することも可能である．このため，プロセス A とプロセス B を別個のサービスとして分割しても条件 1，2 を充足可能である．以上より，プロセス A とプロセス B は分割可能であり，（プロセス A）｜（プロセス B）が成立する．

図 5　結合プロセスの合流

図 6　結合プロセスの連鎖

（ルール 5）結合プロセスの合流

図 5 に複数の結合プロセスが合流している場合のサービス抽出ルールを示す．結合プロセス A＋C と結合プロセス B＋C が存在する場合，プロセス C を実行する前提としてプロセス A とプロセス B が必要となる． A＋C と B＋C を別個のサービスに分割してしまうと，プロセス C を実行するために必要なデータや処理順序が満たされなくなる．以上より，プロセス A，プロセス B，プロセス C を結合する必要があり，（プロセス A）＋（プロセス B）＋（プロセス C）とする．本ルールの適用対象は，MD が発生しているプロセスと，O が発生しているプロセスである．

（ルール 6）結合プロセスの連鎖

図 6 に複数の結合プロセスが連鎖している場合のサービス抽出ルールを示す．結合プロセス A＋B と，結合プロセス B＋C が存在する場合，プロセス C を実行するためにはプロセス B が，プロセス B を実行するためにはプロセス A がそれぞれ必要となる．よって，プロセス C を実行するためにはプロセス A の実行が必要となる．以上より，プロセス A，プロセス B，プロセス C を一つのサービスとして結合する必要があり，（プロセス A）＋（プロセス B）＋（プロセス C）とする．本ルールの適用対象は，MD が発生しているプロセスと，O が発生しているプロセスである．

以上のステップを DFD の各レイヤに適用することで，様々な粒度のサービスが抽出できる．

4. ケーススタディ

4.1. 酒在庫管理システム

ケーススタディとして，酒在庫管理システム[9]の一実装（C 言語，約 800 行）に提案法を適用し，サービス抽出実験を行った．このソースコードでは，酒屋システムの機能として，積荷票処理と出庫依頼処理の 2 つが実装されている．本稿では積荷票処理について説明し，出庫依頼処理については割愛する．

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．図 7 に本処理の概要を示す．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．

