NAIST-IS-MT0451048

修士論文

プログラム処理間の依存解析に基づく
レガシーソフトウェアからのサービス抽出法

木村 隆洋

2006 年 2 月 2 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士(工学) 授与の要件として提出した修士論文である。

木村 隆洋

審査委員：

松本 健一 教授	(主指導教員)
関 浩之 教授	(副指導教員)
門田 暁人 助教授	(副指導教員)

プログラム処理間の依存解析に基づく
レガシーソフトウェアからのサービス抽出法*

木村 隆洋

内容梗概

サービス指向アーキテクチャ(SOA)は，ネットワーク上に分散するシステムを柔軟に連携・統合するための，新しいシステムアーキテクチャとして注目されている．しかしながら，長年保守が繰り返され現在も稼動中のレガシーシステムに対して，SOAを適用することは容易ではない．

レガシーシステムへの効率的なSOA適用を支援するため，本論文では，手続き型言語で書かれたレガシーソフトウェアからサービス(候補)を抽出する手法を提案する．具体的には，まずレガシーソフトウェアのソースコードから，データフローダイアグラム（DFD）を取得する．次に，DFD上に存在するデータフローを3種類に分類し，プロセス間の依存関係を性質付ける．さらにこの依存関係に基づき，依存の強いプロセス群を結合しサービスとして抽出するための6つのルールを提案する．提案手法によって，自己完結，オープンなインターフェース，任意の粒度という，SOAの要件に合致したサービス候補を体系的に導出することが可能となる．

また，提案手法の有効性を検証するために，既存のアプリケーションからサービスを抽出するケーススタディを行った．その結果，既存アプリケーションのソースコードから様々な粒度のサービスを抽出可能であることを確認した．

キーワード

DFD, SOA, 手続き型プログラム，プロセス依存関係，プロセス結合

---

*奈良先端科学技術大学院大学情報科学研究科情報システム学専攻修士論文,
NAIST-IS-MT0451048, 2006 年2 月2 日.

Extracting Services from Legacy Software Based on Dependency Analysis among Processes in Program*

Takahiro Kimura

Abstract

The service-oriented architecture (SOA) is a new system architecture that allows flexible integration and orchestration of distributed heterogeneous systems. However, it is not straight-forward to apply the SOA to the existing legacy systems, which have been maintained for a long time and are still being used.

To support efficient adaptation of legacy software for the SOA, this paper presents a method that systematically extracts services (candidates) from source codes of a procedural system. Specifically, I first obtain data flow diagrams (DFDs) from the given source codes with the existing reverse-engineering technique. Then, I define dependencies among processes on the DFDs by classifying the data into three categories. Based on the dependencies, I apply six rules that aggregate several processes on the DFDs into a service. With the proposed method, one can systematically derive service candidates that satisfy desirable properties of SOA; self-contained, open interface, and multi-level granularity.

To illustrate the effectiveness of the proposed method, I have conducted a case study for an existing legacy application. As a result, it was shown that services with various granularities were identified reasonably from the legacy application.

Keywords:

DFD, SOA, Procedural Program, Process Dependency Relation, Process Binding

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0451048, February 2, 2006.

目次

図目次

表目次

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[13]．しかし，長年に渡り保守されてきた現役の「**レガシーシステム**」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，**サービス指向アーキテクチャ(SOA)**[6]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)と対応させる．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．その結果，ビジネス環境の変化に迅速・柔軟に対応することができる．最近では特に，このような利点から，レガシーシステムを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対応付けて開発する方法が知られている[7]．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[8][9]や，変更による業務への影響が認められない，というような制約がある．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は困難になっている． 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方法が有効であると考えられる．

そこで本論文では，既存のシステムのソースコードを分析し，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4種類に分類，

設定する．この依存関係に基づき，DFD上の複数のプロセスをサービスとして抽出する6つのルールを提案する．

ケーススタディとして，提案手法を酒在庫管理システムの複数の実装に適用し，サービス抽出を行った．その結果，酒在庫管理システムのソースコードからサービスを抽出することが出来た．

本論文の構成は以下の通りである．第2章では，SOAの特徴およびサービスの満たすべき条件について説明した後，レガシーシステムをSOA化する上での課題を述べる．第3章では，提案手法の手順を述べた後に各手順の具体的な内容を説明する．第4章では，提案手法の有効性を確認するために実施したケーススタディの概要および結果を述べる．第5章では提案手法の特徴についての考察と，関連研究について述べる．最後に第6章で，まとめと今後の研究課題について述べる．

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）[6]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，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．SOAに基づくシステムでは複数のソフトウェアの機能を**疎結合**することにより，ビジネス環境の変化に伴うシステムの変更に柔軟に対応することが容易となる．次に，旅行計画システムの例を用いてSOAに基づくシステムの特徴を述べる．

旅行計画システムは，航空券手配システムとホテル予約システムの独立した2システムを組み合わせたものである．旅行計画システムを利用した業務プロセスとして，航空券手配およびホテル予約の2プロセスが独立して存在している．航空券手配プロセスは，空席検索および発券からなっており，ホテル予約プロセスは空室検索および予約からなる．航空券手配，ホテル予約の2プロセスを統合して個人旅行手配プロセスとして，空席および空室が確認できたら発券および予約を実行するように変更したい．

このような場合に，**レガシーソフトウェア**により構成されるシステム（レガシーシステム）ではシステムの構成要素が密に結合しているため，一部の変更が他の部分に波及してしまう．レガシーソフトウェアとは，長年に渡って運用，保守され続けているソフトウェアのことであり，機能追加や仕様変更，修正によりソフトウェアの規模や複雑さが増大した状態になっている[8][9]．図1はレガシーソフトウェアにより構成された旅行計画システムである．個人旅行手配プロセスを構築する際には空席検索と発券を分割する必要があるが，空席検索機能と発券機能が密に結合しているため，航空券手配システムを修正しなければならない．同様に，ホテル予約システムも空室検索と予約を分割するため修正が必要となる．システムの変更に伴いシステムとのインターフェースも変更が必要となる．図2はプロセス統合後の旅行計画システムである．このように，レガシーシステムは

変化に柔軟に対応することが難しいという問題がある．

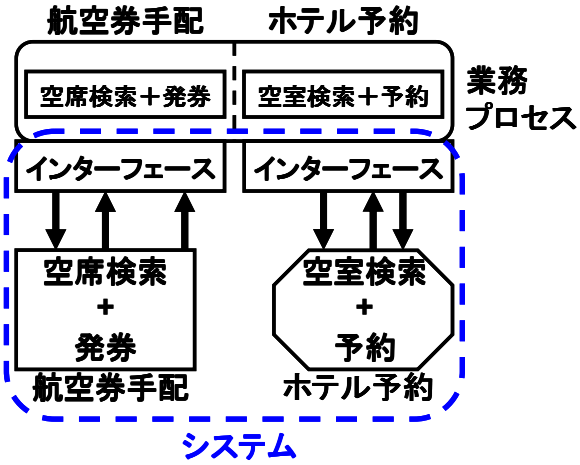

図 1　プロセス統合前の旅行計画システム（レガシーソフトウェア）

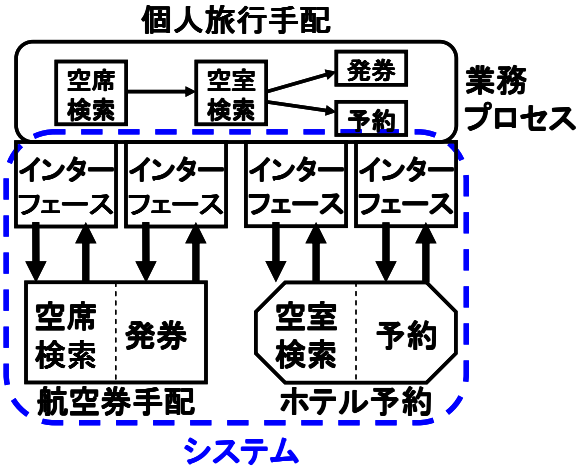

図 2　プロセス統合後の旅行計画システム（レガシーソフトウェア）

一方，SOAに基づくシステムでは，航空券手配システムの機能を空席検索サービス，発券サービスとして，ホテル予約システムの機能を空室検索サービス，予約サービスとして外部に公開する．図3はSOAに基づく旅行計画システムである．旅行計画プロセス変更前は，航空券手配プロセスとして空席検索サービス実行後に発券サービスを実行し，ホテル予約プロセスとして空室検索サービス実行後に

予約サービスを実行していた．航空券手配プロセスとホテル予約プロセスの統合はサービスを空席検索，空室検索，発券および予約という順序で実行するように変更することによって実現するため，プロセスの変更がシステムに波及することはない．また，サービスのインターフェースは，一度公開した後は原則として変更が認められないため，仮にシステム側で変更が発生した場合であっても，その変更が旅行計画プロセスに波及することはない．図4はプロセス統合後の旅行計画システムである．このようにして，ソフトウェアの機能を疎結合することにより，業務プロセスの変化により柔軟に対応することができるようになる．

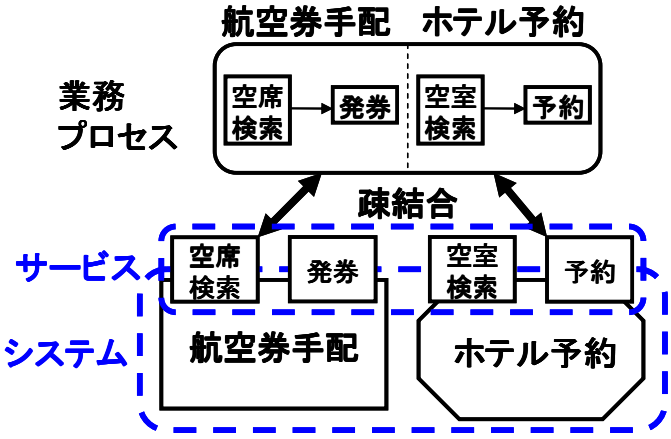

図 3　SOA に基づく旅行計画システム（プロセス統合前）

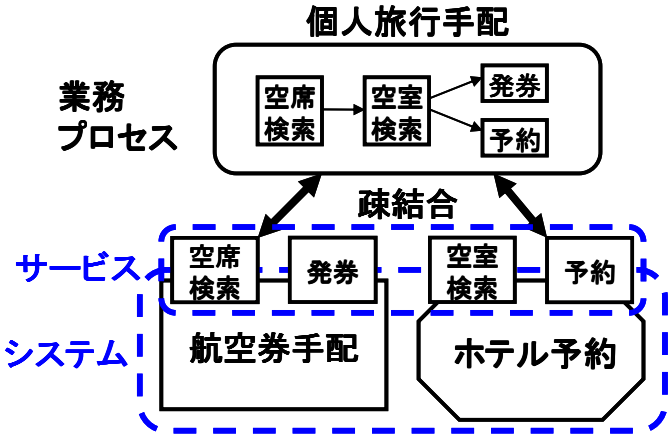

図 4　SOA に基づく旅行計画システム（プロセス統合後）

2.2　サービスの 3 条件

サービスの定義に関しては様々なものが存在する[4][7]が，本論文ではサービスを**自己完結**，**オープンなインターフェース**，**任意の粒度**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．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．これらの条件を満たすソフトウェア処理は，Webサービス[14]等の標準的なSOAフレームワークでラッピングされ，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[18])．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．以下にサービスのそれぞれの条件について述べる．

2.2.1　(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，サービスは単独で実行することや，繰り返し実行すること，他のサービスと自由に組み合わせて利用することが出来なければならない．よって，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処理はサービスとして認められない．

図5は図3で示した旅行計画システムにおけるサービスの呼び出し順序である．旅行計画プロセスを構築する際には，航空券手配システムおよびホテル予約システムが公開するサービスを自由に組み合わせて実行することが可能である．この場合，空席検索サービス，空室検索サービスを実行した後に発券サービス，予約サービスを実行している．

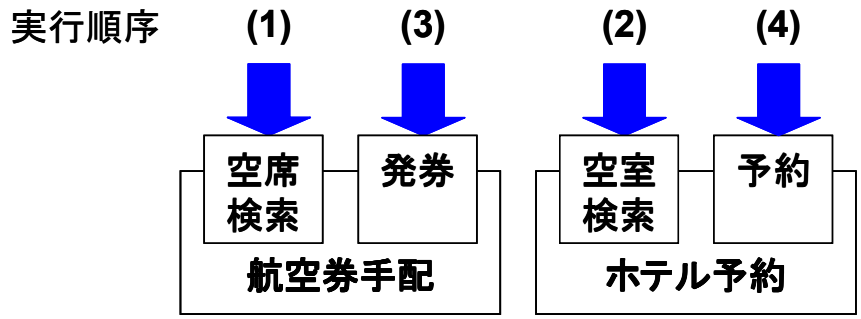

図 5　自己完結の例

2.2.2　（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．オープンなインターフェースとは，サービスを公開するシステムの実装に依存せず，サービス利用者のデータまたは他システムのデータを入出力データとして指定可能なインターフェースのことである．このため，プラットフォームや言語に依存する呼び出し方法を要求するような処理や，サービスを提供する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オープンなインターフェースを備えない．

図6は図3で示した旅行計画システムを利用する際に指定する入力データの例である．空席検索サービスを利用する際にフライト番号を利用しているが，この入力データは外部から指定することが可能である．よって，オープンなインターフェースの要件を満たす．しかし，空室検索サービスで指定するホテルデータベースのキー値は，ホテル予約システムの内部実装に依存するデータである．よって，オープンなインターフェースの要件を満たさない．

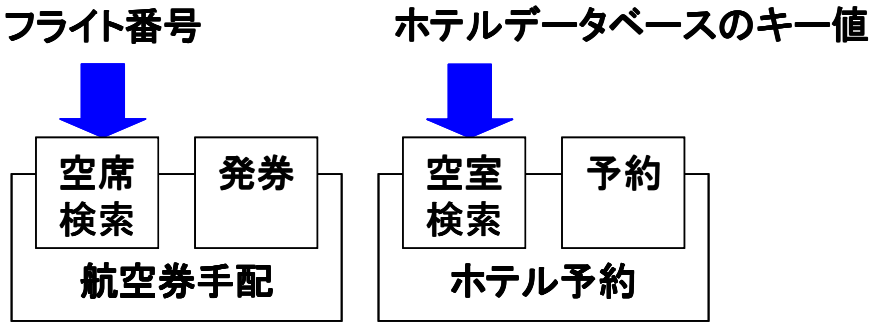

図 6　旅行計画システムへの入力データ

2.2.3　（条件 3)任意の粒度

サービスの粒度は任意である．条件1，条件2を充足するサービスは粒度が粗くなる（粗粒度）傾向があるが，サービスを利用する業務によって最適な粒度は異なる．このため，業務処理に応じて任意のサービスの粒度を選択できるようにすることが必要である．

図7は図3で示した航空券手配システムが公開するサービスの利用例である．サービス利用者が粒度の粗いサービスを希望する場合は，空席検索サービスと発券サービスを組み合わせた航空券手配サービスを実行することが可能である．逆に，粒度の細かいサービスを希望する場合は，空席検索サービス，発券サービスを単体で実行することが可能である．

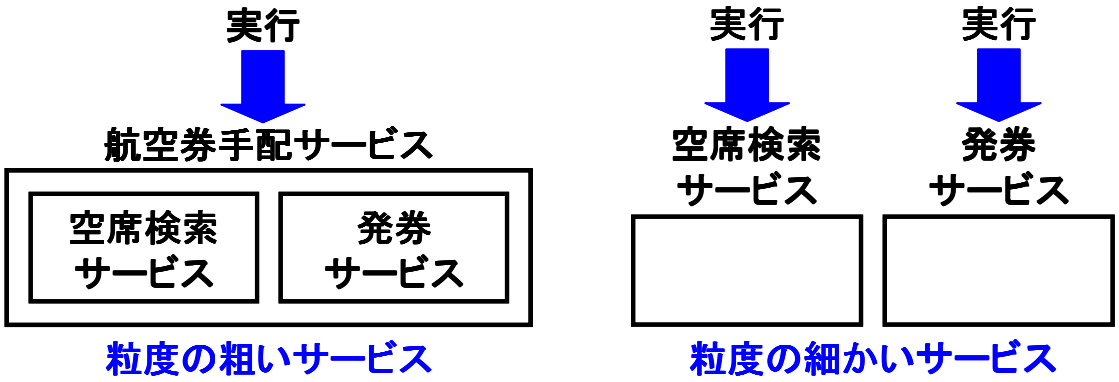

図 7　任意の粒度の例

2.3　レガシーシステムの SOA 化手法

文献[7]において，SOAに基づく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．

図8は2.2節で述べた旅行計画システムをSOAに基づき新規に開発する際の手順である．ビジネス環境を分析して，旅行計画プロセスを作成する．この旅行計画プロセスを詳細化することで，空席検索，発券，空室検索，予約の4つの要素プロセスを抽出する．これらの要素プロセスに対応付けする形で航空券手配システム，ホテル予約システムを開発し，システムの機能をサービスとして公開する．

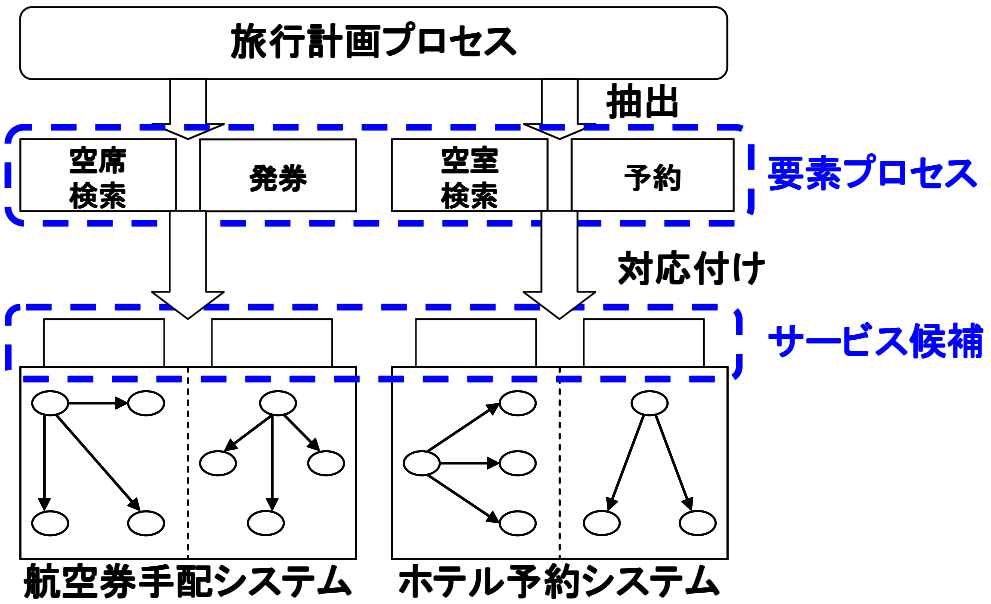

図 8　SOA に基づくソフトウェア開発手順

このような業務フローを中心としたトップダウンなアプローチは，新規にSOAに基づくシステムを開発することを想定している．このため，既にシステムが存在しているレガシーシステムをSOA化する場合には，このアプローチを直接適用

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，レガシーシステムは，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おり[8][9]，また，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． 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の方が現実的であると考えられる．

図7は既に存在する旅行計画システムをSOA化する際の手順である．まず，航空券手配システム，ホテル予約システムを分析して空席検索サービス，発券サービス，空室検索サービス，予約サービスを候補として抽出する．一方で，旅行計画プロセスから要素プロセスを抽出する．そして，サービスの候補と要素プロセスのすりあわせを行い，要素プロセスとサービス候補の乖離がある場合にはどのようにして対応するかを検討する．

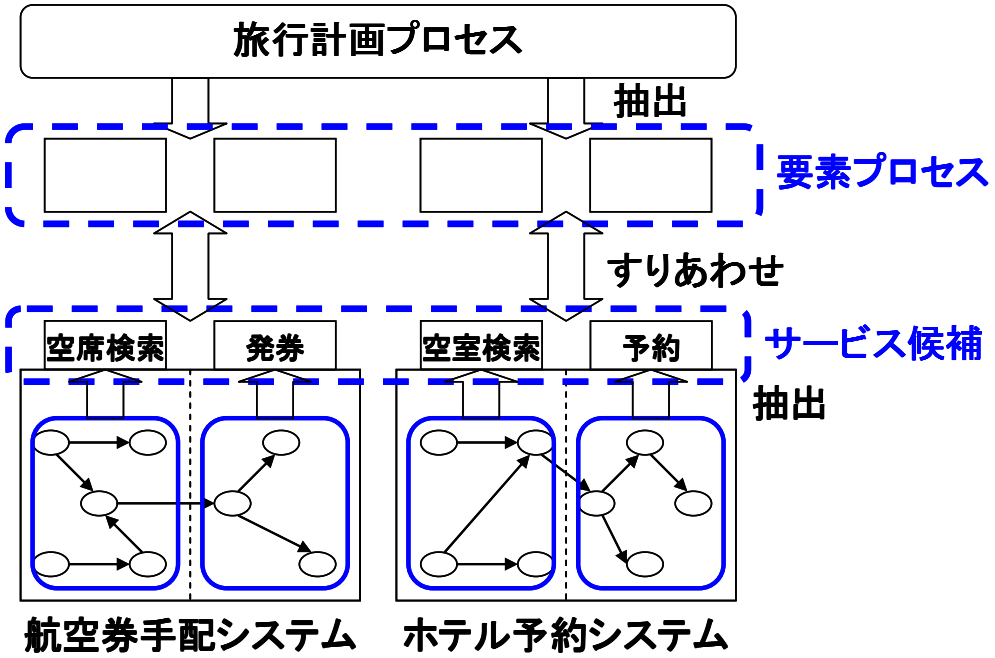

図 9　レガシーソフトウェアへの SOA 適用手順

このようにして，事前にレガシーソフトウェアから抽出可能なサービスを明らかにすることで，業務フローの分析を行う際の参考や方針決定の材料にすることが可能となる．例えば，業務フローから抽出可能な要素プロセスの候補が複数ある場合には，レガシーソフトウェアから抽出したサービスを使いやすくなるようなものを選択することが可能となる．また，レガシーシステムの実装を極力変更したくない場合には，あらかじめ抽出したサービスの候補を指針として，業務フローから要素プロセスを抽出することが可能であろう．逆に，業務フローを基準とする必要があり，そのためにレガシーソフトウェアを改修しなければならない場合にも，抽出したサービスと要素プロセスの差分が明確になるので，改修方針の決定やコスト，期間の見積もりに際して利用することが可能であろう．

2.4　レガシーシステムからのサービス抽出

現状ではレガシーソフトウェアをSOA化する体系的な手法は存在しない．そこで，レガシーシステムのSOA化を支援するために，本論文では以下の課題に取り組む．

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，2.2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，レガシーソフトウェアのソースコードのみを入力としているため，保守が滞りがち[12]な要求仕様書や設計書等の関連ドキュメントは不要である．また，現行のソースコードからサービス候補を抽出することができる．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要素プロセスの決定・対応付けが可能である．更に，COBOLやC言語のような手続き型言語により記述されたレガシーソフトウェアは数多く存在するため，手続き型言語を対象とすることは理にかなっていると考えられる．

2.5　酒屋の在庫問題

本論文では，サービス抽出手法を提案するにあたり，例題として酒屋の在庫問題[16]を用いる．酒屋の在庫問題はプログラミング教育などで共通の例題として提案されたものである．酒屋の在庫問題では，酒屋の倉庫管理が主題となっている．本論文では，酒屋の在庫を管理するシステムを酒在庫管理システムと呼ぶ．

酒在庫管理システムでは，受付係，配送係，倉庫係の3者の間で伝票のやり取りがなされ，酒の在庫管理が行われる．具体的には，入庫，在庫不足解消，出庫，在庫不足登録の4つの処理が行われる．以下でそれぞれの処理について述べる．

2.5.1　入庫

入庫処理とは，倉庫に酒を搬入し，搬入した酒のデータを積荷票に蓄積する処理である．図10は入庫処理の概要図である．入庫処理では，(1)倉庫係が倉庫に酒を搬入した後に，(2)受付係に積荷票を送付する．(3)受付係は積荷票を蓄積，管理する．

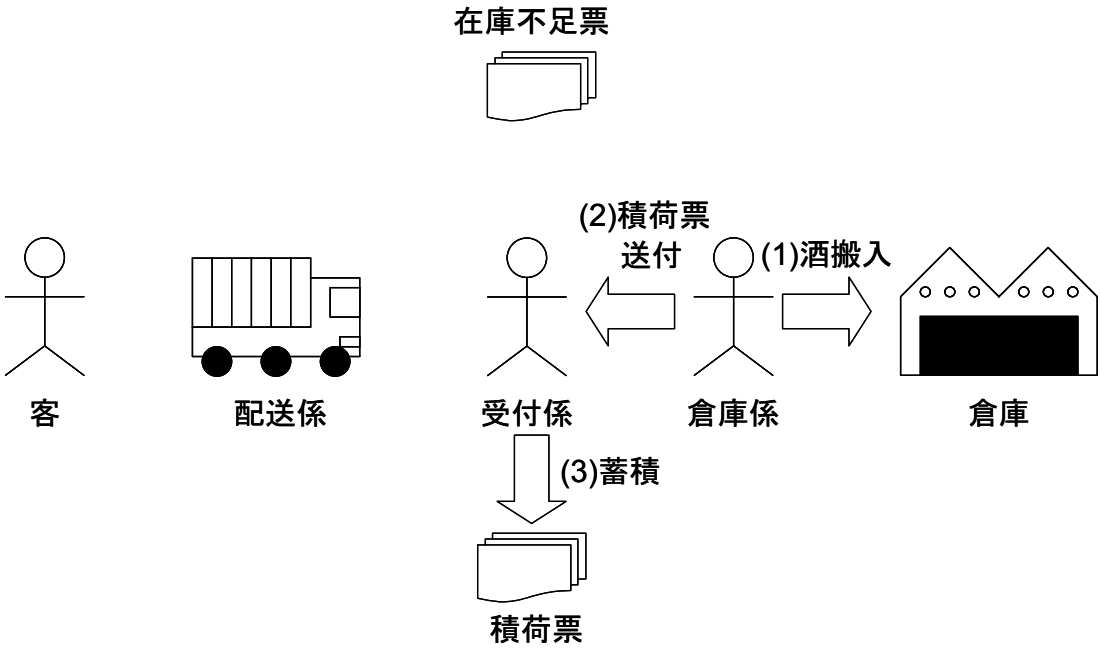

図 10　入庫処理の概要

2.5.2　在庫不足解消

在庫不足解消処理とは，入庫処理終了後に在庫不足票をチェックして，出庫可能な酒を出庫する処理である．図11は在庫不足解消処理の概要図である．在庫不足解消処理では，(1)受付係は在庫不足票の有無を確認し，(2)出庫可能な酒があれば倉庫係に出庫指示票を送付する．(3)出庫した銘柄に対応する積荷票を削除する．最後に(4)出庫した酒に対応する在庫不足票を削除する．

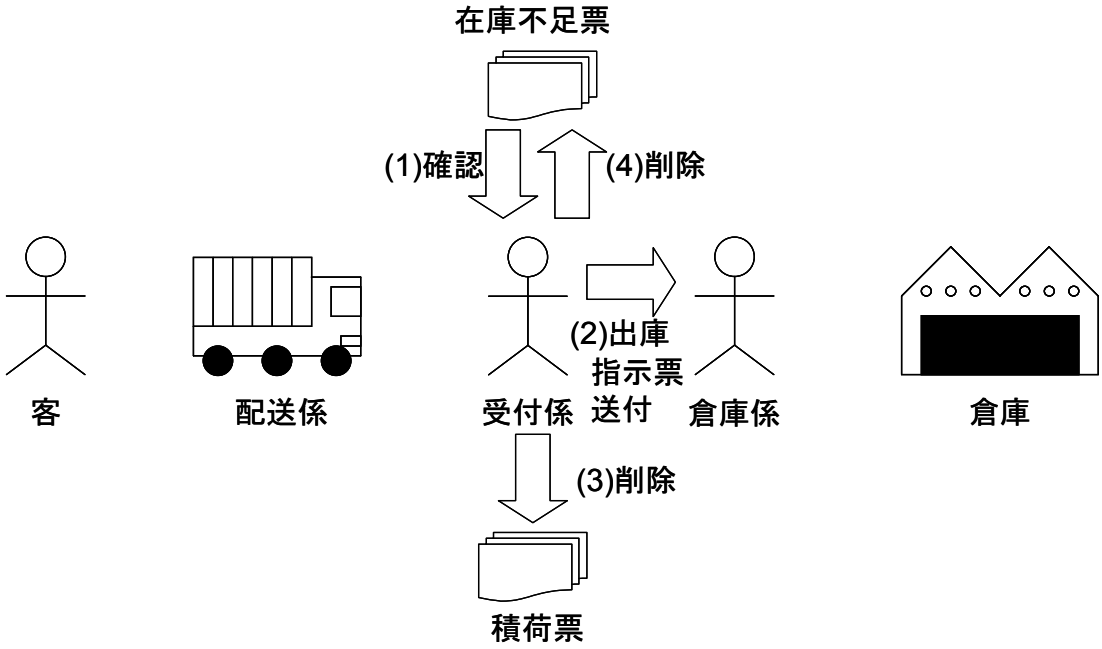

図 11　在庫不足解消処理の概要

2.5.3　出庫

出庫処理とは，客から注文を受けた酒を倉庫から出庫する処理である．図12は出庫処理の概要図である．出庫処理では，(1)配送係が客からの注文を受けると，(2)受付係に出庫依頼票を送付する．(3)受付係は倉庫係に出庫指示票を送付

する．(4) 出庫した銘柄に対応する積荷票を削除する．

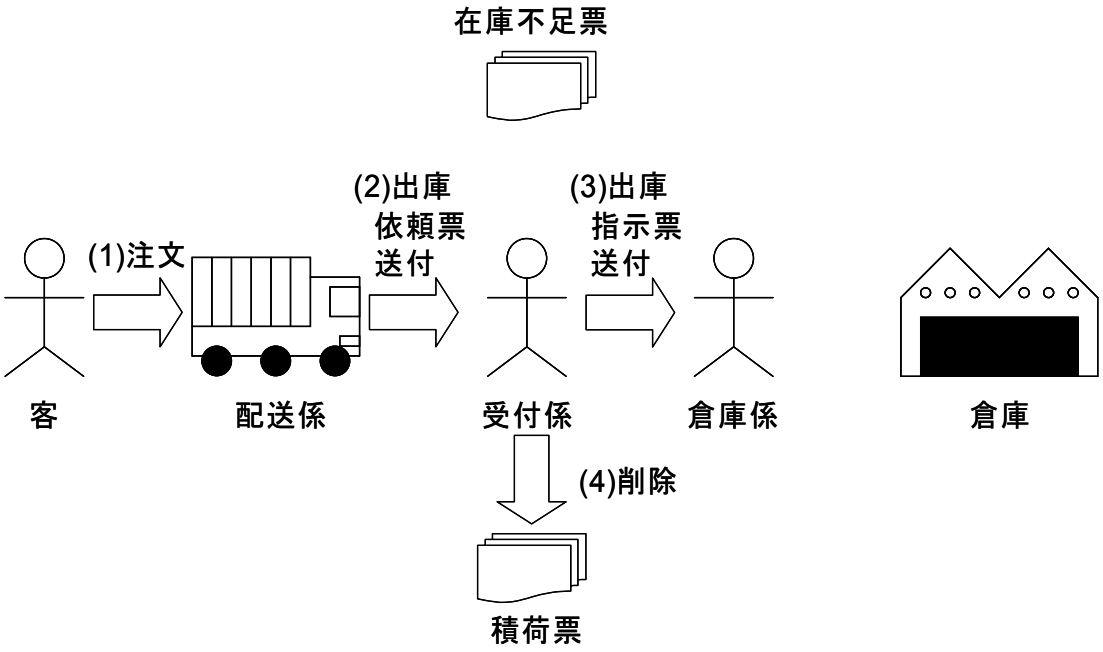

図 12　出庫処理の概要

2.5.4　在庫不足登録

在庫不足登録処理とは，客から注文を受けた銘柄の在庫がない場合に，在庫不足票を蓄積する処理である．図13は在庫不足登録処理の概要図である．在庫不足登録処理では，(1)配送係が客からの注文を受けると，(2)受付係に出庫依頼票を送付する．倉庫に十分な在庫がない場合に，(3)配送係に不足通知票を送付する．(4)在庫不足票を蓄積し，管理する．

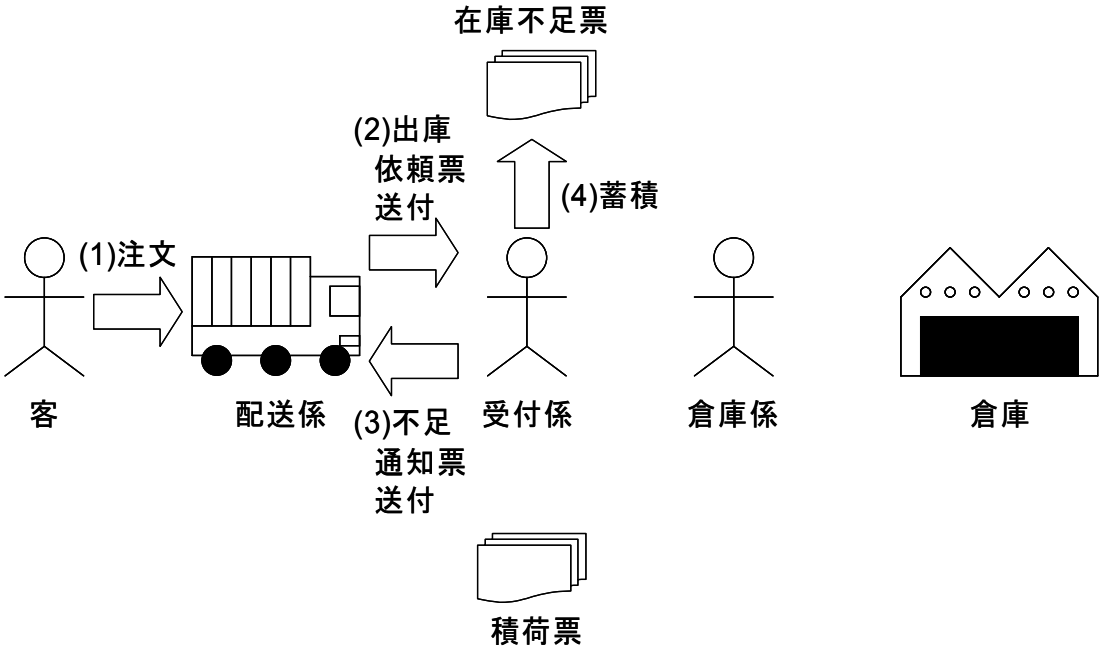

図 13　在庫不足登録処理の概要

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における処理間の依存解析を行い，サービスとして成立する処理群を抽出することである．提案手法は以下の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上の処理間を流れるデータを分類する．
（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．

提案するSTEP1からSTEP4までの手順を図14に示す． DFDは処理間のデータフローを表現するものであり，データの流れに着目して処理間の依存関係を解析するのに適している．また，DFDは階層毎に適用が可能であるため，様々なレイヤで処理間の依存解析を行うことが可能である．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][10]を用いて手動でDFDに変換する．よって，詳細は本論文では割愛する．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．

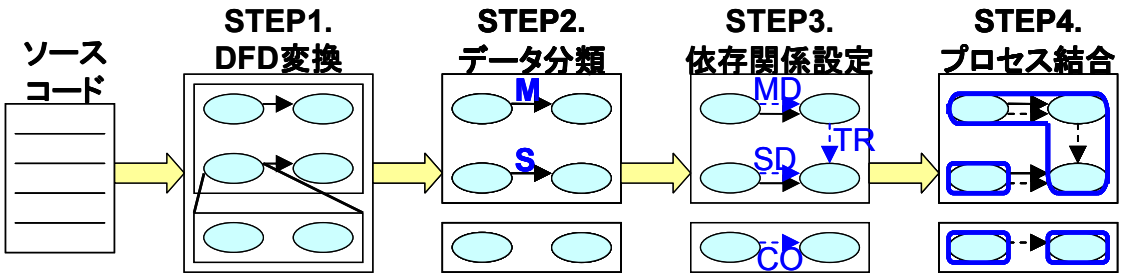

図 14　提案手法の概要

本章では，サンプルプログラムを用いて提案手法について説明する．サンプルプログラムは2.5節にて述べた酒在庫管理システムの一部で，2.5.2節の在庫不足解消処理に該当する．サンプルプログラムでは，入庫後に在庫不足になっている銘柄をチェックして，在庫があり出庫可能な酒の銘柄の出庫処理を行い，出庫した銘柄について出庫指示票を作成する．サンプルプログラムは以下の7つの処理から成り立っており，各々の処理をDFDのプロセスと対応付ける．

(1)不足票チェック

在庫不足票をチェックし，不足を解消しなければならない銘柄を取得する．

(2)酒在庫チェック

倉庫データを取得し，在庫不足になっている銘柄が出庫可能な状態であるかチェックする．

(3)出庫

出庫手続きを行い，出庫済の銘柄のデータを出庫対象コンテナに蓄積する．

(4)出庫指示票ヘッダ出力

出庫手続きを行った銘柄の一覧（ヘッダ部分）を出力する．

(5)出庫指示票データ出力

出庫手続きを行った銘柄の一覧（銘柄データ部分）を出力する．

(6)処理済控え更新

出庫手続きを行った銘柄に対応する在庫不足票を処理済に更新する．

(7)処理済控消去

在庫不足票をチェックし，処理済になっているものを削除する．

図15に在庫不足解消プログラムのソースコードを示す．

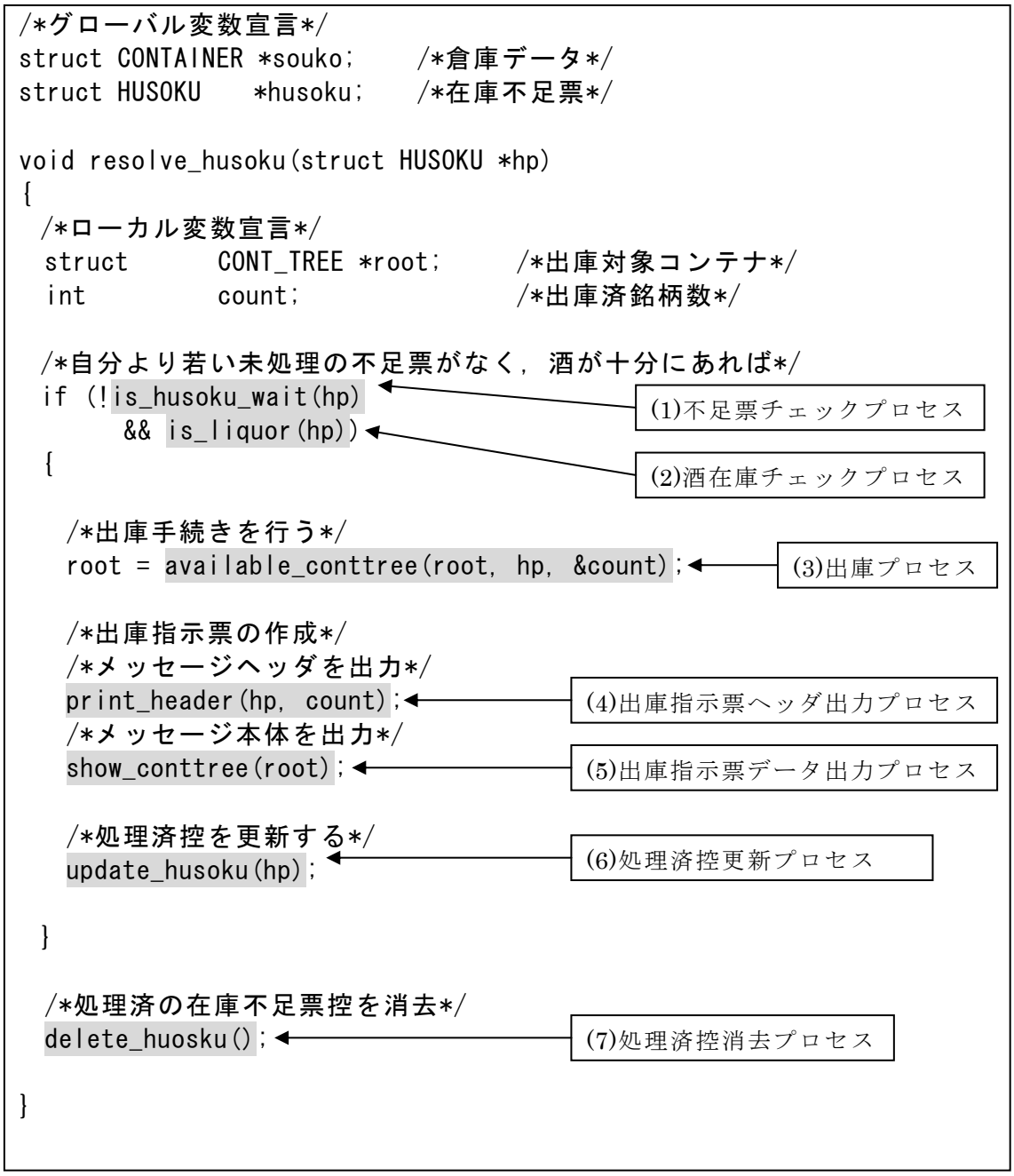

図 15　在庫不足解消プログラムのソースコード

図16は在庫不足解消プログラムのソースコードをDFDに変換したものである.

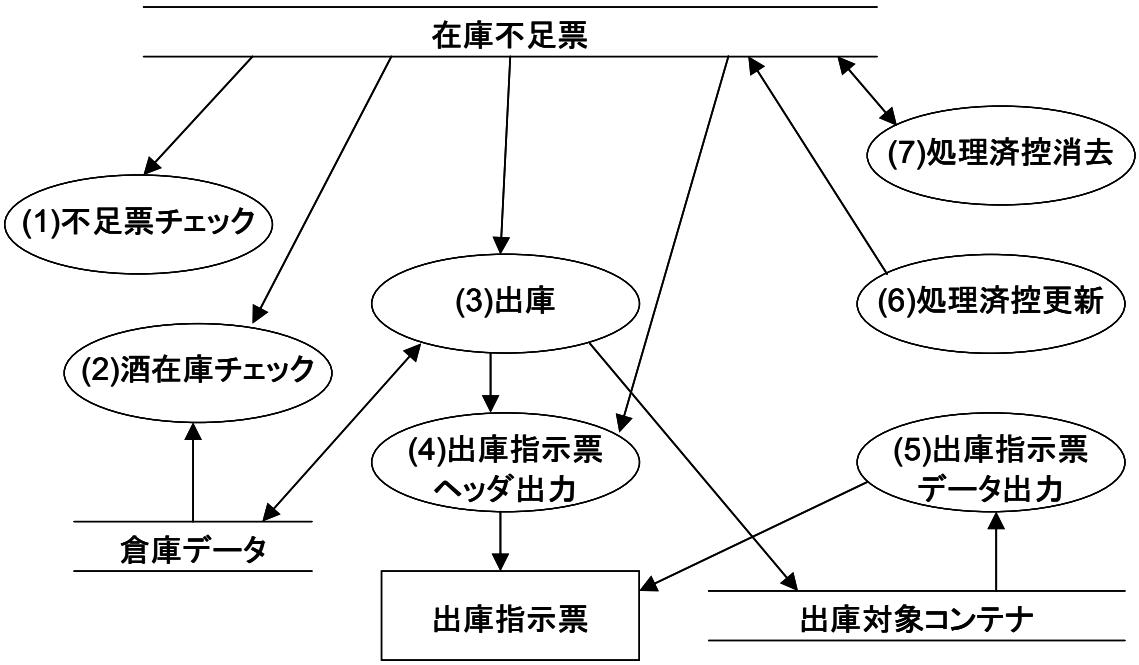

図 16　在庫不足解消プログラムの DFD

3.2　DFD 上のデータの分類(STEP2)

サービスのオープンなインターフェース（2.2節の条件2を参照）を実現するためには,プロセス間を流れるデータの種類を明らかにする必要がある.ここでは,プロセス間のデータを**外部データ**，**モジュールデータ**，**システムデータ**の3種類に分類する.

3.2.1　外部データ

外部データとは，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のことである.

外部データが成立するための要件は,アクターとプロセス間でデータフローが発生することである.プログラムコード上では，ファイルや標準入出力が該当する. DFD上では，データフローにEとラベル付けする.

図17は図16で示したDFDに外部データのラベル付けを行ったものである.(4)出庫指示票ヘッダ出力プロセス，(5)出庫指示票データ出力プロセスと出庫指示票の間にデータフローが発生している.いずれもプロセスとアクター間のデータフローであるから，Eとラベル付けする.