図 7　積荷票処理の概要

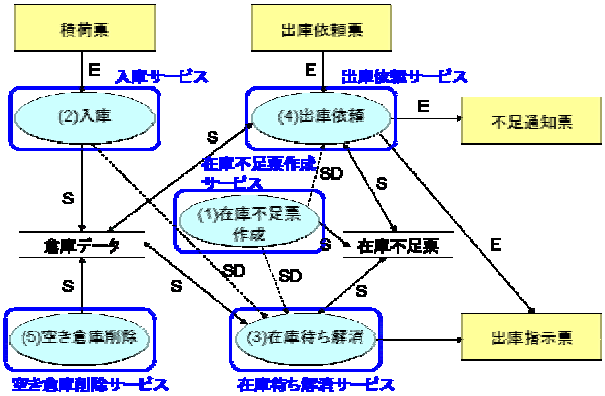

図 8　酒屋システム全体の DFD

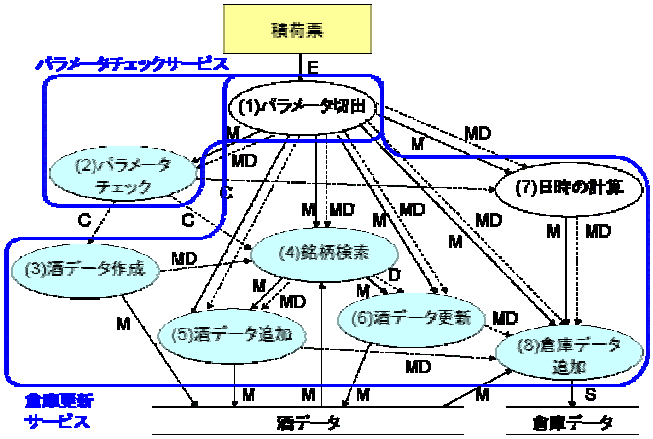

図 9　入庫プロセスの詳細 DFD

その後，在庫不足票を酒銘柄で検索し，在庫待ち注文を特定，在庫が充足されたら出庫処理を行う．出庫した銘柄については，在庫不足票から削除し，出庫依頼票を出力する．

4.2. 酒在庫管理システム全体への適用

まず，本実装から得られた DFD の上位レイヤに対して提案法を適用し，酒屋システム全体から粗粒度なサービス抽出を行った．図 8 にデータ分類および依存関係の分類を設定した DFD を示す．これら 5 つのプロセス(1)～(5)に対し，サービス抽出ルールを適用した．(ルール 2）により，(2)と(3)，(1)と(4)，(1)と(3)はいずれも分割が可能である．他に適用可能なルールはないので，(1) | (2) | (3) | (4) | (5)が酒屋システムの上位レイヤから抽出可能なサービスである（図中ではサービスを角なし枠で囲む）．それぞれ，「在庫不足票作成サービス」，「入庫サービス」，「在庫待ち解消サービス」，「出庫依頼サービス」，「空き倉庫削除サービス」となり，

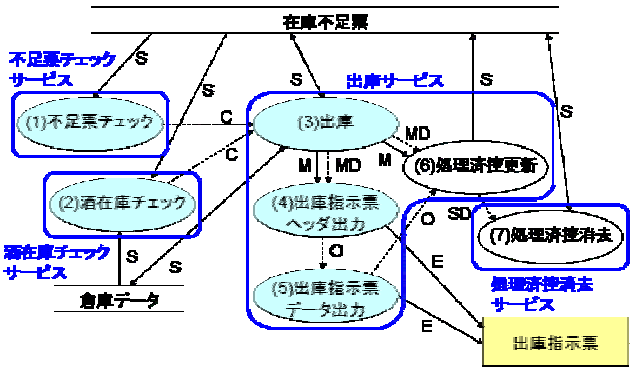

図 10　在庫待ち解消プロセスの詳細 DFD

直感的に，基本業務として成立するサービスが切り出せている．この実装では元来システムの主な処理が 5 つの結合の低いプロセスとしてうまく分割されていたことがわかる．

4.3. 入庫プロセスへの適用

図 8 の(2)入庫プロセスのさらに下位レイヤから，より細粒度のサービスを抽出した．図 9 に適用結果を示す．(1)から(8)のプロセスに対して（ルール 1），（ルール 5），（ルール 6）を適用すると，(1)＋(2) | (1)＋(3)＋(4)＋(5)＋(6)＋(7)＋(8)の 2 つの結合プロセスがサービスとして抽出された．すなわち，パラメータチェックプロセスは独立したサービスとして切り出せるが，残りの入庫プロセスは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．

4.4. 在庫待ち解消プロセスへの適用

同様に，図 8 の(3)在庫待ち解消プロセスの下位レイヤからサービスを抽出した．図 10 に適用結果を示す．(1)から(7)のプロセスに対して，（ルール 1），（ルール 3），（ルール 6）を適用すると，(3)＋(4)＋(5)＋(6)が成立する．（ルール 4）より(1)と(3)，(2)と(3)は分割可能で，（ルール 2）より(6)と(7)も分割可能である．結果的に 4 つのサービス(1) | (2) | (3)＋(4)＋(5)＋(6) | (7)が抽出された．それぞれ，「不足票チェックサービス」，「酒在庫チェックサービス」，「出庫サービス」，「処理済控え消去サービス」であり，SOA の観点からいずれも妥当なサービスが抽出されたと考えている．

5. 考察

5.1. 提案手法の特徴

提案手法は以下の特徴を持つ．

(a)サービス内部仕様の隠蔽

提案手法により抽出されるサービスでは，サービスを構成するプロセスの実行ロジックや，実装に依存した内部データといった内部仕様が隠蔽される．提案するサービス抽出ルールによって，決められた順序で実行しなければならないプロセスは同じサービス内に隠蔽され，その実行順序を外部利用者が意

識することはない．また，実装に強く依存するモジュールデータも外部から見えることはない．システムデータを入出力とするプロセスはサービスとして単独実行可能である．しかし，システムデータはシステムのサービス間で共有されるデータであり，外部データを入出力データとして操作が可能である．このようなプロセスを単独サービスとして外部に公開するのであれば，システムデータから外部データへの単純な変換サービスをラップすることで対応可能であると思われる．このようにしてサービスの内部仕様が隠蔽されているため，各サービスは他サービスとの依存を気にすることなく実行可能である．よって，条件１を満たす．同様に，実装に依存する内部データは入出力から排除される．従って条件２を満たす．なお，サービスの起動に関する条件は，Web サービス等の SOA フレームワークで抽出サービスをそのままラップすればよい．

(b) DFD の詳細度に応じたサービス抽出

提案手法では，ソースコードから得られた DFD に対して，適用すべき DFD レイヤを自由に選択することが出来る．上位**レイヤ**に対して適用した場合には抽出されるサービスの粒度は粗くなる．この場合抽出されるサービスは，下位レイヤのサービスをまとめた高機能なものとなる．下位レイヤに対して適用した場合には抽出されるサービスの粒度は細かくなり，低機能ではあるが再利用を想定したサービス抽出が期待できる．このようにして，様々な粒度のサービスが抽出可能である．従って，DFD の適度な詳細レベルを決定することにより，サービスの条件３を充足可能である．

(c)機械的なサービス抽出

提案手法では，プログラムに含まれる処理のデータ依存に着目してサービスを抽出している．また，データ依存の有無をデータの意味ではなく単純なデータ分類によって判定している．そのため，レガシーソフトウェアの詳細な仕様を理解することなく，機械的にサービスの抽出を行うことが可能となっている．これにより，ソースコードのみから効率よくサービスを抽出することが可能になる．しかしながら，ソースコードのみを情報源として各プロセスの詳細な意味を理解することは難しい．ビジネスやプロセスの意味解析にまで踏み込んでいないため，抽出される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の候補である．サービス候補のより正確な絞込みについては今後の課題としたい．

5.2. 関連研究

Ward の変換図[7]は DFD を拡張したもので，プロセス間の制御依存を破線矢印により表現できる．しかし，Ward の変換図では，もともとプロセスの依存解析を行う目的ことを目的としていないため，データ依存および順序依存を陽に指定できない．従って，そのままでは DFD からサービスを抽出する上での判断基準とすることは難しい．そこで，本研究ではプロセス間の依存関係を制御の発生原因毎に 3 種類に分類して，DFD を依存解析ツールとして拡張し，サービス抽出のツールとした．

6. 終わりに

本稿では，レガシーシステムのソースコードを DFD に変換し，データ依存の解析を行うことで，レガシーソフトウェアからサービスを抽出する手法を提案した．また，提案手法に基づき，酒在庫管理システムのプログラムからサービスを抽出した．

今後の研究では，サービスを利用する為のルールを検討することを計画している．提案手法により，様々な粒度のサービスを抽出可能になっているが，システム側で最適な粒度のサービスを判別するためのルールを考えたい．また，提案手法により抽出したサービスは自由に組み合わせることが可能だが，組み合わせの妥当性については判断が必要であろう．これらの課題を解決できるようなサービス利用フレームワークの開発に取り組んでいく予定である．

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

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

[4] 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.

[5] 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.

[6] 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.

[7] P.T. Ward, "The transformation schema: An extension of the data flow diagram to represent control and timing", IEEE Trans on Software Eng, Vol.12, pp.198-210, 1986.

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

[9] 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.

[10] 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.