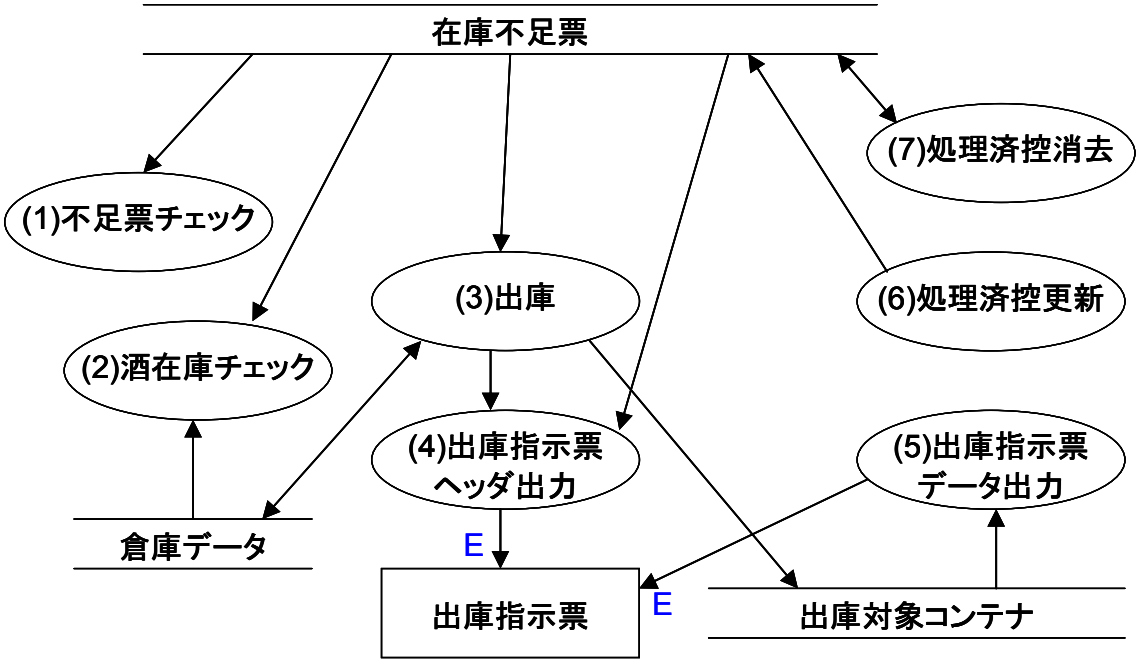

図 17　外部データのラベル付け

3.2.2　モジュールデータ

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．モジュールデータは同一のレイヤに属するプロセス間でのみ利用することが可能であり，他のレイヤに属するプロセスからはそのデータは使用不可能である．

モジュールデータが成立するための要件は，(ⅰ)プロセス間のデータフローが存在する，(ⅱ)プロセスとデータストア間のデータフローが存在し，そのデータストアは特定のレイヤ内でのみ有効である，のいずれかを満たすことである．プログラムコード上ではローカル変数や，グローバル変数のうちスコープが対象システムのソースコードの一部にのみ及ぶものなどが該当する．DFD上では，データフローにMとラベル付けする．

図18は図16で示したDFDにモジュールデータのラベル付けを行ったものである．(4)のプロセスの入力データであるcount変数を(3)のプロセスが出力しているため，(3)のプロセスから(4)のプロセスにデータフローが発生している．プロセス間のデータフローはモジュールデータに該当するため，Mのラベル付けをする．

また，(3)のプロセスから出庫対象コンテナへのデータフローが発生し，出庫対象コンテナから(5)のプロセスへのデータフローが発生している．このデータフローは(3)のプロセスが出庫対象コンテナにデータを出力し，(4)のプロセスが出庫対象コンテナのデータを入力としているために発生している．出庫対象コンテナはプログラム上はローカル変数（root変数）であるため，モジュールデータである．よって，Mのラベル付けをする．

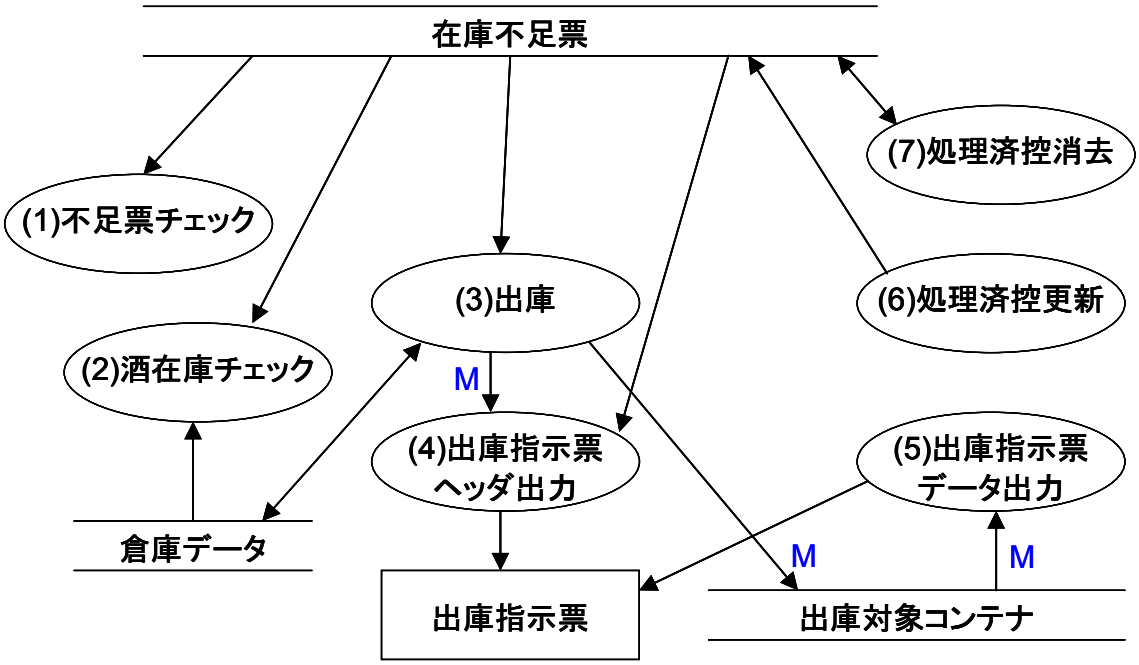

図 18　モジュールデータのラベル付け

3.2.3　システムデータ

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．システムデータは異なるレイヤに属するプロセス間で利用することが可能である．システムデータが成立するための要件は，プロセスとデータストア間のデータフローが存在し，データストア上のデータは全レイヤのプロセスから利用が可能であることである．プログラムコード上ではデータベースに入出力するデータや，グローバル変数のうちスコープが対象システムのソースコード全体に及ぶものなどが該当する．DFD上では，データフローにSとラベル付けす

る．

図19は図16で示したDFDにシステムデータのラベル付けを行ったものである．在庫不足票と(1)，(2)，(3)，(4)のプロセスの間にデータフローが存在するが，(1)，(2)，(3)，(4)のプロセスの入力データであるhp変数を経由して在庫不足票のデータを取得しているためである．(6)のプロセスと在庫不足票の間にデータフローが存在するが，これはhp変数を経由して在庫不足票にデータを出力しているためである．(7)のプロセスと在庫不足票の間にデータフローが存在するが，これはhp変数を経由して在庫不足票のデータの取得および出力をしているためである．在庫不足票はプログラム全体をスコープとするグローバル変数husokuであるため，システム内の全プロセスからアクセスすることが可能であり，システムデータに該当する．よって，Sのラベル付けをする．

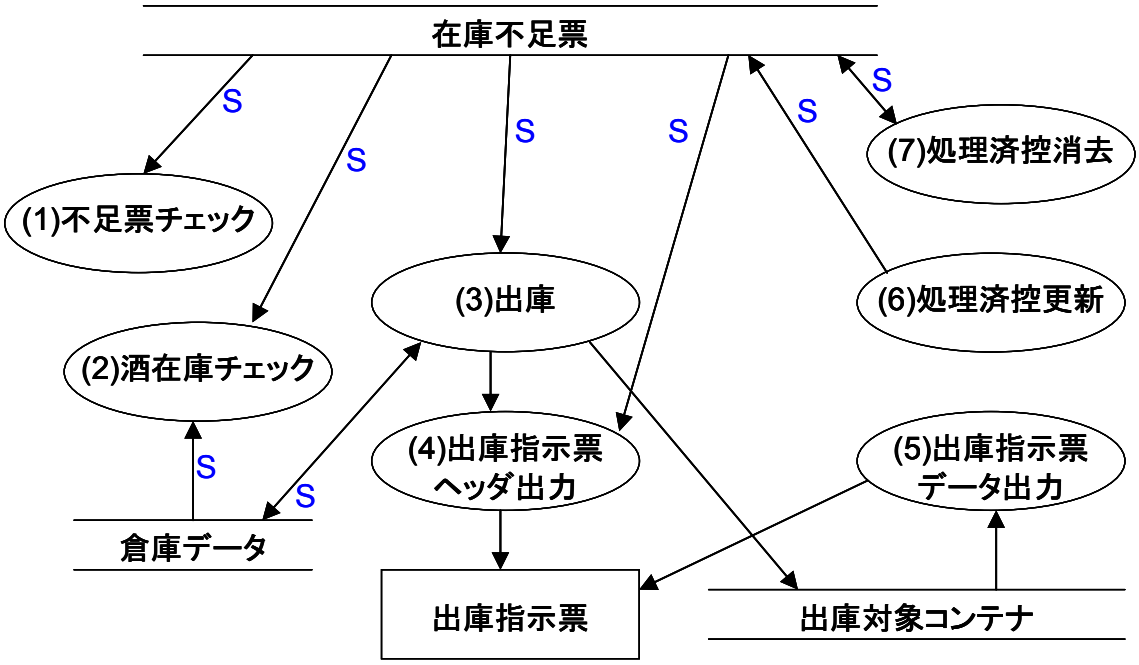

図 19　システムデータのラベル付け

最後に，図16で示した在庫不足解消プログラムのDFDの全データフローに対し，データ分類を行った結果を図20に示す．

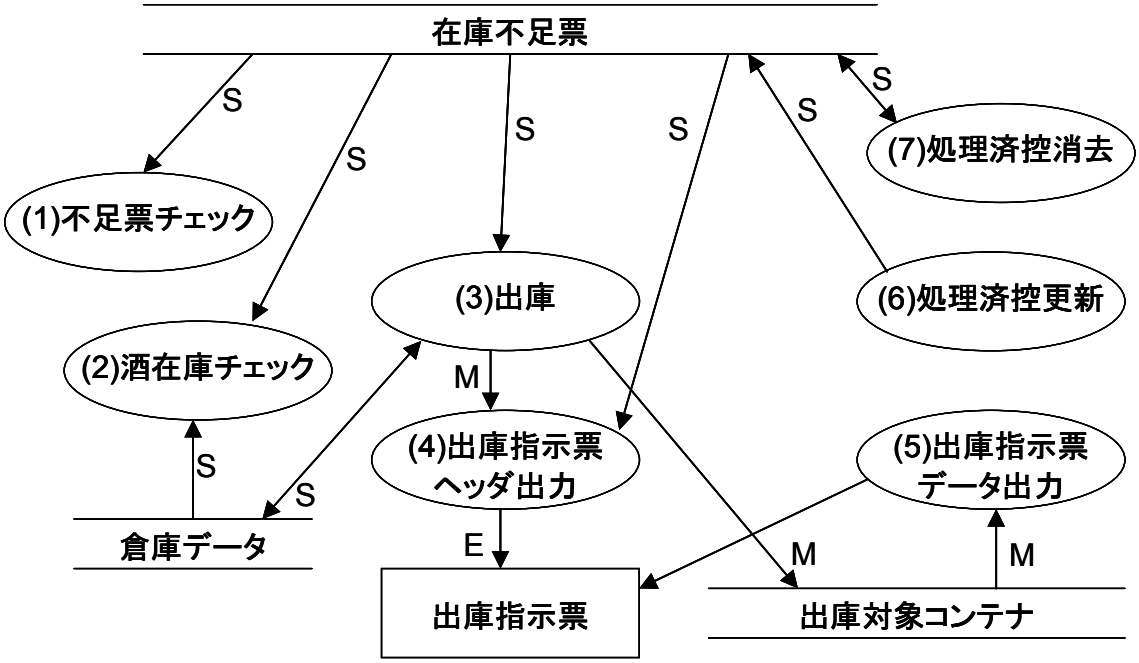

図 20　データ分類後の在庫不足解消プログラムの DFD

3.3　プロセス間の依存関係の設定（STEP3）

サービスの自己完結性（2.2節の条件1を参照）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．依存関係を解析する際に，STEP2のデータフロー分類を利用する．本論文では，**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種類の依存関係を設定する．

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，プロセス間の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，プロセス間の制御により発生する依存関係のことである．DFDの表記上，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．

プロセス間に複数の依存関係が認められる場合には，より強い依存関係を優先して設定する．依存関係の優先順位を図21に示す．最も強い依存関係はモジュールデータ依存である．次に強い依存関係は処理依存であり，その次がシステムデータ依存，最も依存関係が弱いのは制御依存である．例えば，P1とP2の間にシステムデータ依存が設定するためには，P1とP2の間にモジュールデータ依存，処理依存が存在しないことが必要となる．

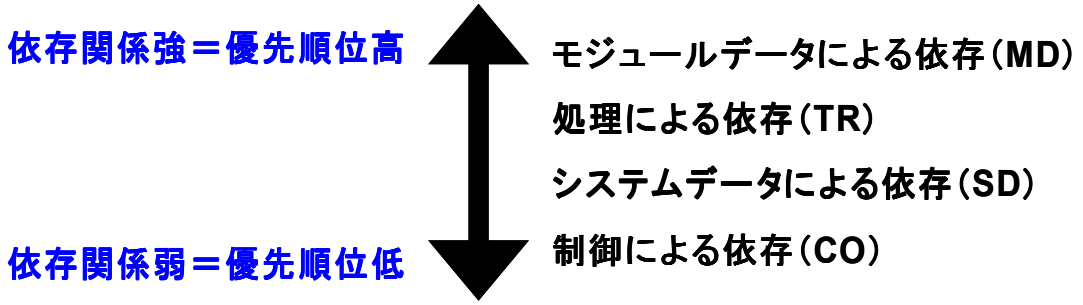

図 21　依存関係の優先順位

3.3.1　モジュールデータ依存（Module Data Dependency）

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

を設定する．モジュールデータのやり取りによるデータ依存をモジュールデータ依存と呼ぶ．DFD上ではP1からP2に破線矢印を引き，MDとラベル付けする．

モジュールデータ依存が成立するための要件は，P1，P2間で直接またはデータストアを介して，モジュールデータの入出力によるデータフローが発生していることである．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であっても，その入力がP1の直接的な出力で無ければ，モジュールデータ依存を設定しない．

プロセスP1，P2間のモジュールデータ依存を以下の論理式で定義する．以降，プロセスP，Qがあって，PからQに対してラベルxによるデータフローが存在する場合，$P \xrightarrow{x} Q$と記述する．

P1からP2にモジュールデータ依存が存在 $\Leftrightarrow \exists d : d \in MD \wedge P1 \xrightarrow{MD} P2$

以降，P1からP2にモジュールデータによる依存が存在する場合，MD(P1，P2)と書く．

図22は図20で示したDFDにモジュールデータ依存を併記したものである．(3)のプロセスと(4)のプロセスの間にはモジュールデータによるデータフローが存在するため，モジュールデータ依存を設定する．(5)のプロセスは(3)のプロセスが出庫対象コンテナに出力したデータを入力とする．出庫対象コンテナへのデータフローはモジュールデータであるため，モジュールデータ依存が成立する．よって，(3)のプロセスと(4)のプロセスの間および(3)のプロセスと(5)のプロセスの間に破線矢印を引き，MDとラベル付けをする．

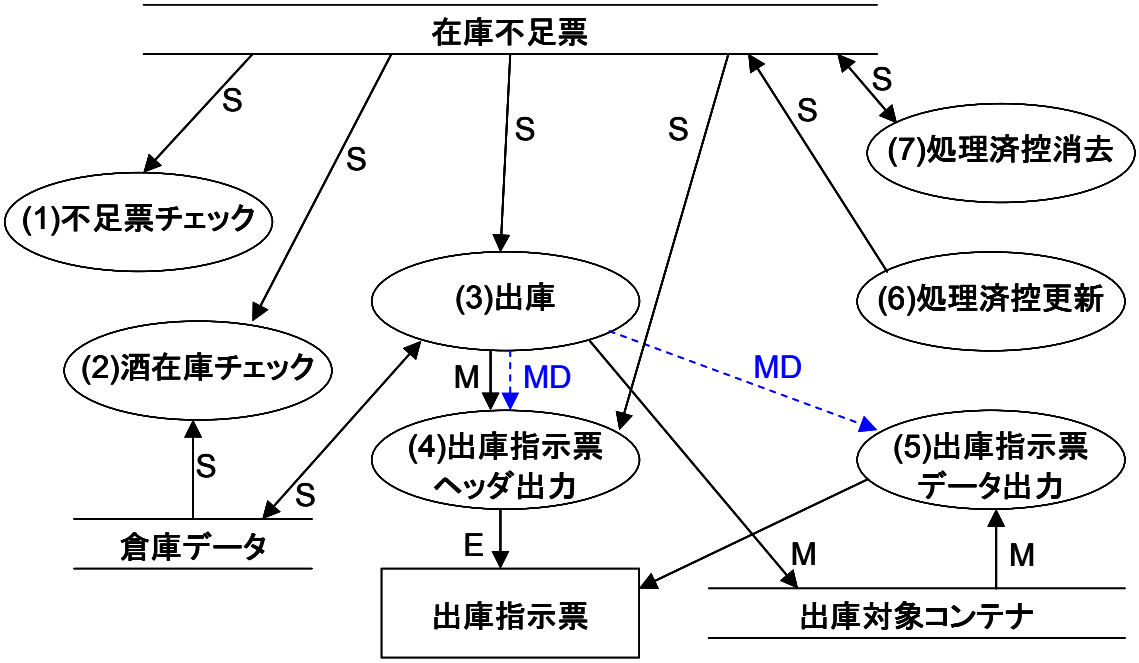

図 22　モジュールデータ依存設定後の DFD

3.3.2　処理依存（Transaction Dependency）

P1とP2の同期を取って実行しなければならない場合，P1とP2の間に処理依存を設定する．DFD上はP1とP2の間に破線矢印を引き，TRとラベル付けする．P1を実行後にP2を実行しなければならない場合には破線矢印はP1からP2への一方向とし，P1とP2の実行順はどちらが先でも良いが，同期を取って実行しなければならない場合には破線矢印は双方向とする．

処理依存が成立するための要件は，P1とP2を同一の**トランザクション**内で実行する必要があること，P1とP2の間にモジュールデータ依存が存在しないこと，の2点である．ここでいうトランザクションとは，複数の処理を一貫性を保持した形で実行することである[17]．　P2を実行するための前提としてP1の実行が必要な場合，P1とP2のどちらを先に実行しても良いが，同一のトランザクション内で実行することが必要な場合などが処理依存に該当する．

プロセスP1，P2間に存在する処理依存を以下の論理式で定義する．以降，P1，P2が同一のトランザクション内で実行しなければならない処理である場合，TX(P1，P2)と書く．

P1 と P2 の間に処理依存が存在 ⇔ ¬MD(P1, P2) ∧ TX(P1, P2)

以降，P1とP2の間に処理依存が存在する場合，TR(P1, P2)と書く．

図23は図20で示したDFDに処理依存を併記したものである．(6)のプロセスでは，(3)のプロセスで出庫処理を行った銘柄について在庫不足票を処理済に更新する．(3)のプロセスと(6)のプロセスで入力となる銘柄は同一であるため，(3)のプロセスと(6)のプロセスのどちらを先に実行しても差し支えない．つまり，在庫不足票を処理済に更新した後に，出庫処理を行うことも可能である．しかし，(3)のプロセスと(6)のプロセスは同一のトランザクション内で実行する必要がある．また，(3)のプロセスと(6)のプロセスの間にはデータフローが存在しないため，モジュールデータ依存は成立しない．このため，(3)のプロセスと(6)のプロセスの間には処理依存が成立し，双方向の破線矢印を引き，TRとラベル付けする．(4)のプロセスと(5)のプロセスは共に出庫指示票を作成する処理であるが，(4)のプロセスは共通ヘッダを，(5)のプロセスで実データを出力する．ヘッダを出力後に実データを出力しないと在庫不足票としての形式を成さないため，(5)のプロセスを実行するための前提として(4)のプロセスを実行する必要がある．また，(4)のプロセスと(5)のプロセスの間にはデータフローが存在しないため，モジュールデータ依存は成立しない．よって，(4)のプロセスと(5)のプロセスの間に処理依存が成立し，(4)のプロセスから(5)のプロセスに向けて破線矢印を引き，TRとラベル付けする．

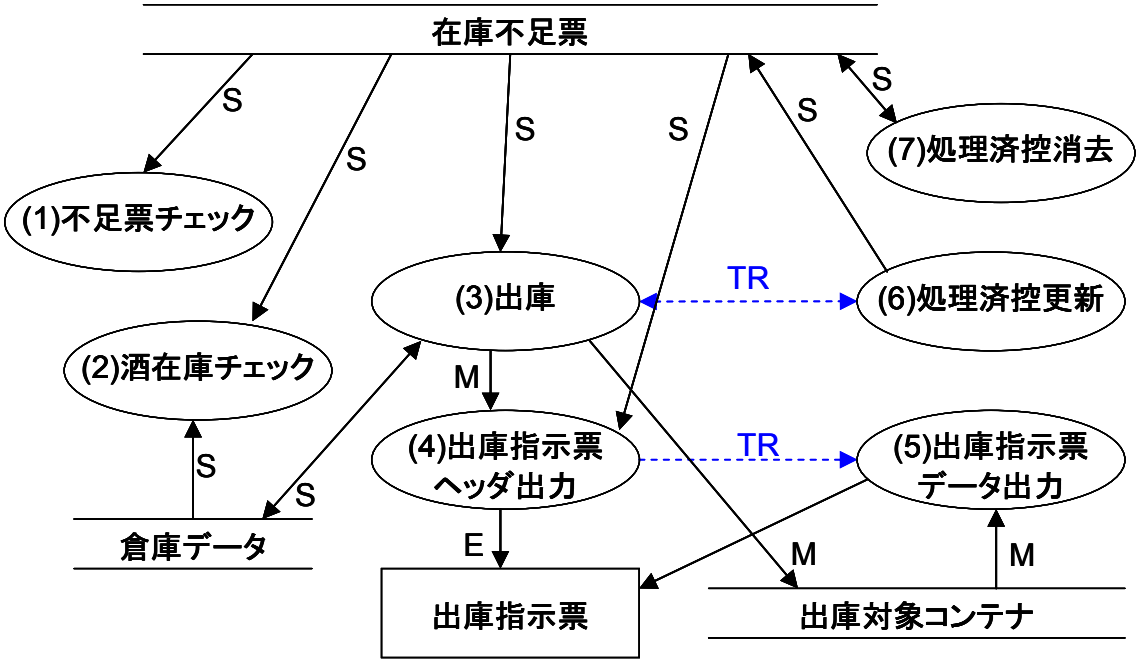

図 23　処理依存設定後の DFD

3.3.3　システムデータ依存（System Data Dependency）

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へのデータ依存を設定する．システムデータのやり取りによるデータ依存をシステムデータ依存と呼ぶ．DFD上はP1からP2に破線矢印を引き，SDとラベル付けする．

システムデータ依存が成立するための要件は，P1，P2間でデータストアを介して，システムデータによるデータフローが発生していること，モジュールデータ依存および処理依存が成立していないこと，の2点である．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であっても，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．

プロセスP1，P2間に存在するシステムデータ依存は以下の論理式により求められる．

P1からP2にシステムデータ依存が存在

$$\Leftrightarrow \neg MD(P1,\ P2) \wedge \neg TR(P1,\ P2) \wedge P1 \overset{SD}{\rightarrow} P2$$

以降，P1からP2にシステムデータによる依存が存在する場合，SD(P1, P2)と書く．

図24は，図20で示したDFDにシステムデータ依存を併記したものである．(7)のプロセスは(6)のプロセスが在庫不足票に出力したデータを入力とし，在庫不足票へのデータフローはシステムデータである．(6)のプロセスと(7)のプロセスの間にはモジュールデータ依存は存在しないため，システムデータ依存が成立する．よって，(6)のプロセスと(7)のプロセスの間に破線矢印を引き，SDとラベル付けする．

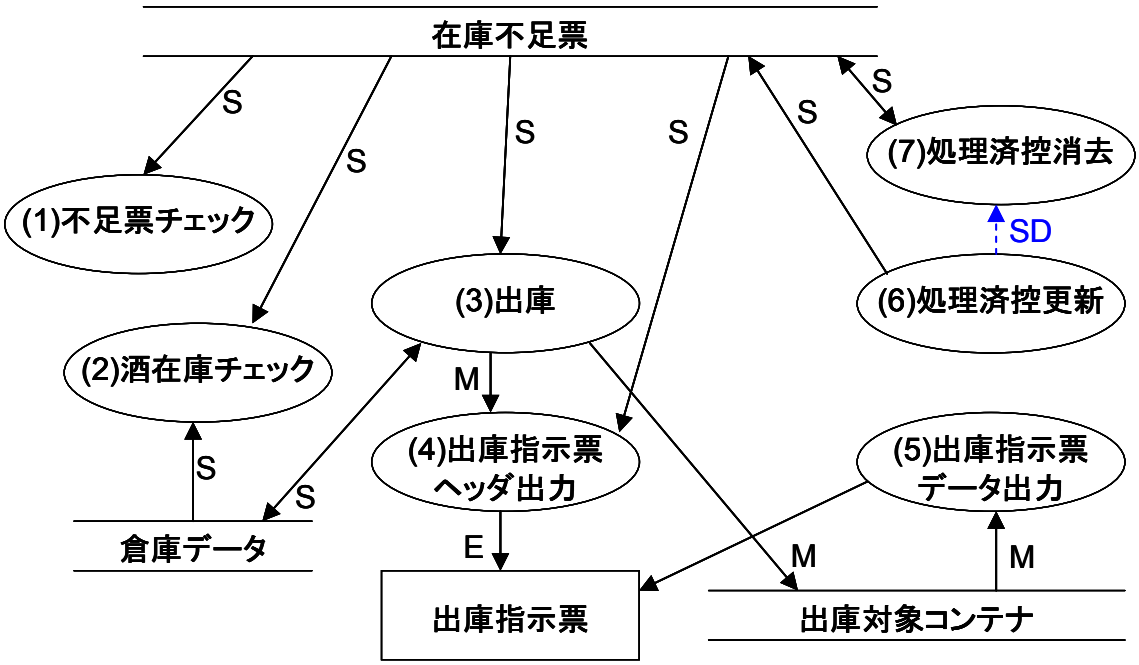

図 24　システムデータ依存設定後の DFD

3.3.4　制御依存（Control Dependency）

P1の出力によって，P2の実行の有無を決定する（つまり，P1がP2の制御フラグとなる）場合，P1とP2の間に制御依存を設定する．DFD上はP1からP2へ破線矢印を引き，COとラベル付けする．

制御依存が成立するための要件は，P1の実行結果によりP2の実行有無に影響が及ぶ実装になっていること，P1とP2の間にモジュールデータ依存，システムデー

タ依存，処理依存のいずれも存在しないことの2点である．

プロセスP1, P2間に存在する制御依存は以下の論理式によって表される．以降，P1の実行結果によりP2の実行有無が制御される場合，IF(P1, P2)と書く．

P1 から P2 に制御依存が存在 ⇔

$$\neg MD(P1, P2) \wedge \neg SD(P1, P2) \wedge \neg TR(P1, P2) \wedge IF(P1, P2)$$

以降，P1からP2に制御依存が存在する場合，CO(P1, P2)と書く．

図25は図20で示したDFDに制御依存を併記したものである．プログラム上は，(1)のプロセスおよび(2)のプロセスの出力によって(3)のプロセスを実行するか否かが制御されている．また，(1)のプロセス，(2)のプロセス，(3)のプロセスの間にはモジュールデータ依存，処理依存，システムデータ依存のいずれも成立しない．よって，(1)のプロセスと(3)のプロセスの間および(2)のプロセスと(3)のプロセスの間に制御依存が成立するため，破線矢印を引き，COとラベル付けする．

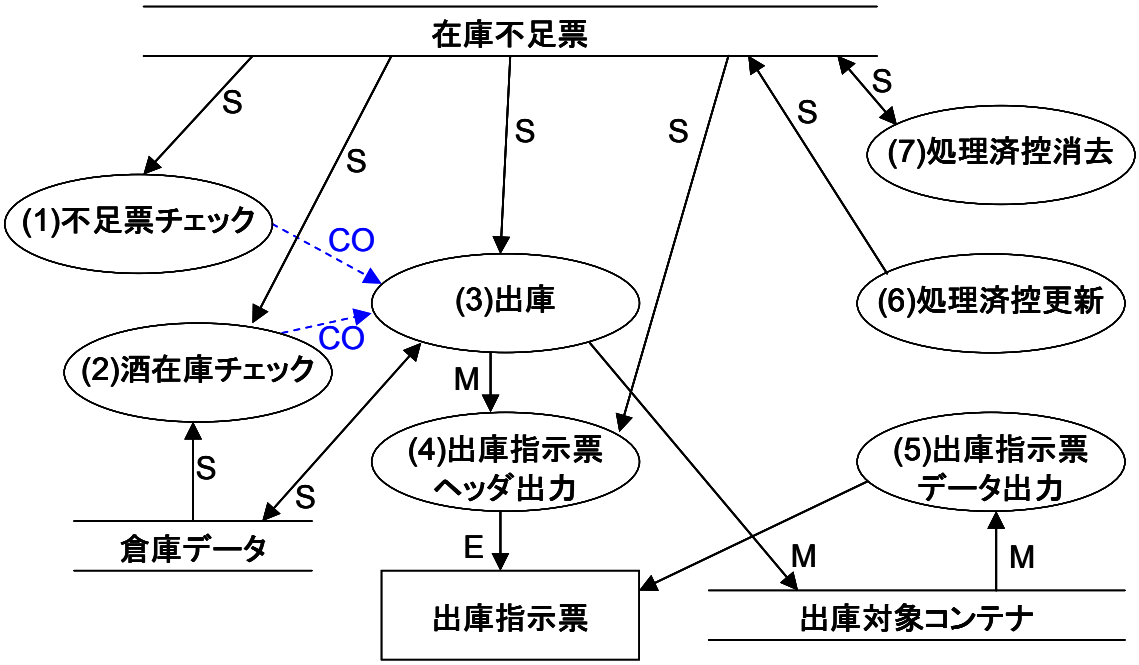

図 25　制御依存設定後の DFD

最後に，図20で示した在庫不足解消プログラムのDFDに対し，全ての依存関係を設定した結果を図26に示す．

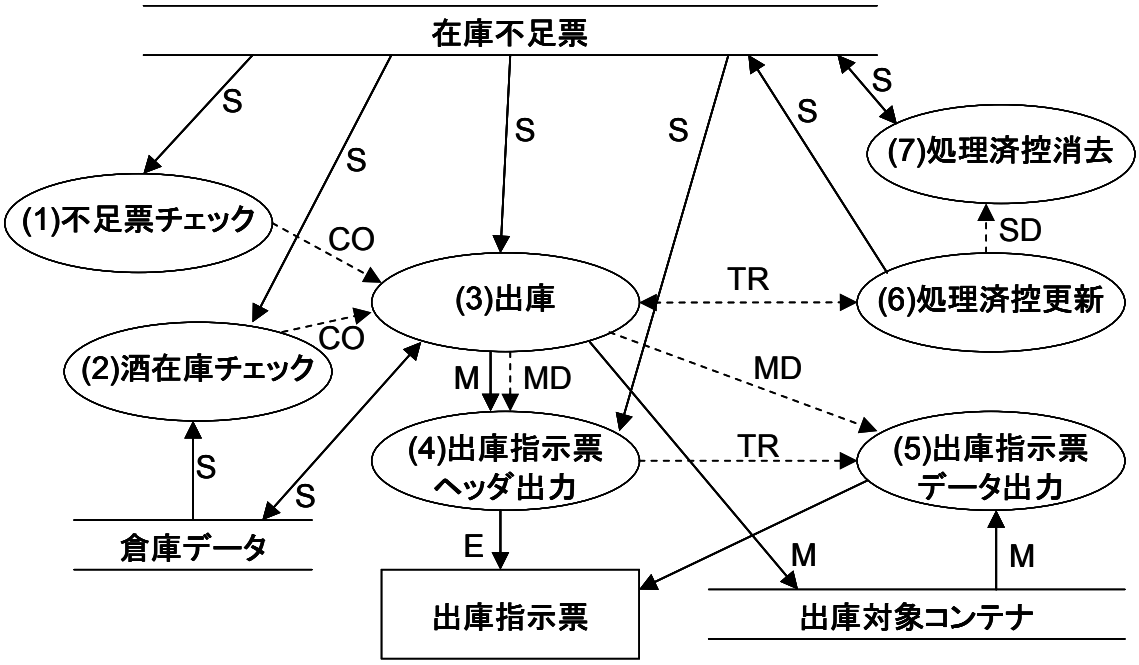

図 26　依存関係設定後の在庫不足解消プログラムの DFD

3.4　プロセス結合ルールの適用(STEP4)

STEP3で設定した依存関係を利用し，依存関係の強いプロセス群を結合し，サービスの候補として抽出する．DFD上のプロセス群をサービスとして抽出するためには，対象となるプロセス群がサービスの条件を満たしていなければならない．2.2節で述べたとおり，サービスの条件には，自己完結（条件1），オープンなインターフェース（条件2），任意の粒度（条件3）の3つがある．提案するサービス抽出ルールは，サービスの条件1とサービスの条件2を充足する形でDFD上のプロセス群を結合するものである．提案手法を任意のDFDレイヤに適用し，任意の粒度のサービスを抽出可能とすることにより，サービスの条件3を充足することができる．

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，サービスは自由に組み合わせて利用することが可能であるため，複数の分割プロセスを結合して1つのサービスと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．

3.4.1　(ルール 1)モジュールデータ依存プロセスの結合

モジュールデータによる依存が存在するプロセスは，結合が必要である．サービスの条件1を実現するためには，サービスを自由に組み合わせて実行することが可能でなければならない．しかし，モジュールデータはシステム内の同一レイヤに属するプロセス間でのみ利用可能な，実装に極めて依存したデータである．P1とP2の間にモジュールデータ依存が存在する場合に，P1とP2を別々のサービスとして分割してしまうと，サービス利用者がP1サービス，P2サービス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．これではP1サービス実行後にP2サービスを実行しなければならなくなり，サービスの条件1に反する．P1とP2を結合することにより，決まった順序で実行が必要なプロセスをサービス内部に隠蔽し，サービスを自由に組み合わせて利用することが可能となる．また，サービスの条件2を実現するためには，システム内部のデータをサービスの入出力から排除する必要がある．P1とP2を別々のサービスとして分割してしまうと，モジュ

ールデータをサービスの入出力データとしなければならなくなる．モジュールデータはシステム内部のデータであるため，サービスの入出力とすることはサービスの条件2に反する．P1とP2を結合することにより，モジュールデータをサービスの入出力から排除することが可能となる．以上より，P1とP2は結合する必要があり，P1+P2が成立する．

図27は，図26で示したDFDにルール1を適用した結果である．(3)のプロセスと(4)のプロセスの間にはモジュールデータ依存が存在する．モジュールデータ依存を有するプロセスは結合する必要があるため，(3)+(4)が成立する．同様に，(3)のプロセスと(5)のプロセスの間にもモジュールデータ依存が存在するため，(3)+(5)が成立する．

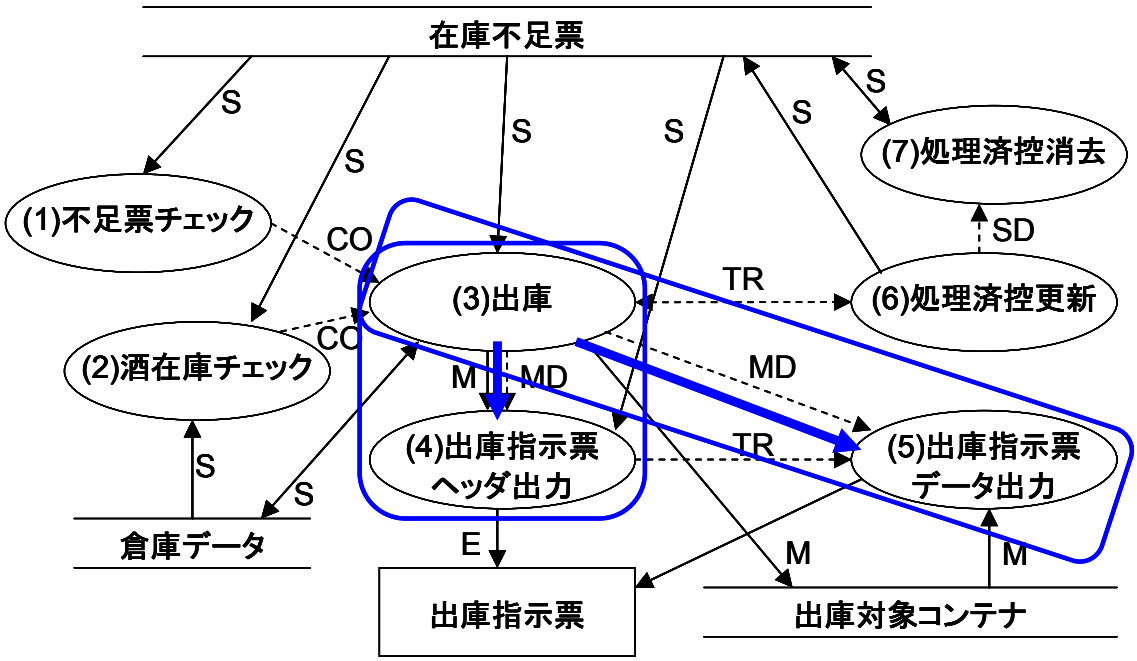

図 27　モジュールデータ依存プロセスの結合

3.4.2　（ルール 2）システムデータ依存プロセスの分割

システムデータによる依存が存在するプロセスは，分割することが可能である．P1とP2の間にシステムデータ依存が存在する場合であっても，P1はデータを任意のタイミングでデータストアに出力することが可能である．システムデータはシ

ステム内の全てのプロセスから参照が可能であるため，P1が適切なデータを出力した後であれば，P2はそのデータを任意のタイミングで取得することが可能である．よって，P1とP2は互いに依存することなく非同期に実行することが可能である．このため，P1とP2を別々のサービスとして分割しても，P1とP2を非同期に実行することが可能であり，サービスの条件1を充足する．また，システムデータはシステム内部のデータであるため，サービスの入出力にするとサービスの条件2を実現することが出来なくなる．しかし，システムデータは異なるレイヤに属するプロセス間で利用が可能である．そのため，P1とP2を別々のサービスとして分割しても，P2はP1が出力したデータを取得することが可能である．よって，P1とP2の入出力データとしてシステムデータを指定することなくデータの受け渡しが可能となり，サービスの条件2を充足する．以上より，P1とP2は分割が可能であり，P1 ｜ P2が成立する．なお，必要に応じて，P1とP2を結合して，1つのサービスとすることも可能である．

図28は，図26で示したDFDにルール2を適用した結果である．(6)のプロセスと(7)のプロセスの間にはシステムデータ依存が存在する．システムデータ依存が存在するプロセスは分割が可能であるため，(6) ｜ (7)が成立する．

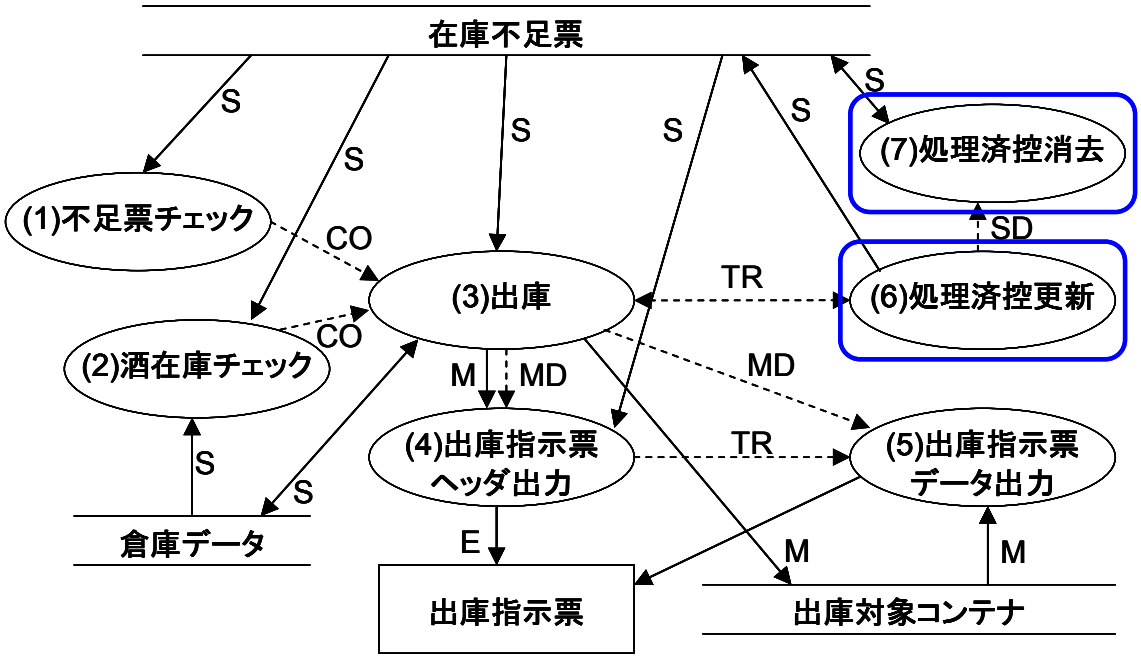

図 28　システムデータ依存プロセスの分割

3.4.3 （ルール 3)処理依存プロセスの結合

処理による依存が存在するプロセスは，結合が必要である．P1とP2の間に処理依存が存在する場合，P1とP2は同一のトランザクション内で実行しなければならない．P1とP2を別々のサービスとして分割した場合，利用者がP1とP2を同一のトランザクション内で実行することが必要となる．このことはサービスを自由に組み合わせて利用することが不可能であることを意味し，サービスの条件1に反する．P1とP2を結合することにより，同一トランザクション内で実行が必要な処理をサービス内に隠蔽し，サービスの自由な組み合わせを実現することが可能になる．また，処理依存が存在するプロセス間のデータフローには，システムデータの入出力によるデータフローのみが存在する．ルール2で述べたとおり，システムデータは入出力データとして指定する必要はない．このため，P1とP2を別々のサービスとして分割しても，サービスの条件2を充足することが可能である．以上より，P1とP2は結合する必要があり，P1+P2が成立する．

図29は，図26で示したDFDにルール3を適用した結果である．(3)のプロセスと(6)のプロセスの間には，処理依存が存在する．処理依存を有するプロセスは結合が必要であるため，(3)+(6)が成立する．同様に，(4)のプロセスと(5)のプロセスにも処理依存が存在するため，(4)+(5)も成立する．

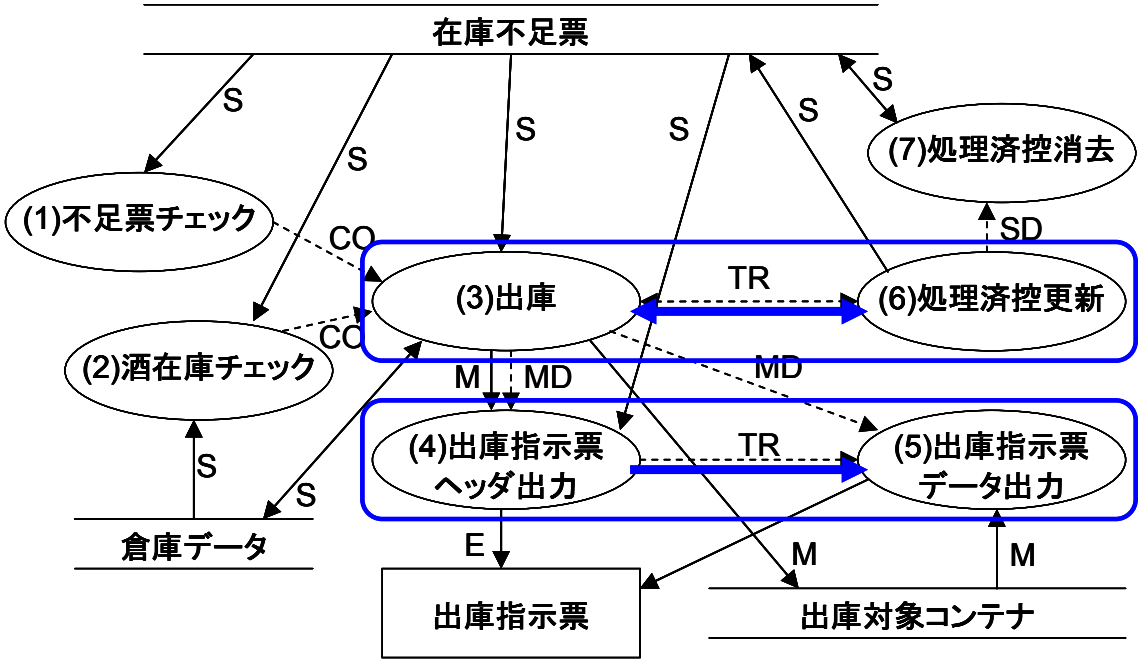

図 29　処理依存プロセスの結合

3.4.4　（ルール 4）制御依存プロセスの分割

制御による依存が存在するプロセスは，分割することが可能である． P1の出力はP2を実行するか否かを判断するための制御フラグであり，P2の入力データにはならない．つまり，P1はP2を実行するための必要条件を満たすわけではなく，実行の有無を判断しているだけである．このため，P2を実行するための条件が整っているのであれば，P1を実行せずにP2を実行することも可能である．よって，P1とP2を分割してサービスとした場合であっても，P1とP2を自由に組み合わせて実行することが可能であり，サービスの条件1を充足する．また，制御依存が存在するプロセス間にはデータフローは存在しない．このため，P1とP2を別々のサービスとして分割した場合であっても，サービスの条件2を充足する．以上より，P1とP2は分割が可能であり，P1 ｜ P2が成立する．

図30は，図26で示したDFDにルール4を適用した結果である．(1)のプロセスと(3)のプロセスの間には，制御依存が存在する．制御依存を有するプロセスは分割が可能であるため，(1) ｜ (3)が成立する．同様に，(2) ｜ (3)も成立する．

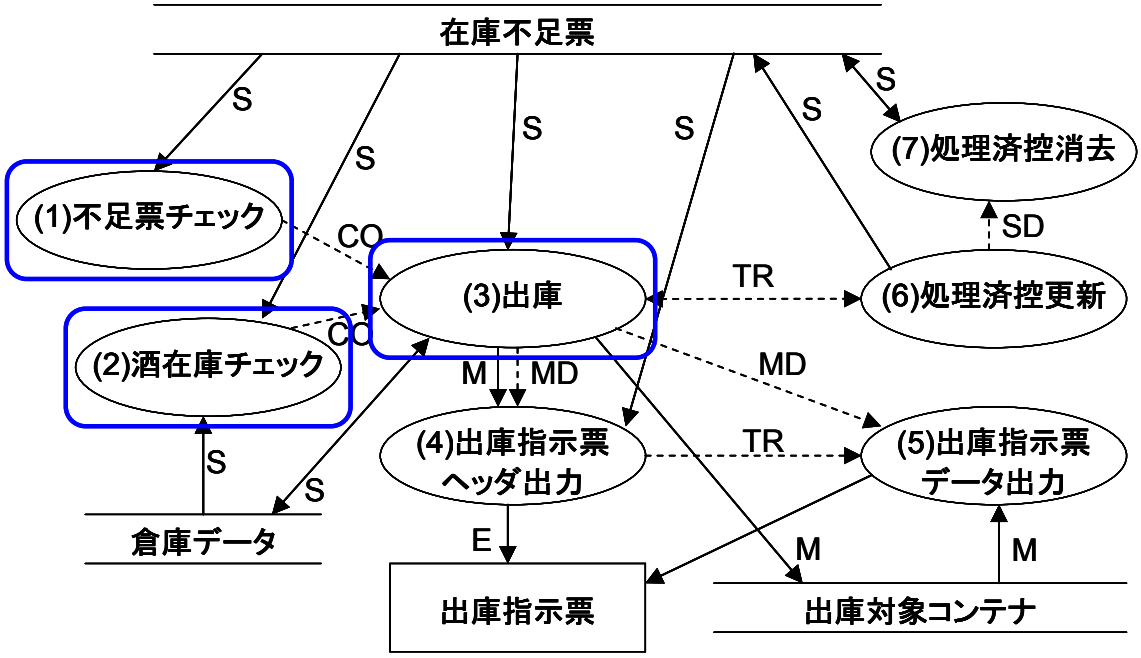

図 30　制御依存プロセスの分割

3.4.5　（ルール 5）結合プロセスの合流

結合プロセスの合流とは，複数の結合プロセスが特定のプロセスで合流している場合を指す．結合プロセスの合流が発生している場合，これらのプロセスは全て結合する必要がある．結合プロセスP1+P2および結合プロセスP3+P2が存在する場合，P2を実行するためにはP1およびP3が必要となる．P1+P2とP3+P2を別々のサービスとして分割した場合，P2を実行するためのデータやトランザクションの同期を満たすことが不可能になる．よって，P1，P2，P3を結合する必要があり，P1+P2+P3が成立する．本ルールの適用が必要となるのは，モジュールデータ依存が存在するプロセスが合流している場合と，処理依存が存在するプロセスが合流している場合である．

図31は，図26で示したDFDにルール5を適用した結果である．(3)のプロセスと(5)のプロセスはモジュールデータ依存による結合プロセスであり，矢印は(3)から(5)に向いている．(4)のプロセスと(5)のプロセスは処理依存による結合プロセスであり，矢印は(4)から(5)に向いている．(3)のプロセスと(4)のプロセスは(5)のプロセスで合流しているため，(3)+(4)+(5)が成立する．

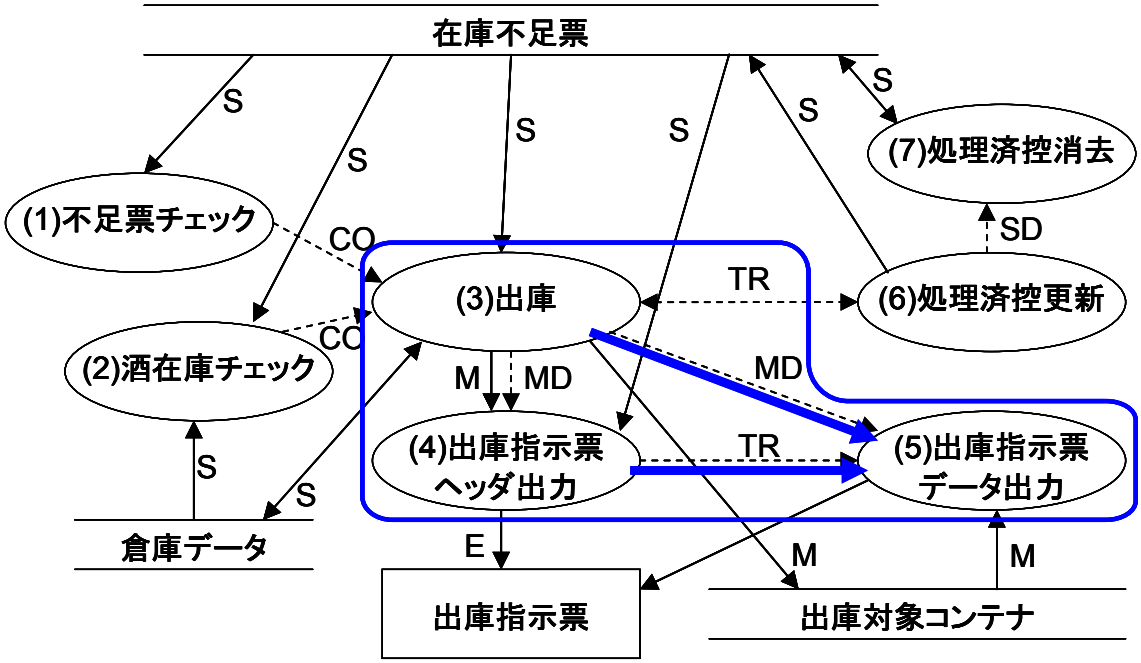

図 31　結合プロセスの合流

3.4.6　（ルール 6）結合プロセスの連鎖

結合プロセスの連鎖とは，あるプロセスを経由して複数の結合プロセスが連続している場合を指す．結合プロセスの連鎖が発生している場合，これらのプロセスは全て結合する必要がある．結合プロセスP1+P2および結合プロセスP2+P3が存在する場合，P3を実行するためにはP2が必要であり，P2を実行するためにはP1が必要である．よって，P3を実行するためにはP2だけでなくP1も必要となる．以上より，P1，P2，P3を結合することが必要となり，P1+P2+P3が成立する．本ルールの適用が必要となるのは，モジュールデータ依存が存在するプロセスが連鎖している場合と，処理依存が存在するプロセスが連鎖している場合である．

図32は，図26で示したDFDにルール6を適用した結果である．(3)のプロセスと(4)のプロセスは結合プロセスであり，矢印は(3)から(4)に向いている．(3)のプロセスと(6)のプロセスも結合プロセスで，矢印は(6)から(3)に向いている．(6)のプロセスと(4)のプロセスは(3)のプロセスを経由して連鎖しているため，(6)+(3)+(4)が成立する．

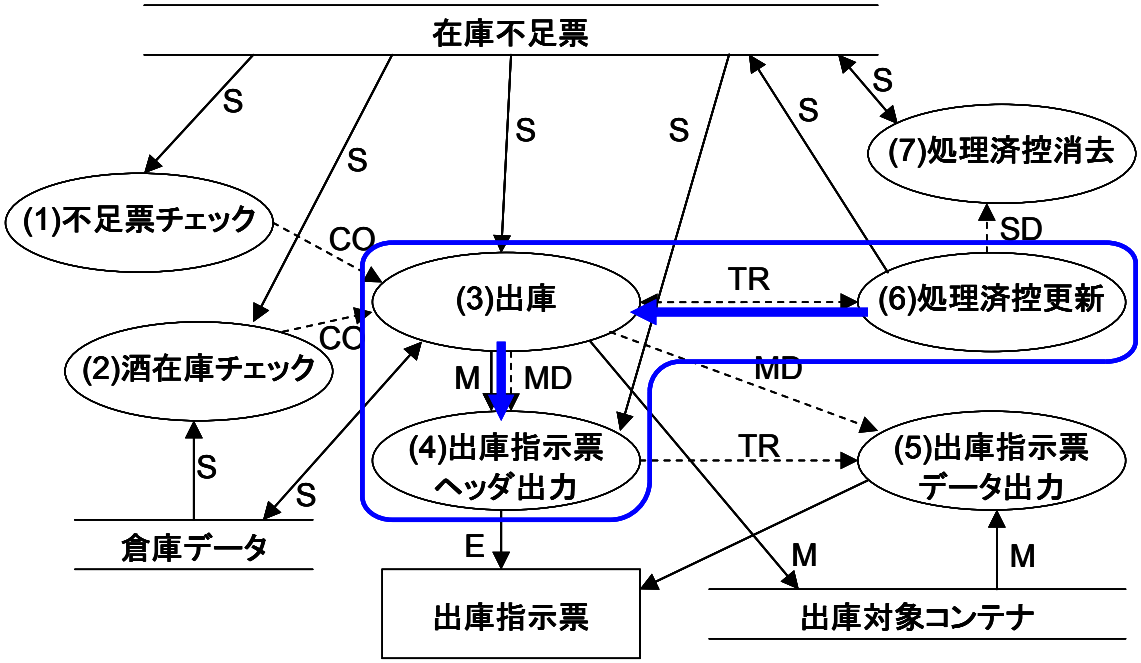

図 32　結合プロセスの連鎖

図33は，図26で示したDFDに提案手法を全て適用した結果である．提案手法を適用することにより，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，処理済控消去サービスという4つのサービスをDFDから抽出することが出来た．

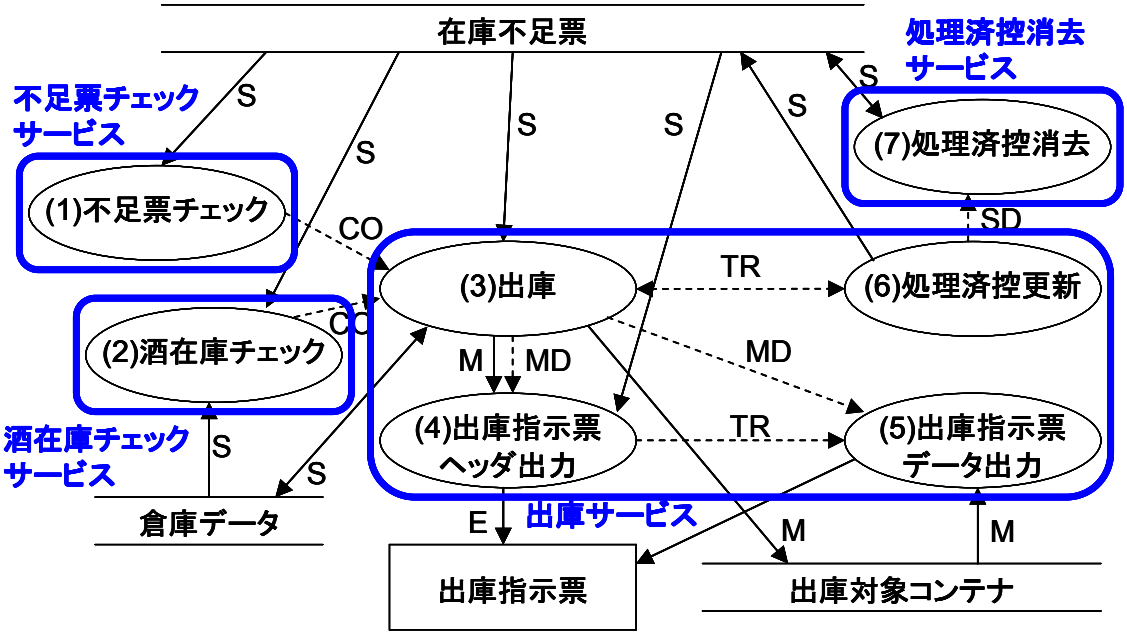

図 33　プロセス結合後の在庫不足解消プログラムの DFD

4. ケーススタディ

4.1 酒在庫管理プログラムの概要

提案手法の有効性を確認するために，酒在庫管理システム[16]の2つの実装に提案手法を適用し，サービス候補の抽出を行った．両実装共にC言語にて作成されており，1つ目の実装は約800行，2つ目の実装は約430行であり，1つ目を酒在庫管理プログラムA，2つ目を酒在庫管理プログラムBと呼ぶことにする．これらのソースコードでは，酒在庫管理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．

積荷票処理は，酒を倉庫に入庫する処理と，入庫後に在庫待ち状態になっている酒を出庫する処理で，2.5.1節の処理と2.5.2節の処理を組み合わせたものである．図34に本処理の概要を示す．受付係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，在庫不足票を酒銘柄で検索し，特定した酒銘柄の在庫が十分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し，出庫依頼票を出力する．

図 34　積荷票処理の概要

出庫依頼票処理は，酒を倉庫から出庫する処理で，2.5.3節の処理と2.5.4節の処理を組み合わせたものである．図35に本処理の概要を示す．受付係が出庫依頼票を入力すると，記載されている酒銘柄の出庫処理を行い，出庫した分だけ倉庫データから該当する酒銘柄の在庫を減らす．出庫処理を行った酒銘柄については，出庫指示票を出力する．出庫依頼票で要求された酒銘柄の在庫が倉庫に十分にな

い場合には，在庫不足票に出庫できない銘柄のデータを追加する．在庫不足票に追加した酒銘柄については，不足通知票を出力する．

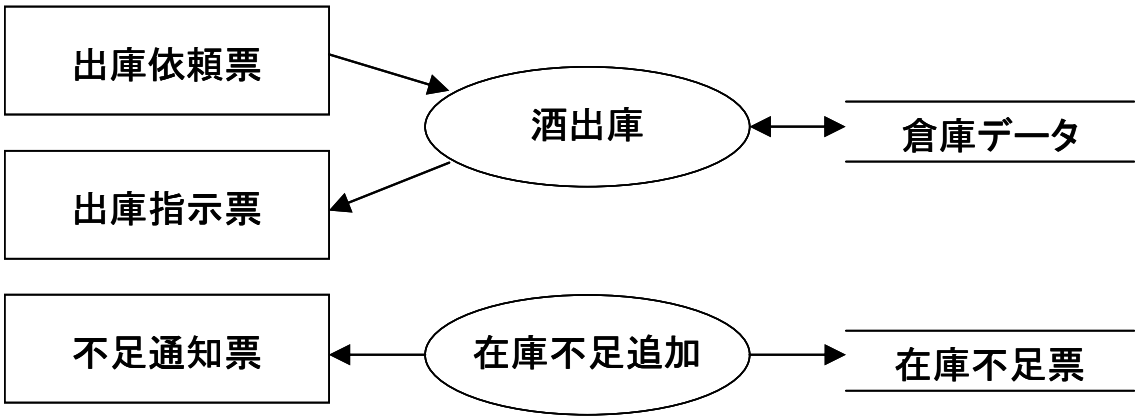

図 35　出庫依頼票処理の概要

4.2　酒在庫管理プログラム A への適用

4.2.1　酒在庫管理システム全体(プログラム A)への適用

まず，酒在庫管理プログラムAの実装から取得したDFDの上位レイヤに対して提案手法を適用し，酒屋システム全体からサービス抽出を行った．図36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丸枠で囲む）．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，本実装ではシステムの主な処理が，5つの依存関係の弱いプロセスとして分割されていたことが分かる．

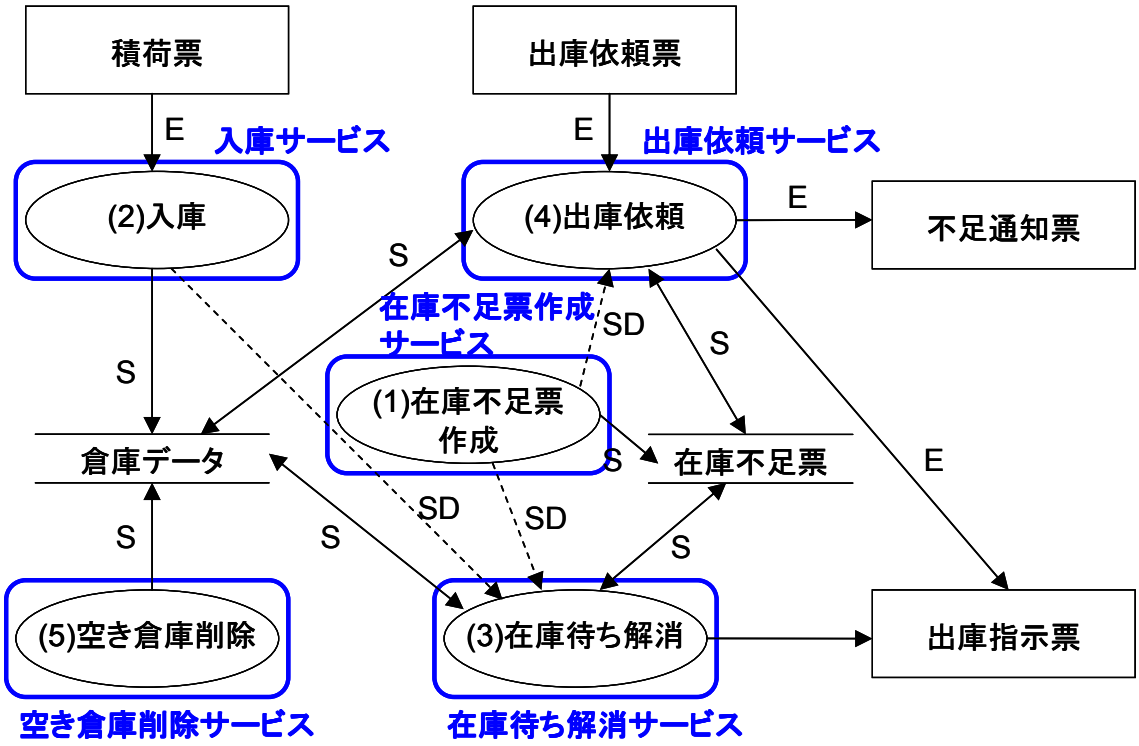

図 36　酒在庫管理プログラム全体（プログラム A）への提案手法適用結果

4.2.2　入庫プロセス(プログラム A)への適用

図36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37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．

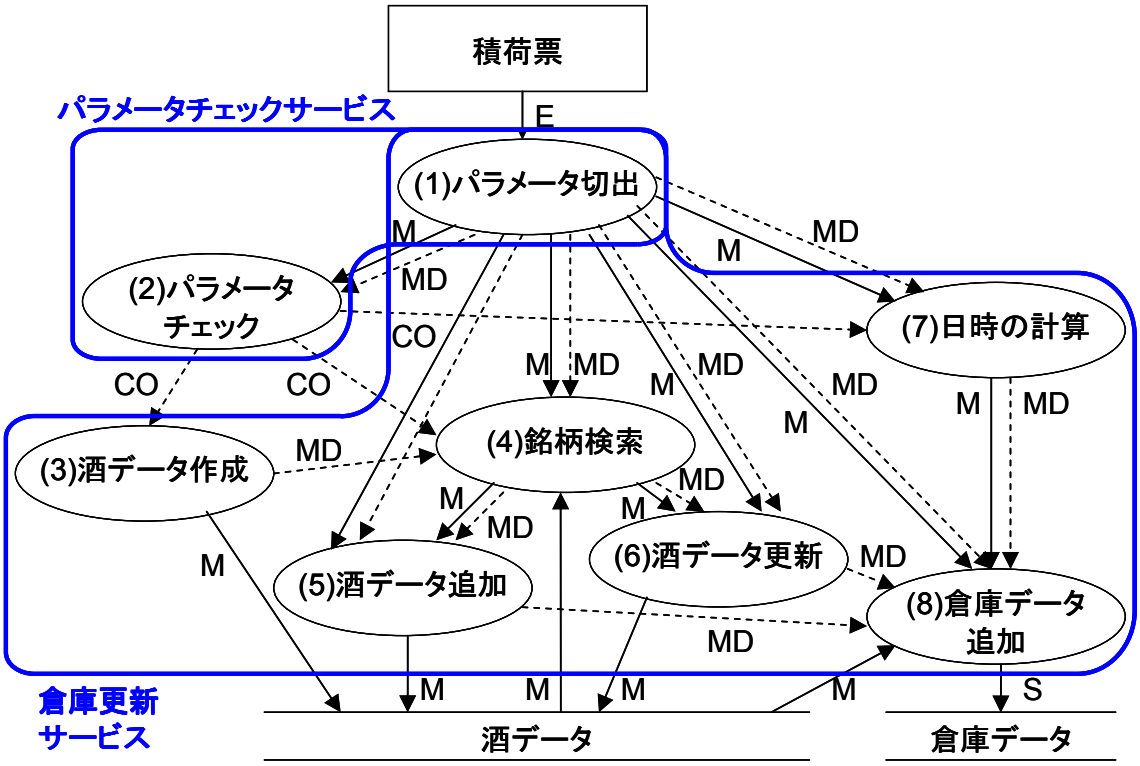

図 37　入庫プロセス（プログラム A）への提案手法適用結果

4.2.3　在庫待ち解消プロセス(プログラム A)への適用

同様に，図36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38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3），（ルール6）を適用すると，(1)+(2)+(3)+(4)+(5)+(6)+(7)が成立する．（ルール2）より(7)と(8)は分割可能である．結果的に2つのサービス (1)+(2)+(3)+(4)+(5)+(6)+(7) ｜ (8)が抽出された．

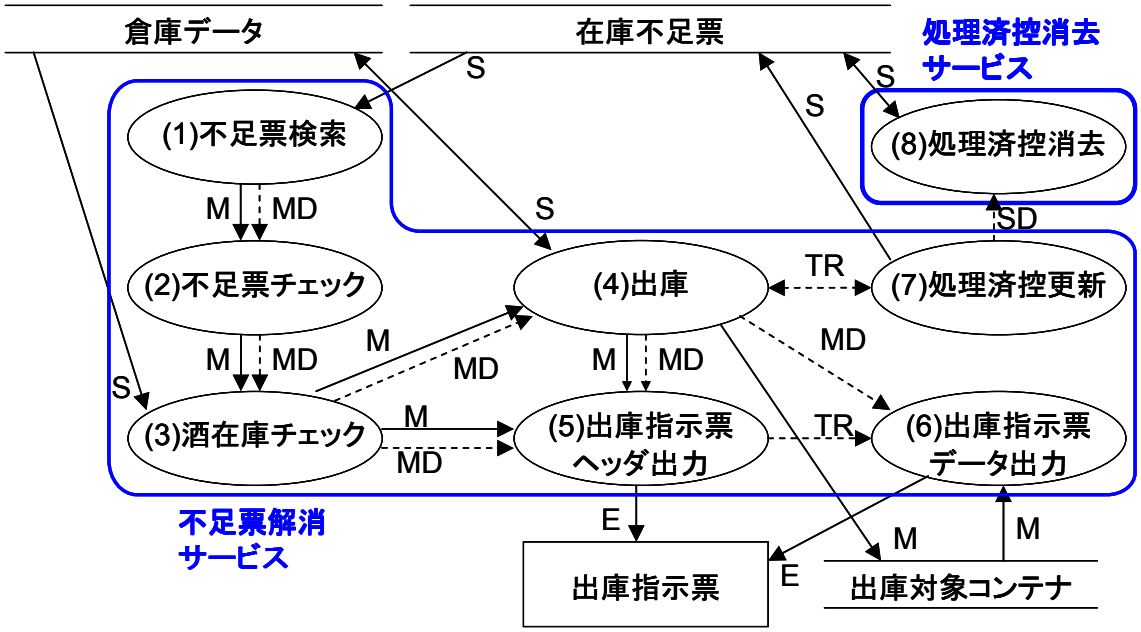

図 38　在庫待ち解消プロセス（プログラム A）への提案手法適用結果

4.2.4　出庫依頼プロセス(プログラム A)への適用

同様に，図36の(4)出庫依頼プロセスの下位レイヤからサービスを抽出した．図39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3），（ルール4），（ルール5），（ルール6）を適用すると，(1)+(2)｜(1)+(3)｜(1)+(4)｜(1)+(5)+(6)+(7)｜(1)+(8)が抽出された．

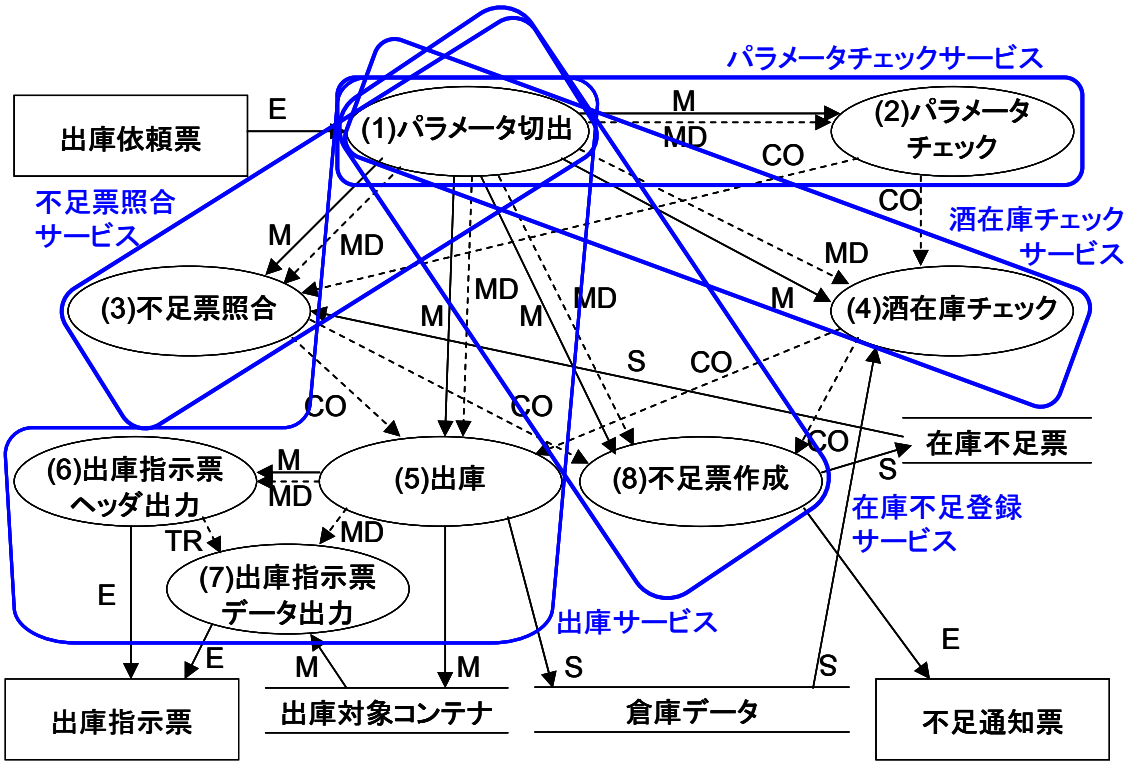

図 39　出庫依頼プロセス（プログラム A）への提案手法適用結果

4.3　酒在庫管理プログラム B への適用

4.3.1　酒在庫管理システム全体（プログラム B）への適用

4.2節と同様に，酒在庫管理プログラムBの実装から取得したDFDの上位レイヤに提案手法を適用した．図40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2），（ルール4），（ルール6）を適用すると，(1)+(2)+(3)+(4)｜(1)+(2)+(3)+(5)｜(1)+(2)+(3)+(6)｜(1)+(2)+(3)+(7)｜(8) の5つのサービスが抽出された．

酒在庫管理プログラムAとの相違点は2点ある．まず，酒在庫管理プログラムでいうところの入庫プロセスに該当する部分が倉庫作成サービス，倉庫追加サービス，倉庫更新サービスの3つに分割されたことである．もう1点は，在庫待ち解消サービスの入力として，外部データである積荷票が必要となることである．

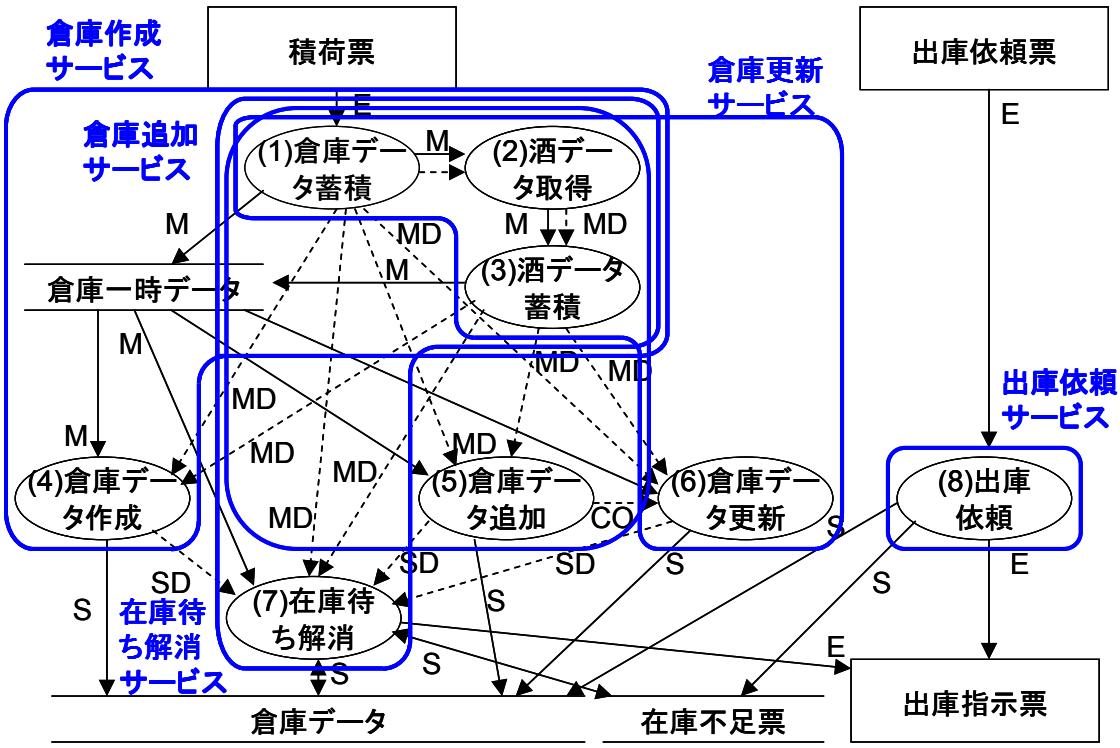

図 40　酒在庫管理プログラム全体（プログラム B）への提案手法適用結果

4.3.2　在庫待ち解消プロセス(プログラム B)への適用

図40の(1)から(6)までのプロセスは，酒在庫管理プログラムAでいうところの入庫プロセスに該当する．酒在庫管理プログラムBでは，入庫プロセスに該当する処理が下位レイヤの処理として実装されていない．よって，(7)の在庫待ち解消プロセスに対して提案手法を適用する．図41に適用結果を示す．(1)から(5)までの処理に対して（ルール1），（ルール3），（ルール5）を適用すると，(1)+(2)+(3)+(4)+(5)が抽出された．(4)と(5)の間で双方向矢印の処理依存が成立する理由は，(4)の出庫指示票出力プロセスは在庫不足票のデータを元に処理を行うため，(5)の不足表削除プロセスで在庫不足票を更新してしまうと，前提となるデータがなくなってしまうためである．よって，(4)のプロセスと(5)のプロセスは同一トランザクション内で実施することが必要である．

酒在庫管理プログラムAとの相違点は，酒在庫管理プログラムAの処理済控え消去サービスが酒在庫管理プログラムBの在庫待ち解消サービスに含まれていることである．具体的には，(5)の不足票削除プロセスにて，(4)の出庫指示票出力プロセスと同期を取って処理済控え消去サービスに該当する処理を実行している．

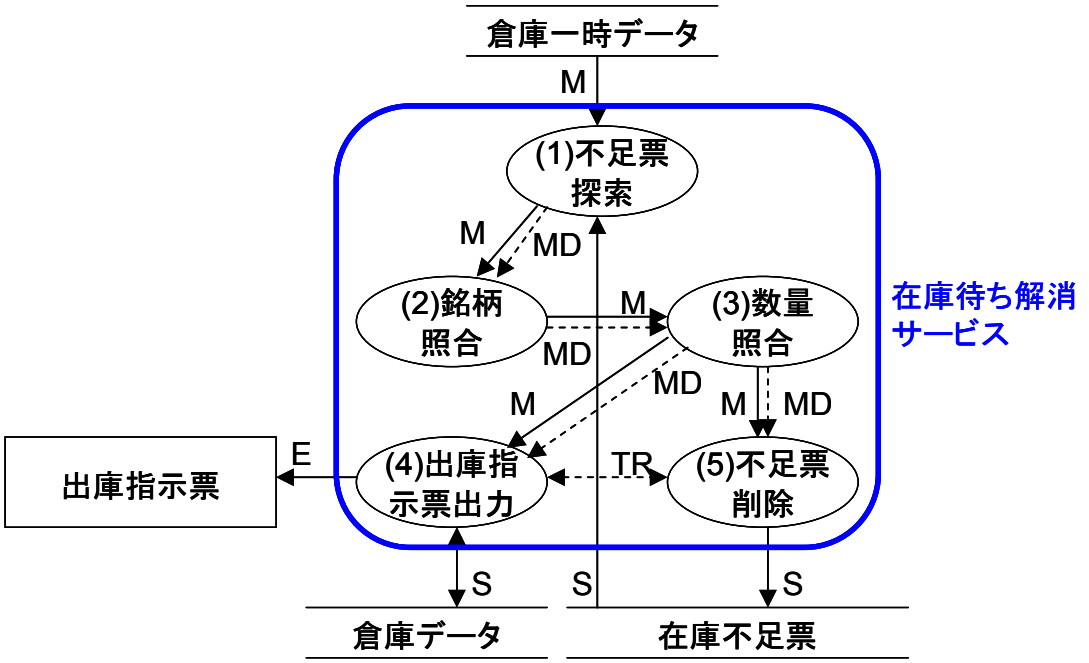

図 41　在庫待ち解消プロセス（プログラム B）への提案手法適用結果

4.3.3　出庫依頼プロセス(プログラム B)への適用

図40の(8)出荷依頼プロセスに対して提案手法を適用する．図41に適用結果を示す．(1)から(5)までの処理に対して（ルール1），（ルール4）を適用すると，(1)+(2) ｜ (1)+(3) ｜ (1)+(4)が抽出された．

酒在庫管理プログラムAとの相違点は2点ある．まず，出庫依頼書のパラメータチェックを行っていないためにパラメータチェックサービスが存在しない．また，在庫不足票のチェックを行っていないために不足票照合サービスも存在しない．

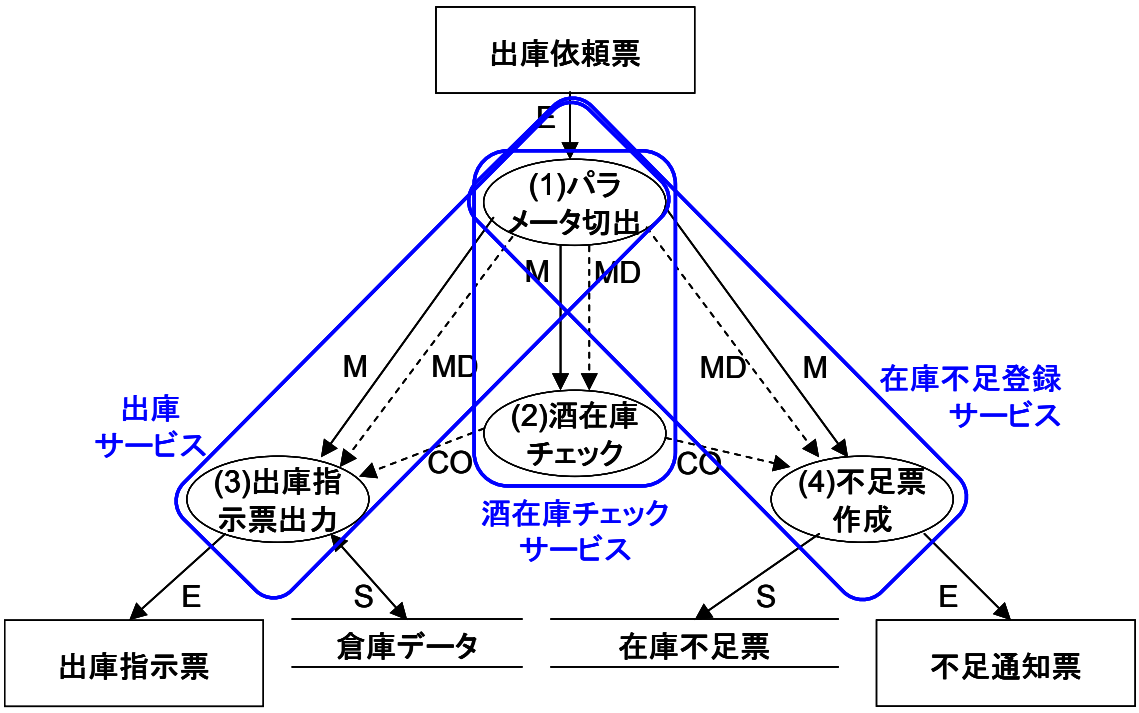

図 42　出庫依頼プロセス（プログラム B）への提案手法適用結果

5. 考察

5.1 提案手法の特徴

5.1.1 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．例として，酒在庫管理プログラムAから抽出可能なサービスの候補を表1に示す．

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一のサービス内に隠蔽され，その実行順序をサービス利用者が意識する必要はない．抽出したサービス候補は，単独で実行することも，繰り返し実行することも，他のサービスと組み合わせて実行することも可能である．例えば，在庫不足解消サービスは入庫サービス実行前に単独で実行することも可能であるし，入庫サービスと組み合わせて実行することも可能である．このようにして，抽出したサービス候補は他のサービス候補との依存を気にすることなく実行することが可能である．よって，サービスの条件1を満たす．

また，提案手法により抽出したサービス候補は，外部データ（積荷票，出庫指示票）を入力データとするか，入力データが不要である．システムの実装に強く依存するモジュールデータ，外部データともにサービス利用者は意識する必要はない．但し，入力データ，出力データともにサービスの属するレイヤによって異なる場合がある．例えば，酒在庫管理サービスは積荷票と出庫指示票の両方を入力データとし，入庫サービスは積荷票のみを入力データとしている．入庫サービスを単独で外部に公開するのであれば，積荷票のみを抽出する変換サービスをラッピングすることで対応が可能であろう．従って条件2を満たす．なお，出力データについては，サービスを実行した結果出力される外部データだけでなく，サービスを実行した結果（正常終了または異常終了）を利用者に通知する必要があるだろう．例えば，入庫処理は積荷票に記載された酒銘柄を登録する処理を行うのみで外部データの出力は行わないが，データの登録が正常終了したのか異常終了したのかをサービス利用者に通知する必要がある．

表 1　酒在庫管理プログラム A から抽出可能なサービス候補

レイヤ	サービス名	入力	出力
1	酒在庫管理	積荷票 出庫依頼票	処理結果（成功/失敗） 出庫指示票 不足通知票
2	入庫	積荷票	処理結果（成功/失敗）
3	パラメータチェック	積荷票	処理結果（真/偽/失敗）
3	倉庫更新	積荷票	処理結果（成功/失敗）
2	在庫待ち解消	なし	処理結果（成功/失敗） 出庫指示票
3	不足票解消	なし	処理結果（成功/失敗） 出庫指示票
3	処理済控え消去	なし	処理結果（成功/失敗）
2	出庫依頼	出庫依頼票	処理結果（成功/失敗） 出庫指示票 不足通知票
3	パラメータチェック	出庫依頼票	処理結果（真/偽/失敗）
3	不足票照合	出庫依頼票	処理結果（真/偽/失敗）
3	酒在庫チェック	出庫依頼票	処理結果（真/偽/失敗）
3	出庫	出庫依頼票	処理結果（成功/失敗） 出庫指示票
3	在庫不足登録	出庫依頼票	処理結果（成功/失敗） 不足通知票
2	在庫不足票作成	なし	処理結果（成功/失敗）
2	空き倉庫削除	なし	処理結果（成功/失敗）

5.1.2　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，任意のDFDレイヤを選択することができる．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．表1に示したサービス候補を例にすると，最上位レイヤ（レイヤ1）のサービスは酒在庫管理サービスであり，入庫サービスや在庫待ち解消サービスなどをまとめた高機能なサービス候補である．積荷票または出庫依頼票を入力データとするだけで，入庫サービス，出庫依頼サービスなどに該当する処理を内部で判別して実行することが可能である．一方で，最下位レイヤ（レイヤ3）のサービス候補としてパラメータチェックサービスが抽出可能である．このサービスは積荷票のデータが正しいかどうかチェックする機能のみを有しており，入庫前に積荷票のデータをあらかじめチェックする用途などで利用することが可能である．このようにして，抽出するサービスの粒度を自由に決定することが可能である．従って，DFDの適度な詳細度を決定して提案手法を適用することにより，サービスの条件3を充足可能である．

5.1.3　レガシーソフトウェアの実態を考慮したサービス抽出

提案手法を用いることで，レガシーソフトウェアの実態を考慮したサービスの抽出が可能となる．同一の仕様からプログラムを作成した場合であっても，実装が異なることは十分に考えられる．この場合，提案手法を適用して抽出可能となるサービス候補は別のものとなる．ケーススタディにおいても，酒在庫管理プログラムAと酒在庫管理プログラムBは同一の仕様を元に作成したが，実装が異なるため，抽出したサービス候補も異なるものであった．よって，仕様を元にレガシーソフトウェアからサービスを抽出する手法はシステムの実態にそぐわない結果になることが考えられる．酒在庫管理プログラムAの実装に合わせたサービス候補を考えていた場合，酒在庫管理プログラムBは大幅な修正が必要となるであろう．SOAに基づくシステムを新規開発する際の手法では業務フローを元にサービスを設計する．ここでベースとなる業務フローはシステムの仕様よりもさらに上流の行程である．このため，業務フローを元にサービスを設計する手法を用いてレガシーソフトウェアにSOAを適用した場合，システムの実装からかけ離れたサービス候補が出来上がってしまうことが十分に考えられる．しかしながら，提

案手法ではソースコードを入力としてサービスを抽出しているため，システムの実装を正確に反映することができる．

5.1.4 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

5.2.1 Ward の変換図

Wardの変換図[11]はDFDを拡張したもので，プロセス間の依存関係を破線矢印により表現できる．しかし，Wardの変換図はもともとプロセスの依存解析を行うことを目的としていないため，依存関係の種類を明示的に指定できない．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発生原因毎に4種類に分類して，DFDを依存解析ツールとして使えるよう拡張し，サービス抽出に利用した．

5.2.2 モジュールの独立性

プログラムモジュールの独立性を測る尺度として，**モジュール凝集度**と**モジュール結合度**が提唱されている[3][15]．モジュール凝集度とは，単一モジュール内に含まれる処理がどの程度機能的な関連性を有しているかを測る尺度である．モジュール結合度とは，2つのモジュール間のつながりがどの程度強いかを測る尺度である．モジュール凝集度が高く，モジュール結合度が弱いプログラムのほうが，プログラムモジュールの独立性が確保されており，保守性の高いプログラムであるとされている．提案手法では，依存関係の強い処理を結合することにより，DFDからサービスを抽出する．モジュール凝集度とモジュール結合度は，プログラムモジュール間の依存関係を測る従来手法であるので，提案手法との関連性を考察する．

5.2.3 モジュール凝集度

モジュール凝集度は凝集度の弱い順に，暗号的凝集，論理的凝集，時間的凝集，手順的凝集，通信的凝集，逐次的凝集，機能的凝集の7段階に分類されている[2][3][15]．

暗号的凝集とは，モジュール内の処理間に，機能的な関連性が全くない場合が該当する．各処理がプロセスに該当すると考えると，プロセス間に依存関係が認

められない状態である．暗号的凝集が発生しているプロセスを図43に示す．プロセスAとプロセスBには何ら依存関係が存在しないため，分割が可能である．

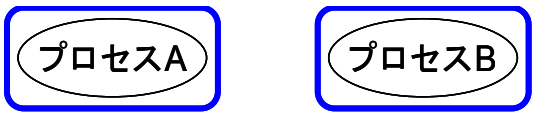

図 43　暗号的凝集が発生しているプロセス

論理的凝集とは，条件判定を行い，2つの処理からその条件に対応するものを選択して実行する場合が該当する．時間的凝集は論理的凝集のサブセットであり，「プログラム起動直後の場合」や「プログラム終了直前の場合」というような条件判定に応じて処理を選択，実行する．論理的凝集（時間的凝集）が発生しているプロセスを図44に示す．条件判定プロセスにより，プロセスA，プロセスBのいずれを実行するかを決定する．いずれの場合も，プロセスAとプロセスBの間には依存関係が存在しないため，分割が可能である．

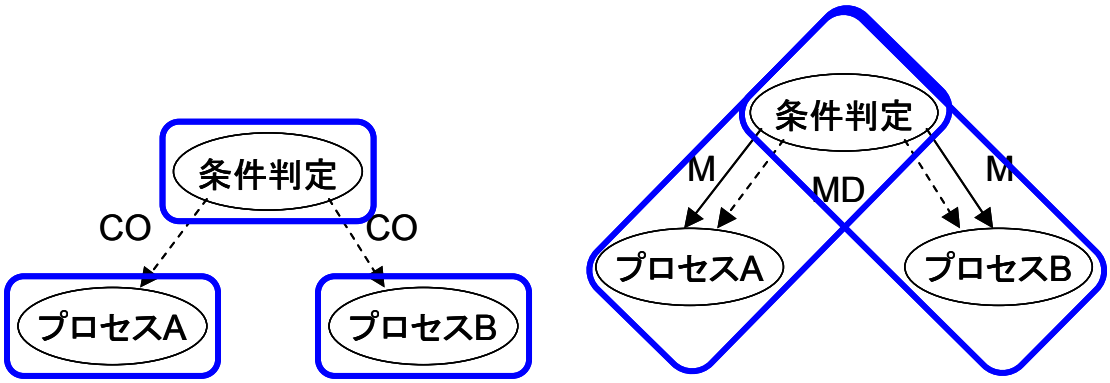

図 44　論理的凝集，時間的凝集が発生しているプロセス

手順的凝集とは，同一のイテレーション内や同一の条件判定内，同一のトランザクション内で2つの処理が実行されている場合が該当する．手順的凝集が発生しているプロセスを図45に示す．プロセスAとプロセスBを同一のトランザクショ

ン内で実行する必要がある場合，処理依存が存在する．このため，プロセスAとプロセスBは結合する必要がある．

図 45　手順的凝集が発生しているプロセス

通信的凝集とは，2つの処理の入力データが同一であるか，出力データが同一である場合が該当する．通信的凝集が存在するプロセスを図46，図47に示す．図46はデータがモジュールデータである場合である．同一のモジュールデータを入力としている場合はプロセスAとプロセスBの間には依存関係が存在しないため，プロセスAとプロセスBは分割が可能である．同一のモジュールデータを出力する場合は，結合プロセスの合流が発生するため，プロセスAとプロセスBは結合する必要がある．図47はデータがシステムデータである場合である．同一のシステムデータを入力としている場合であっても，同一のシステムデータを出力する場合であっても，プロセスAとプロセスBの間には依存関係が存在しないため，分割することが可能である．

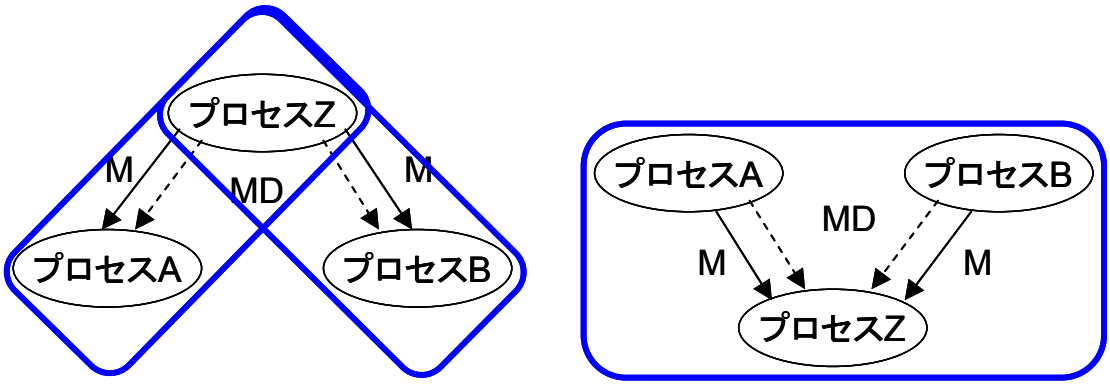

図 46　モジュールデータによる通信的凝集が発生しているプロセス

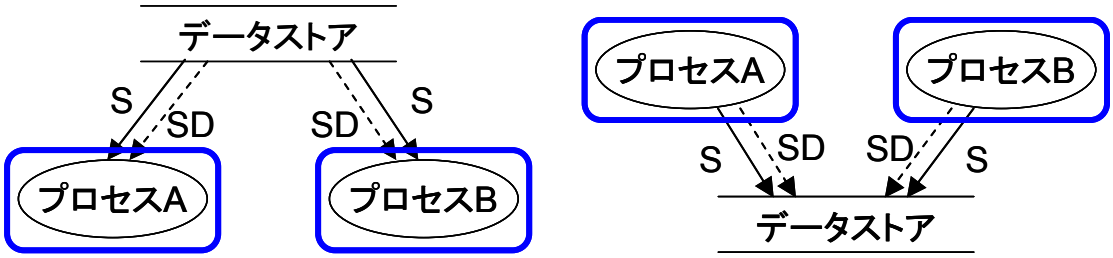

図 47　システムデータによる通信的凝集が発生しているプロセス

逐次的凝集とは，一方の処理の出力データが他方の処理の入力データである場合が該当する．逐次的凝集が発生しているプロセスを図48に示す．プロセスAとプロセスBの間に存在するデータフローがモジュールデータであれば，結合する必要がある．プロセスAとプロセスBの間に存在するデータフローがシステムデータであれば，分割が可能である．

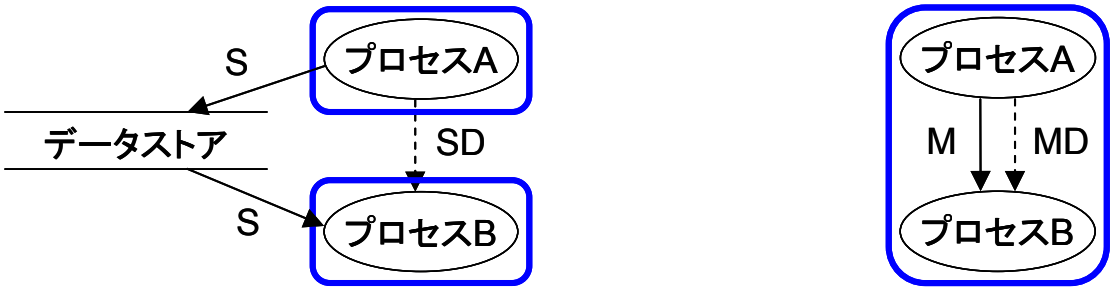

図 48　逐次的凝集が発生しているプロセス

機能的凝集とは，モジュール内の処理が単一機能のみを実現する場合が該当する．具体的には暗号的凝集，論理的凝集，時間的凝集，手順的凝集，通信的凝集，逐次的凝集のいずれにも該当しない場合に，機能的凝集が該当する．つまり，サービス内の処理が1プロセスで構成されている場合が該当する．このような場合は既に粒度が最も細かくなっているため，分割は不可能である．

モジュール凝集度と，プロセスの結合要否の対応を表1に示す．同一のモジュール凝集度で結合が必要なプロセスと分割可能なプロセスが存在する場合には，結合が必要なプロセスを優先している．提案手法を用いることにより，2つのプロセスの間で暗号的凝集，論理的凝集，時間的凝集が発生している場合は分割し

て別サービスにすることが可能である．このことは，提案手法により抽出されるサービス候補は，手順的凝集以上の凝集度の高さを実現することが可能であることを意味する．

表 2　モジュール凝集度とプロセス結合要否の対応

凝集度	プロセス結合判定
暗号的凝集	分割可能
論理的凝集	分割可能
時間的凝集	分割可能
手順的凝集	要結合
通信的凝集	要結合
逐次的凝集	要結合
機能的凝集	1プロセス構成のため，判定不可

5.2.4　モジュール結合度

モジュール結合度はモジュール間の通信方法により，内容結合，共通結合，制御結合，スタンプ結合，データ結合，非直接結合の6段階に分類されている[3][5][15]．内容結合が最も結合度が強く，非直接結合が最も結合度が弱い．

内容結合とは，モジュールAがモジュールB内のデータを更新したり，モジュールBの処理を実行したりする場合が該当する．この結合度は直接他のモジュールのデータ部のアドレスにアクセスするなど特殊な操作が必要となるため，本論文では対象外とする．

共通結合とは，大域データを用いてモジュール間で通信を行う場合が該当する．つまり，モジュールAがグローバル変数に出力したデータをモジュールBが入力データとするような場合である．モジュールをDFDのプロセスに対応させると，プロセスAとプロセスBの間にシステムデータによるデータフローが発生している場合が該当する．共通結合が発生しているプロセスを図49に示す．プロセスAとプロセスBの間に存在する依存関係はシステムデータ依存であるため，分割することが可能である．

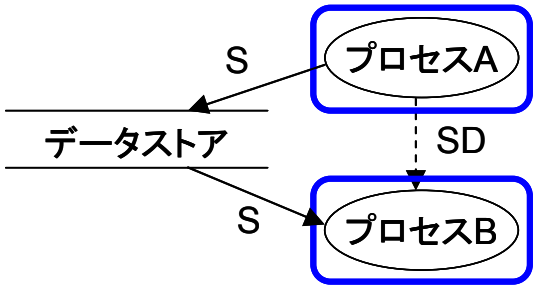

図 49　共通結合が発生しているプロセス

制御結合，スタンプ結合，データ結合は，いずれもモジュールの入力インターフェースにデータを指定することでモジュール間の通信を行う場合が該当する．制御結合は入力データによりモジュールの振る舞いが制御される場合であり，スタンプ結合，データ結合は入力データが単純なデータである場合である．スタンプ結合とデータ結合の違いは，スタンプ結合はインターフェースに指定された入力データの一部のみをモジュール内で使用するのに対し，データ結合は必要なデータのみが入力データとしてインターフェースに指定されることである．スタンプ結合の例としては構造体全体を入力データとして渡すが，実際にモジュール内で利用されるのは構造体の一部の要素である場合が挙げられる．データ結合の例としては，必要な変数だけをモジュールの入力データとして指定する場合が挙げられる．制御結合，スタンプ結合，データ結合が発生しているプロセスを図50に示す．いずれも，プロセス間でモジュールデータによるデータフローが発生している場合が該当する．プロセスAとプロセスBの間にはモジュールデータ依存が存在するため，結合が必要である．

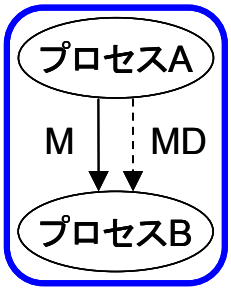

図 50　制御結合，スタンプ結合，データ結合が発生しているプロセス

非直接結合とは，モジュールAとモジュールBの間に通信が存在しない場合が該当する．非直接結合が発生しているプロセスを図51に示す．非直接結合に該当するのはプロセス間にデータフローが発生しない場合である．具体的には，プロセス間に依存関係が存在しない場合，プロセス間に制御依存が存在する場合，プロセス間に処理依存が存在する場合の3パターンが考えられる．プロセスAとプロセスBの間に依存関係が存在しない場合，プロセスAとプロセスBの間に存在する依存関係が制御依存である場合は分割が可能である．プロセスAとプロセスBの間に処理依存が存在する場合は結合する必要がある．

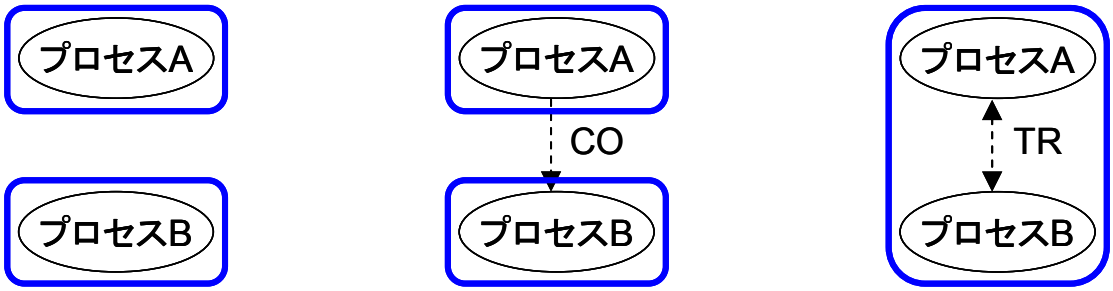

図 51　非直接結合が発生しているプロセス

モジュール結合度と，プロセス結合要否の対応を表3に示す．同一のモジュール結合度で結合が必要なパターンと分割可能なパターンの両方が存在する場合には，結合が必要なパターンを優先している．提案手法を適用した結果，モジュール結合度の強いプロセスは分割可能，モジュール結合度が弱いプロセスは結合が必要となった．よって，提案手法により抽出されるサービス候補は，モジュール結合度とは関連性が見られない．SOAでは，サービスを提供するシステムの実装に依存するデータはサービスのインターフェースから排除して，外部から見えないようにしている．これに対し，モジュール結合度では，システムの実装に依存するデータであってもモジュールのインターフェースとして明示的に指定するほうが良いとしている．このように，SOAの考え方とモジュール結合度の考え方には相容れない部分があり，提案手法はモジュール結合度の低いサービス候補を抽出することが出来ないものと思われる．

表 3　モジュール結合度とプロセス結合要否の対応

モジュール結合度	プロセス結合判定
内容結合	該当なしのため，判定不可
共通結合	分割可能
制御結合	要結合
スタンプ結合	要結合
データ結合	要結合
非直接結合	要結合

6. まとめ

本論文では，サービス指向アーキテクチャの構成要素であるサービスの条件として，自己完結，オープンなインターフェース，疎結合の3つを明確にした．その上で，レガシーソフトウェアのソースコードをDFDに変換し，データ依存の解析を行うことで，サービスの3条件を満たす形でレガシーソフトウェアからサービスの候補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複数の実装からサービス候補を抽出し，抽出したサービスの候補がサービスの3条件を満たしていることを確認した．

今後の研究では，レガシーソフトウェアに提案手法を適用することを考えている．本研究では酒在庫管理システムの複数の実装に提案手法を適用したが，大規模なソフトウェアには適用していない．また，レガシーソフトウェアには長年の保守により改修が難しいという特徴もある．今後は実際に業務で利用されているレガシーソフトウェアに提案手法を適用し，提案手法の実用性の評価を行いたい．その際には，提案手法の自動化が必要となるであろう．大規模なソフトウェアに提案手法を適用する場合に，手作業によりサービスの抽出を行うのは負荷が大きい．ソースコードを入力としてサービスの候補を出力するツールが必要になるものと思われる．また，提案手法により抽出されるのは，サービスの必要条件を満たすサービス候補である．サービスの十分条件の判定を行い，最適粒度のサービスを抽出する手法を考えていきたい．

謝辞

本研究を進めるにあたり，多くの方に御指導，御協力を頂きました．ここに感謝の意を表させて頂きたいと思います．本当にありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，本研究の主指導教官を担当して頂き，大局的な観点から研究に対する御指導を賜りました．また，研究室での活動全般において多大なる御指導を賜りました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 関 浩之 教授には，副指導教官を担当して頂き，本研究の課題や方針などで的確な御助言，御指摘を賜りました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 助教授には，レガシーソフトウェアの調査や，研究発表の場において数多くの御指導，御助言を賜りました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 中村 匡秀 助手には，サービス指向アーキテクチャの基本的な考え方や提案手法の評価など技術的な内容から，研究論文の書き方や研究会などでの発表の作法，研究過程において発生した課題への対処方法に至るまで，全面的に御指導を賜りました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 飯田 元 教授には，ソフトウェアプロセス改善技術について御指導賜りました．パーソナル・ソフトウェア・プロセスや CMM などソフトウェア開発において重要でありながらなかなか理解が難しい事項に関し知識を得ることが出来ました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助手には，主に研究発表の場で数多くの御指導，御助言を賜りました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 Web サービス班の皆様には，週次の打ち合わせや日々の研究活動に際して，様々な御助言，御支援を賜りました．ありがとうございます．

最後に，奈良先端科学技術大学院大学 情報科学研究科の皆様には，研究や日々の生活で様々な御支援を賜りました．おかげさまで非常に充実した，かけがえのない二年間を過ごさせて頂きました．ありがとうございます．

参考文献

[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] A.Lakhotia, ``Rule-based approach to computingmodule cohesion, '' In Proceedings of the 15th Conference on Software Engineering (ICSE-15), pp.34-44, May 1993.

[3] Edward Yourdon, Larry L. Constantine, ``Structured Design Foundamentals of a Discipline of Computer Program and Systems Design, '' YOURDON PRESS, pp.473, 1979.

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

[5] Jarallah Al-Ghamdi, Muhammed Shafique, Sadiq Al-Nasser, Tariq Al-Zubaidi, ``Measuring the Coupling of Procedural Programs, '' IEEE International Conference on Computer Systems and Applications (AICCSA 2001), pp.297-303, June 2001.

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

[7] 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, pp.60-63, January 2005.

[8] 松村 知子, 門田 暁人, 松本 健一, ``潜在コーディング規則違反を原因とするフォールトの検出支援方法の提案, '' 情報処理学会論文誌, Vol.44, No.4, pp.1070-1082, April 2003.

[9] 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.

[10] 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, October 1989.

[11] 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.

[12] S.W.L. Yip, ``Software maintenance in Hong Kong, '' Proceedings of the International Conference on Software Maintenance (ICSM'95), pp.88-97, May 1995.

[13] 城田真琴, ``IT ソリューションフロンティア:技術　SOA―サービス指向アーキテクチャー, '' http://www.itmedia.co.jp/survey/articles/0502/25/news001.html, February 2005.

[14] Hugo Haas, ``W3C Web Service Activity, '' http://www.w3.org/2002/ws/

[15] W.P.Stevens, G.J.Myers, L.L.Constantine, ``Structured design, '' IBM Systems Journal, VOL.13, No.2, pp.115-139, 1974.

[16] 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.

[17] 山田正隆, 陸振宏, ``XML Web サービスの技術動向, '' 東芝レビュー, Vol.58, No.2, pp.3-6, 2003.

[18] 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.

付録 A　酒在庫管理システム要求仕様書

1. 要求仕様

ある酒屋では，すべての在庫(酒）を幾つかのコンテナに載せて，一つの倉庫に保管している.「受付係」の仕事は,倉庫で在庫の積み下ろしを行う「倉庫係」，及び，客の注文を聞いて客に酒を届ける「配送係」との間で数種類の伝票をやりとりしながら，倉庫内の在庫を管理することにある.

1.1 受付係への入力

受付係への入力は，プログラム外部から倉庫係と配送係の出力として，ファイルまたは標準入力より与えられる．また，倉庫係と配送係の出力は区別されず，プログラムへ入力される.

(I1) 積荷票

酒を積んだコンテナが倉庫に搬入されると，倉庫係から「積荷票」が送られてくる．1 台のコンテナに酒を10 銘柄まで混載できるため，積荷票には，搬入されたコンテナの番号(コンテナに付けられた一意なシリアル番号)，搬入年月，搬入日時と共に，積載銘柄数が記入されている．さらに，積載されている銘柄と銘柄毎の数量(積載量）が積載銘柄数だけ繰り返し記入されている.

積荷票　コンテナ番号　搬入年月　搬入日時　銘柄数(銘柄，数量）の繰り返し

(I2) 出庫依頼票

客から酒の注文があると，配送係から「出庫依頼票」が送られてくる．出庫依頼票には，出庫すべき酒の銘柄(1 銘柄のみ)，数量，及び，送り先名が記入されている.

出庫依頼票　送り先名　銘柄　数量

1.2 受付係の出力

受付係の出力は全て，標準出力へなされるものとする．配送係への出力である在庫不足票と，倉庫係への出力である出庫指示票の区別は行わず，順次，標準出力へ出力するものとする．

(01) 在庫不足票

出庫依頼票により出庫を依頼されたが，出庫すべき酒が必要量だけ倉庫にないなどの理由により出庫できない場合，配送係に「在庫不足票」が送られる．在庫不足票には，出庫できなかった酒の銘柄(1 銘柄のみ)，数量，送り先名，及び，対応する出庫依頼票の依頼番号(1.3 の(A1) 参照) が記入されている．

在庫不足票　依頼番号　送り先名　銘柄　数量

(02) 出庫指示票

出庫すべき酒が必要量以上に倉庫にある場合，倉庫係に「出庫指示票」が送られる．複数のコンテナから出庫するような指示も許されるので，出庫指示票には，出庫すべき酒の銘柄(1 銘柄のみ)，送り先名，対応する出庫依頼票の依頼番号と共に，出庫の対象となるコンテナの数(最大10 個) が記入されている．さらに，コンテナ番号，コンテナ毎の数量(出庫量)，及び，空コンテナマーク(出庫によってコンテナが空になるかどうかを示すマーク) が出庫の対象となるコンテナの数だけ繰り返し記入されている．

出庫指示票　依頼番号　送り先名　銘柄　コンテナ数 （コンテナ番号，数量，空コンテナマーク）の繰り返し

1.3 受付係の作業

(A0) 在庫の把握

積荷票や出庫指示票に基づいて，倉庫内のコンテナや酒の在庫状況を常に把握する．

(A1) 出庫依頼票への依頼番号の付加

倉庫係から送られてきた出庫依頼票に，受付順に1 からの通し番号を依頼番号として付加する．依頼番号の付加は，出庫依頼票が送られてきた時点で行う．

(A2) 在庫不足票の作成

出庫依頼票に依頼番号を付加した時点で，出庫依頼票で出庫するよう指定された酒が必要量だけ倉庫にない場合，在庫不足票を作成し，配送係に送ると共に，その控え(「在庫不足票控」) を作成し，保管する．ただし，指定された酒と同一銘柄の酒に対する在庫不足票控が存在する場合には，在庫の有無に関わらず在庫不足票を作成し，配送係に送ると共に，その控えを作成し，保管する．なお，在庫不足票控は在庫不足票と同一形式で，その記入内容は対応する在庫不足票と同じとする．在庫不足票控は，プログラム内部で保持しているだけで良く，ファイルへの保存や出力は行わなくて良い．

(A3) 出庫指示票の作成

出庫依頼票に依頼番号を付加した時点で，出庫依頼票で出庫するよう指定された酒と同一銘柄の酒に対する在庫不足票控が存在せず，かつ，指定された酒が必要量以上に倉庫にある場合，出庫指示票を作成し，倉庫係に送る．また，在庫不足票控が存在する場合には，新たに在庫が搬入された(積荷票が倉庫係から送られてきた) 時点で在庫を確認し，在庫不足票控で指定された酒が必要量以上に倉庫にある場合，出庫指示票を作成し，倉庫係に送る．

なお，出庫指示票の作成においては，次のような制約条件がある．

· 出庫の対象となるコンテナが複数の場合には，コンテナ番号の小さいものから順に記入されているものとする．

· 出庫の対象となるコンテナ数が10 を越える場合には，コンテナ番号の小さいものから順にコンテナを10 個ずつのグループに分け，グループ毎に出庫指示票を作成するものとする．

· 出庫すべき酒が必要量よりも多く倉庫にある場合には，コンテナの搬入年月，及び，搬入日時に基づき，古いものから出庫されるように出庫指示票を作成するものとする．

・在庫不足票控が複数存在し，かつ，新たな在庫搬入によって同時に複数の出庫指示票が作成される場合には，対応する在庫不足票控の依頼番号の小さいものから順に出庫指示票を作成するものとする．
・在庫不足票控で指定された酒が必要量以上に倉庫にある場合であっても，その依頼番号よりも小さな依頼番号を持つ同一銘柄に対する在庫不足票控が存在する場合には，出庫指示票を作成することはできないものとする．

(A4) 在庫不足票控の破棄
在庫不足票控に基づいて出庫依頼票を作成し，倉庫係に送った場合には，対応する在庫不足票控をその時点で破棄する．

2. 外部仕様
プログラムへの入力は「積荷票」と「出庫依頼票」の時系列であり，出力はそれに基づいて作成された「在庫不足票」と「出庫指示票」の時系列である．
プログラムの実行ファイル名はsakaya で，実行方法は次の通り．なお，実行開始時点では，倉庫内に在庫はないものとする．ファイルは複数指定できるが，その場合はfilename1.test, filename2.test... の順に，1ファイルずつ処理していくものとする．

```
%sakaya [ filename1.test filename2.test ...]
```

sakaya への入力はUNIX の標準入力から与えられる．入力には構文的にも意味的にも誤りは含まれないものとする．なお，入力をファイルから与える場合には，そのファイル名の拡張子はtest とする．
sakaya の出力はUNIX の標準出力に対して行う．

3. sakaya の入出力データフォーマット

[入力データフォーマット]

積荷票… 票識別子，コンテナ番号，搬入年月，搬入日時，銘柄数からなる1つの

レコードと，銘柄，数量からなる1つ以上のレコードで構成される．

データ項目	票識別子“T”	コンテナ番号	搬入年月	搬入日時	銘柄数
属性	文字定数	整数	整数	整数	整数
桁数	1	5	4	4	2

データ項目	銘柄	数量
属性	属性文字	整数
桁数	20	4

「搬入年月」は，上位2 桁が西暦年の下位2 桁を，下位2 桁が月(1～12) を，それぞれ表す．
「搬入日時」は，上位2 桁が日(1～31) を，下位2 桁が時(0～23) を，それぞれ表す．

例) 1988 年10 月22 日15 時にコンテナが搬入された場合
搬入年月8810
搬入日時2215
1989 年1 月9 日8 時にコンテナが搬入された場合
搬入年月8901
搬入日時0908

出庫依頼票… 票識別子，送り先名，銘柄，数量からなるレコードである．

データ項目	票識別子“S”	送り先名	銘柄	数量
属性	文字定数	文字	文字	整数
桁数	1	20	20	4

［出力データフォーマット］

在庫不足票… 出力メッセージからなる2 種類のレコードと，依頼番号，送り先名，銘柄，数量からなる1 つのレコードで構成される．

データ項目	出力メッセージ “--- zaiko fusoku hyou ---”
属性	文字定数
桁数	25

データ項目	出力メッセージ “irai-bangou okurisaki-mei meigara suuryou”
属性	文字定数
桁数	61

データ項目	依頼番号	送り先名	銘柄	数量
属性	整数	文字	文字	整数
桁数	5	20	20	4

出庫指示票… 出力メッセージからなる3 種類のレコード, 依頼番号, 送り先名, 銘柄, コンテナ数からなる1 つのレコード, コンテナ番号, 数量, 空コンテナマークからなる1 つ以上のレコードで構成される.

データ項目	出力メッセージ “--- shukko sijihyou ---”
属性	文字定数
桁数	23

データ項目	出力メッセージ “irai-bangou okurisaki-mei meigara kontena-suu”
属性	文字定数
桁数	65

データ項目	依頼番号	送り先名	銘柄	コンテナ数
属性	整数	文字	文字	整数
桁数	5	20	20	2

データ項目	出力メッセージ “kontena-bangou suuryou kara-kontena”
属性	文字定数
桁数	35

データ項目	コンテナ番号	数量	空コンテナマーク
属性	整数	整数	文字定数 空なら“yes”，空でなければ“no”
桁数	5	4	3

注）各データ項目は1 つ以上の空白で，各レコードは1 つの改行記号で，各伝票は1 つの改行記号で，それぞれ区切られるものとする．

＜＜＜正しい入出力の例＞＞＞

＜実行例１＞
「>>」から始まる行は標準入力から読み込まれた行である．

★ 酒屋受付係

>>T 10005 8806 1210 3

積荷票: container 番号10005, 搬入日時88061210, 銘柄数3

>>JUN 50

銘柄: JUN , 数量: 50

```
>>OLD 100

  銘柄: OLD , 数量: 100

>>ROYAL 80

  銘柄: ROYAL , 数量: 80

>>S MATSUMOTO JUN 50

出庫依頼票:
  依頼番号1, 送り先名MATSUMOTO , 銘柄JUN , 数量50

--- shukko sijihyou ---
irai-bangou   okurisaki-mei meigara       kontena-suu
1             MATSUMOTO     JUN           1
kontena-bangou       suuryou       kara-kontena
10005                50            no

>>S OGIHARA OLD 80
出庫依頼票:
  依頼番号2, 送り先名OGIHARA , 銘柄OLD , 数量80

--- shukko sijihyou ---
irai-bangou   okurisaki-mei meigara       kontena-suu
2             OGIHARA       OLD           1
kontena-bangou       suuryou       kara-kontena
10005                80            no

>>S MII ROYAL 80
```

```
出庫依頼票:
  依頼番号3, 送り先名MII , 銘柄ROYAL , 数量80

--- shukko sijihyou ---
irai-bangou   okurisaki-mei meigara       kontena-suu
3             MII           ROYAL         1
kontena-bangou       suuryou       kara-kontena
10005                80            no

>>S INADA OLD 20
出庫依頼票:
  依頼番号4, 送り先名INADA , 銘柄OLD , 数量20

--- shukko sijihyou ---
irai-bangou   okurisaki-mei meigara       kontena-suu
4             INADA         OLD           1
kontena-bangou       suuryou       kara-kontena
10005                20            yes
```

＜実行例２＞ファイル指定

```
% ./liq2 test6

★ 酒屋受付係

--- zaiko fusoku hyou ---
irai-bangou   okurisaki-mei meigara       suuryou
1             YAMADA        JUN           200
--- shukko sijihyou ---
irai-bangou   okurisaki-mei meigara       kontena-suu
```

```
1               YAMADA          JUN             10
kontena-bangou          suuryou         kara-kontena
10001                   10              no
10002                   20              yes
10003                   30              yes
10004                   5               yes
10005                   15              yes
10006                   10              yes
10007                   30              yes
10008                   10              yes
10009                   20              yes
10010                   20              yes
--- shukko sijihyou ---
irai-bangou     okurisaki-mei meigara           kontena-suu
1               YAMADA          JUN             3
kontena-bangou          suuryou         kara-kontena
10011                   10              yes
10012                   5               yes
10013                   15              no
```

付録 B　酒在庫管理プログラム A

```
/************************************************/
/* sakaya.c 酒管理プログラム                    */
/*                                              */
/* 実行書式 : $./sakaya [file1, file2, ... ]    */
/*                                              */
/* (c) 中村匡秀 masa-n@is.naist.jp 2004/12/03   */
/************************************************/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#define BUFSIZE  1024

/*―――――― 在庫不足票メッセージヘッダ ――――――*/
#define H_MSG1 "―― zaiko fusoku hyou ――"
#define H_MSG2 "irai-bangou okurisaki-mei       meigara
suuryou"

/*―――――― 出庫指示票メッセージヘッダ ――――――*/
#define S_MSG1 "―― shukko sijihyou ――"
#define S_MSG2 "irai-bangou okurisaki-mei       meigara
kontena-suu"
#define S_MSG3 "kontena-bangou suuryou kara-kontena"

/*―――――― 酒構造体リスト――――――――――*/
struct LIQLIST {
  char brand[21];   /*銘柄*/
  int  qty;         /*数量*/
  struct LIQLIST *next;
};

/*―――――― 在庫不足票控リスト――――――――――*/
struct HUSOKU {
  int reqnum;       /*受付番号*/
  char to[21];      /*配送先*/
  char brand[21];   /*銘柄*/
  int  qty;         /*数量*/
  int  done;        /*済マーク*/
  struct HUSOKU *next;
};


/*―――――― コンテナ構造体(2分木) ――――――――*/
struct CONTAINER {
  int cont_num;     /*コンテナ番号*/
  int date;         /*搬入日時*/
  int brand_num;    /*銘柄数*/
  struct LIQLIST    *liq; /*コンテナに格納される酒のリスト
*/
  struct CONTAINER *left;
  struct CONTAINER *right;
};

/*―――――― 出庫対象のコンテナ2分木 ――――――――*/
struct CONT_TREE {
  struct CONTAINER *cont; /*コンテナへのポインタ*/
  int qty;              /*コンテナから出庫する数*/
  int mark;             /*空コンテナマーク*/
  int done;             /*表示したかどうか*/
  struct CONT_TREE *left;
  struct CONT_TREE *right;
};

struct CONTAINER *souko=NULL;    /*コンテナ倉庫のルート*/
struct HUSOKU    *husoku = NULL; /*在庫不足票のルート*/

/************************************************/
/* create_husoku                                */
/* 在庫不足票控を新たに作り，値をセットする     */
/************************************************/
struct HUSOKU *create_husoku(int reqnum, char to[],
                                        char brand[], int
qty) {
  struct HUSOKU *hp;

  hp = (struct HUSOKU *)malloc(sizeof(struct HUSOKU));
  hp->reqnum = reqnum;
  strcpy(hp->to, to);
  strcpy(hp->brand, brand);
  hp->qty = qty;
  hp->done = 0;
  hp->next = NULL;
  return hp;
}
/************************************************/
/* insert_husoku                                */
/* 不足票をリストにつなぐ．依頼番号reqnumは，通し */
/* 番号が与えられるため，結果的に依頼が早いものが */
/* リストの前に来る                             */
/************************************************/
void insert_husoku(struct HUSOKU *root,
                           int reqnum, char to[],
                           char brand[], int qty) {
  struct HUSOKU *hp;

  /*不足票リストを探索．最後まで*/
  for (hp=root; hp->next!=NULL; hp=hp->next);
  hp->next = create_husoku(reqnum, to, brand, qty);
}

/************************************************/
/* show_husoku                                  */
/* 不足票リストを表示する                       */
/************************************************/
void show_husoku(struct HUSOKU *root) {
  struct HUSOKU *hp;
  printf("□□□□□在庫不足票リスト□□□□□¥n");
  for (hp=root->next; hp!=NULL; hp=hp->next) {
    printf("%05d¥t%-20s%-20s¥t%4d¥n",
               hp->reqnum, hp->to, hp->brand, hp->qty);
  }
}

/************************************************/
```

```
/* create_liqlist                                    */
/* 酒リスト構造体を新たに作り，値をセットする     */
/*****************************************************/
struct LIQLIST *create_liqlist(char brand[], int qty) {
  struct LIQLIST *lp;

  lp = (struct LIQLIST *)malloc(sizeof(struct LIQLIST));
  strcpy(lp->brand, brand);
  lp->qty = qty;
  lp->next = NULL;
  return lp;
}

/*****************************************************/
/* insert_liqlist                                    */
/* 酒リストにつなぐ．成功すれば1，失敗は0を返す   */
/* 同じコンテナ内に，同じ銘柄があった場合は，      */
/* まとめて入れる                                    */
/*****************************************************/
int insert_liqlist(struct LIQLIST *root, char brand[], int
qty) {
  struct LIQLIST *lp;
  int count=0;

  /*酒リストを探索．最後まで行くか，銘柄見つかるまで*/
  for (lp=root; lp!=NULL && strcmp(lp->brand, brand)!=0;
       count++, lp=lp->next);

  if (count>10){ /*10銘柄よりも多い*/
    fprintf(stderr, "Warning: Exceeding 10 brands.
Rejected¥n");
    return 0;
  }

  if (lp==NULL) { /*最後まで行ったら先頭につなぐ*/
    lp = create_liqlist(brand, qty);
    lp->next = root->next;
    root->next = lp;
  } else {          /*銘柄見つかった*/
    lp->qty += qty; /*新たに水増しする*/
  }
  return 1;
}

/*****************************************************/
/* create_container                                  */
/* コンテナ構造体を新たに作り，値をセットする     */
/*****************************************************/
struct CONTAINER *create_container(int cont_num, int date,
                                                    int
brand_num, struct LIQLIST *liq) {
  struct CONTAINER *cp;

  cp = (struct CONTAINER *)malloc(sizeof(struct
CONTAINER));
  cp->left = cp->right = NULL;
  cp->cont_num = cont_num;
  cp->date = date;
  cp->brand_num = brand_num;
  cp->liq = liq;

  return cp;
}

/*****************************************************/
/* show_liqlist                                      */
/* 酒リストの表示．デバッグ用.                     */
/*****************************************************/
void show_liqlist(struct LIQLIST *root) {
  struct LIQLIST *lp;
  /*リストの先頭はダミーなので次から*/
  for (lp=root->next; lp!=NULL; lp=lp->next) {
    printf("銘柄:%-20s¥t数量:%4d¥n",
           lp->brand, lp->qty);
  }
}

/*****************************************************/
/* show_container                                    */
/* コンテナを表示する．デバッグ用.                 */
/*****************************************************/
void show_container(struct CONTAINER *root) {
  if (root!=NULL) {
    show_container(root->left);  /*左の子から表示*/
    printf("コンテナ番号:%5d¥t搬入日時:%08d¥t銘柄
数:%2d¥n",
           root->cont_num, root->date,
root->brand_num);
    show_liqlist(root->liq);     /*酒リストの表示*/
    show_container(root->right); /*右の子を表示*/
  }
}

/*****************************************************/
/* insert_container                                  */
/* コンテナをrootで示される2分木に新たに挿入する. */
/* 日付の若いものが左の子に来るように，2分木を    */
/* 作成する．既に存在するコンテナ番号が来た場合，  */
/* メッセージを出して棄却する.                     */
/*****************************************************/
struct CONTAINER *insert_container(struct CONTAINER *root,
                                                    int
cont_num, int date,
                                                    int
brand_num,
                                                    struct
LIQLIST *liq) {
  if (root==NULL) { /*新たに作る*/
    root = create_container(cont_num, date, brand_num, liq);
  } else if (root->cont_num == cont_num) { /*同じコンテナ
番号があった*/
    fprintf(stderr, "Warning: Container %5d already exists.
Rejected.¥n", cont_num); /*2分木には入れない*/
  } else if (root->date > date) { /*コンテナが古い時*/
```

```
    /*左の子に入れる*/
    root->left = insert_container(root->left, cont_num,
                                                  date,
brand_num, liq);
  } else if (root->date <= date) { /*コンテナが新しい時*/
    /*右の子に入れる*/
    root->right =  insert_container(root->right, cont_num,
                                                  date,
brand_num, liq);
  }
  return root;
}

/*************************************************/
/* check_tsumini_param                           */
/* 積荷票のパラメータ値をチェックする            */
/* パラメータ異常値なら0, 正常なら1を返す        */
/*************************************************/
int check_tsumini_param(int cont_num, int yymm,
                                    int ddhh, int
meigara_num) {
  if (( cont_num < 1 || cont_num > 99999) /*コンテナ番号5
桁*/
      || ( yymm<1 || yymm > 9999) /*年月4桁*/
      || ( ddhh<1 || ddhh > 9999) /*日時4桁*/
      || ( meigara_num < 1 || meigara_num > 10)) /*銘柄数
10まで*/
    return 0;

  return 1;
}



/*************************************************/
/* check_sake_param                              */
/* 積荷票の酒銘柄, 数量のパラメータ値をチェック  */
/* パラメータ異常値なら0, 正常なら1を返す        */
/*************************************************/
int check_sake_param(char brand[], int qty) {
  if (( strlen(brand)>20)         /*銘柄20文字まで*/
     || ( qty<0 || qty > 9999))   /*数量4桁まで*/
    return 0;

  return 1;
}

/*************************************************/
/* process_tsumini                               */
/* buf の積荷票をチェックし, 倉庫に入れる.       */
/* 返り値は1:搬入成功, 0:失敗                    */
/*************************************************/
int process_tsumini(char buf[], FILE *fp) {
  int cont_num;          /*コンテナ番号*/
  int date, yymm, ddhh;  /*搬入年月日時, 年月, 日時*/
  int brand_num;         /*銘柄数*/
  int i, qty;
  char brand[BUFSIZE];
  struct LIQLIST *liq_root;

  /*積荷表の処理*/
  if (sscanf(buf+1, "%d %d %d %d", &cont_num, &yymm,
                &ddhh, &brand_num) != 4) { /*パラメータ個
数のチェック*/
    fprintf(stderr, "Invalid format: %s", buf);
    return 0;
  } else if (!check_tsumini_param(cont_num, yymm, ddhh,
brand_num)) {
    /*パラメータ値のチェック*/
    fprintf(stderr, "Invalid parameter value: %s", buf);
    return 0;
  } else { /*酒の処理*/
    liq_root = create_liqlist("dummy", 0); /*ダミー*/
    for (i=0; i<brand_num; i++) { /*銘柄数繰り返し, 酒を入
れていく*/
      fscanf(fp, "%s %d", brand, &qty);
      if (!check_sake_param(brand, qty)) { /*酒のパラメー
タ値のチェック*/
            fprintf(stderr, "Invalid sake pamameter
value: %s %d", brand, qty);
            return 0;
      }
      insert_liqlist(liq_root, brand, qty); /*酒を入れてい
く*/
    }
    date = 10000*yymm + ddhh;   /*日時の計算*/
    souko = insert_container(souko, cont_num, date,
brand_num, liq_root); /*倉庫に入れる*/
    return 1;
  }
}

/*************************************************/
/* check_irai_param                              */
/* 依頼票のパラメータ値をチェックする            */
/* パラメータ異常値なら0, 正常なら1を返す        */
/*************************************************/
int check_irai_param(char to[], char brand[], int qty) {
  if ((strlen(to)>20) /*あて先は20桁まで*/
      || (strlen(brand)>20) /*酒の銘柄は20桁まで*/
      || (qty < 0 || qty > 9999)) /*数量4桁*/
    return 0;

  return 1;
}

/*************************************************/
/* is_wait                                       */
/* 銘柄brandで酒待ちをしている在庫不足票控えが   */
/* あれば, 1を返す. 無ければ0を返す.             */
/*************************************************/
int is_wait(char brand[]) {
  struct HUSOKU *hp;

  /*husoku は在庫不足票控えのルート*/
```

```
  for (hp=husoku->next; hp!=NULL; hp=hp->next)
    /*銘柄が見つかった，かつ，未処理の在庫不足票*/
    if (strcmp(hp->brand, brand)==0 && hp->done ==0)
      return 1;
  return 0;
}

/*************************************************/
/* count_liquor                                  */
/* 銘柄brandの酒がrootで示される倉庫に何本あるか  */
/* を返す.                                       */
/*************************************************/
int count_liquor(struct CONTAINER *root,
                         char brand[]) {
  int count=0;
  struct LIQLIST *lp;

  if (root!=NULL) {
    /*左の子をサーチ*/
    count += count_liquor(root->left, brand);
    /*当該コンテナをサーチ*/
    for (lp=root->liq->next; lp!=NULL; lp=lp->next)
      /*銘柄が見つかった*/
      if (strcmp(lp->brand, brand)==0) {
            count += lp->qty;  /*在庫数を加える*/
      }
    /*右の子をサーチ*/
    count += count_liquor(root->right, brand);
  }
  return count;
}

/*************************************************/
/* is_liquor                                     */
/* 倉庫の中に銘柄brandの酒がqty本以上あれば1を，   */
/* 無ければ0を返す.                              */
/*************************************************/
int is_liquor(char brand[], int qty) {
  /*souko は倉庫をあらわすコンテナ2分木のルート*/
  if (count_liquor(souko, brand) >= qty)
    return 1;
  else
    return 0;
}

/*************************************************/
/* is_empty_container                            */
/* コンテナcontが空であれば1を返す．さもなくば，   */
/* 無ければ0を返す.                              */
/*************************************************/
int is_empty_container(struct CONTAINER *cont){
  struct LIQLIST *lp;

  for (lp=cont->liq->next; lp!=NULL; lp=lp->next)
    if (lp->qty!=0) return 0;

  return 1;
}

/*************************************************/
/* create_conttree                               */
/* 出庫対象コンテナ2分木構造体を作る              */
/*************************************************/
struct CONT_TREE *create_conttree(struct CONTAINER *cont,
                                  int qty)
{
  struct CONT_TREE *cp;

  cp = (struct CONT_TREE *)malloc(sizeof(struct
CONT_TREE));
  cp->left = cp->right = NULL;
  cp->cont = cont;
  cp->qty = qty;
  cp->done = 0;
  cp->mark = is_empty_container(cont); /*空コンテナマーク
*/

  return cp;
}

/*************************************************/
/* insert_conttree                               */
/* 出庫対象コンテナをrootで示される2分木に新たに  */
/* 挿入する．コンテナ番号の小さいものが左の子に来 */
/* るように，2分木を作成する.                     */
/* 既に存在するコンテナ番号が見つかった場合，      */
/* メッセージを出す.                             */
/*************************************************/
struct CONT_TREE *insert_conttree(struct CONT_TREE *root,
                                  struct
CONTAINER *cont,
                                  int qty)
{
  int cond;

  if (root==NULL) { /*新たに作成する*/
    root = create_conttree(cont, qty);
  } else {
    /*コンテナ番号の差分を取る*/
    cond = (cont->cont_num) - (root->cont->cont_num);
    if (cond == 0)  /*同じコンテナ番号があった*/
      /*警告を出し，何もしない*/
      fprintf(stderr, "Warning: Container %5d is visited
twice.¥n",
              cont->cont_num);
    else if (cond < 0)  /*コンテナ番号が小さい時は*/
      /*左の子に入れる*/
      root->left = insert_conttree(root->left, cont, qty);
    else if (cond > 0)  /*コンテナ番号が大きい時は*/
      /*右の子に入れる*/
      root->right =  insert_conttree(root->right, cont,
qty);
  }
  return root;
```

```
}

/*************************************************/
/* update_souko                                  */
/* 要求された銘柄brand, 数量qty を基に, 出庫対象 */
/* となるコンテナを調べ, あれば倉庫を更新する.    */
/* 出庫対象のコンテナは, rootで示されるコンテナ   */
/* 2分木につなぐ. また, 出庫対象コンテナ数を      */
/* countに返す.                                  */
/*************************************************/
struct CONT_TREE *update_souko(struct CONTAINER *cont,
                               struct
CONT_TREE *root,
                               char brand[],
                               int *qty,
                               int *count) {
  struct LIQLIST *lp;
  int num; /*出庫対象コンテナから出庫できる酒の数量*/

  if (cont!=NULL && *qty>0) { /* *qtyには現在の要求量が入
っている*/
    /*仕入れが旧いもの(左の子)から倉庫を探索*/
    root = update_souko(cont->left, root, brand, qty,
count);

    if (*qty>0) { /*まだ必要であれば当該コンテナを調べる*/
      for (lp=cont->liq->next; lp!=NULL; lp=lp->next) {
            if (strcmp(lp->brand, brand)==0 && lp->qty > 0)
{
              /*当該コンテナに酒が見つかった!!*/
              *count += 1;
              if (lp->qty >= *qty) { /*酒が足りた*/
                num = *qty;          /*出庫するのはリクエス
トした数量*/
                lp->qty -= *qty;     /*コンテナの数量を減ら
す*/
                *qty = 0;            /*要求量を0に*/
              } else { /*足りなかった*/
                num = lp->qty;       /*出庫するのはコンテナ
にある数量*/
                *qty -= lp->qty;     /*要求量を減らす*/
                lp->qty = 0;         /*コンテナの数量を0に
*/
              }
              /*コンテナリストにつなぐ*/
              root = insert_conttree(root, cont, num);
            }
      }
    }
    if (*qty>0) /*まだ必要であれば, 次のコンテナを調べる*/
      root = update_souko(cont->right, root, brand, qty,
count);
  }

  return root;
}
```

```
/*************************************************/
/* show_conttree                                 */
/* 出庫対象コンテナ2分木を表示. 最大count回表示   */
/* する. また, 以前に表示されたもの(doneフラグ1)は*/
/* 表示しない. 返り値は, 後何回表示すべきか       */
/*************************************************/
int show_conttree(struct CONT_TREE *root, int count) {

  if (root!=NULL && count > 0) {
    count = show_conttree(root->left, count);
    if (count > 0 && root->done==0) {
      printf("   %05d¥t%4d¥t%s¥n", root->cont->cont_num,
                 root->qty, (root->mark==0)?"no":"yes");
      root->done = 1;
      count--;
    }
    if (count > 0)
      count = show_conttree(root->right, count);

  }
  return count;
}

/*************************************************/
/* free_conttree                                 */
/* 出庫対象コンテナの2分木を解放する             */
/*************************************************/
void free_conttree(struct CONT_TREE *root) {

  if (root!=NULL) {
    free_conttree(root->left);
    free_conttree(root->right);
    free(root);
  }
}

/*************************************************/
/* print_shukko_shijihyo                         */
/* 出庫指示票を作成(表示)する. 出庫対象のコンテナ */
/* 数countが10を超える場合は, 分けて表示する.     */
/*************************************************/
void print_shukko_shijihyo(struct CONT_TREE *root,
                           int reqnum,
                           char to[],
                           char brand[],
                           int qty,
                           int count) {
  do {
    printf("¥n%s¥n", S_MSG1); /*メッセージヘッダを出力*/
    printf("%s¥n", S_MSG2);

    printf("%05d      %-20s %-20s   %2d¥n",
              reqnum, to, brand, (count > 10)?10:count);
    /*最大10個の出庫対象コンテナを表示*/
    printf("%s¥n", S_MSG3); /*メッセージヘッダを出力*/
    show_conttree(root, 10);
    count -= 10;
```

```
  } while(count > 0);
}


/*************************************************/
/* available_conttree                            */
/* 要求された銘柄brandと数量qtyを基に，倉庫を更新 */
/* して，出庫対象となるコンテナの2分木を取得      */
/*************************************************/
struct CONT_TREE *available_conttree(struct CONT_TREE
*root,
                                     char
brand[],
                                     int
qty,
                                     int
*count) {
  root = update_souko(souko, root, brand, &qty, count);

  return root;
}

/*************************************************/
/* process_shukko                                */
/* 出庫処理を行う．出庫対象コンテナを取得し，出庫 */
/* 指示票を作成する．                            */
/*************************************************/
void process_shukko(int reqnum, char to[], char brand[],
                    int qty){

  struct CONT_TREE *root=NULL; /*出庫対象コンテナのルート
*/
  int count = 0;

  /*倉庫から銘柄brandの酒を取り出し，出庫対象コンテナを取
得*/
  /*また出庫対象コンテナ数をcountで取得する*/
  root = available_conttree(root, brand, qty, &count);

  /*出庫指示票の作成*/
  print_shukko_shijihyo(root, reqnum, to, brand, qty,
count);

  /*コンテナリストを開放*/
  free_conttree(root);
}


/*************************************************/
/* process_husoku                                */
/* 在庫不足処理を行う．在庫不足票を作成し，不足票 */
/* の控えのリストに挿入する．                    */
/*************************************************/
void process_husoku(int reqnum, char to[], char brand[],
                    int qty){

  /*在庫不足票を作成(表示)する*/
  //printf("●●●●●在庫不足票●●●●●¥n");
  printf("¥n%s¥n", H_MSG1);
  printf("%s¥n", H_MSG2);

  printf("%05d      %-20s %-20s %4d¥n", reqnum, to, brand,
qty);

  /*在庫不足票控えのリストにつなぐ．*/
  insert_husoku(husoku, reqnum, to, brand, qty);

}


/*************************************************/
/* process_irai                                  */
/* buf の出庫依頼票をチェックし，依頼番号reqnumで */
/* 受け付ける．倉庫を調べて酒があれば出庫処理，   */
/* 無ければ在庫不足処理を行う．                  */
/*************************************************/
int process_irai(int reqnum, char buf[], FILE *fp) {
  char brand[BUFSIZE]; /*酒の銘柄*/
  char to[BUFSIZE];    /*配送先*/
  int  qty;            /*数量*/

 /*積荷表の処理*/
  if (sscanf(buf+1, "%s %s %d", to, brand,
             &qty) != 3) { /*パラメータ個数のチェック
*/
    fprintf(stderr, "Invalid format: %s", buf);
    return 0;
  } else if (!check_irai_param(to, brand, qty)) {
    /*パラメータ値のチェック*/
    fprintf(stderr, "Invalid parameter value: %s", buf);
    return 0;
  } else {
    /*銘柄brand で酒待ちをしている人がいない，かつ，
     酒が充分にあるならば*/
    if (!is_wait(brand) && is_liquor(brand, qty)) {
      /*出庫手続きを行う*/
      process_shukko(reqnum, to, brand, qty);
    } else {
      /*在庫不足票を発行する*/
      process_husoku(reqnum, to, brand, qty);
    }
  }
  return 1;
}

/*************************************************/
/* is_husoku_wait                                */
/* 与えられた在庫不足票よりも依頼番号が若い，別の */
/* 未処理の不足票が存在すれば1を返す             */
/*************************************************/
int is_husoku_wait(struct HUSOKU *current) {
  struct HUSOKU *hp;

  for (hp=husoku->next; hp!=current; hp=hp->next)
```

```
    if (strcmp(hp->brand, current->brand)==0 &&
            hp->done == 0) /*同銘柄で未処理の不足票が存在
*/
      return 1;

  return 0;
}

/*************************************************/
/* resolve_husoku                                */
/* 在庫不足を解消する                            */
/*************************************************/
void resolve_husoku(void) {
  struct HUSOKU *hp, *nhp;

  /*在庫不足票控えをサーチ*/
  for (hp=husoku->next; hp!=NULL; hp=hp->next){
    /*自分より若い未処理の不足票がなく，酒が十分にあれば*/
    if (!is_husoku_wait(hp) &&
            is_liquor(hp->brand, hp->qty)) {
      /*出庫手続きを行う*/
      process_shukko(hp->reqnum, hp->to, hp->brand,
hp->qty);
      hp->done = 1; /*済マークをつける*/
    }
  }

  /*処理済の在庫不足票控えを消去*/
  for (hp=husoku; hp->next!=NULL;){
    nhp = hp->next;
    if (nhp->done == 1) {
      hp->next = nhp->next;
      free(nhp);
    } else
      hp=hp->next;
  }
}

/*************************************************/
/* copy_nonempty_container                       */
/* 空コンテナ以外のコンテナをsrcからdestにコピー */
/* する                                          */
/*************************************************/
struct CONTAINER *copy_nonempty_container(struct CONTAINER
*src,
                                          struct
CONTAINER *dest) {
  struct LIQLIST *lp, *nlp;

  if (src!=NULL) {
    /*左の子のコピー*/
    dest = copy_nonempty_container(src->left, dest);
    /*右の子のコピー*/
    dest = copy_nonempty_container(src->right, dest);

    if (!is_empty_container(src)) { /*空でなかったら*/
      /*まず酒リストのコピー lpからnlpへ*/
      nlp = create_liqlist("dummy", 0); /*ダミー*/
      for (lp=src->liq->next; lp!=NULL; lp=lp->next) {
            insert_liqlist(nlp, lp->brand, lp->qty); /*酒
を入れていく*/
      }
      /*新たなコンテナとして，destに挿入*/
      dest = insert_container(dest, src->cont_num,
src->date,
                              src->brand_num,
nlp);
    }
  }
  return dest;
}

/*************************************************/
/* free_container                                */
/* コンテナ2分木を解放する                       */
/*************************************************/
void free_container(struct CONTAINER *root) {
  struct LIQLIST *lp, *nlp;
  if (root!=NULL) {
    free_container(root->left);
    free_container(root->right);

    /*当該コンテナの酒リストを解放*/
    for (lp=root->liq; lp!=NULL;) {
      nlp = lp;
      lp = lp->next;
      free(nlp);
    }
    free(root); /*コンテナ本体を解放*/
  }
}

/*************************************************/
/* free_empty_container                          */
/* 倉庫から空コンテナを削除する                  */
/*************************************************/
void *free_empty_container() {
  struct CONTAINER *souko_new=NULL;

  /*空でないコンテナのみを新たにsouko_newに構築*/
  souko_new = copy_nonempty_container(souko, souko_new);
  /*オリジナルの倉庫を解放*/
  free_container(souko);
  /*souko_new を新たな倉庫とする*/
  souko = souko_new;
}

/*************************************************/
/* process_liquor                                */
/* 入力ファイルfpを処理し，酒の在庫管理を行う    */
/*************************************************/
void process_liquor(FILE *fp) {
  char buf[BUFSIZE];
```

```
  int linenum = 0;
  static int reqnum = 0; /*依頼番号*/

  if (husoku==NULL)
    husoku = create_husoku(0,"","",0); /*不足票リストのダ
ミー*/

  /*ファイルの最後まで一行ずつ処理*/
  while(fgets(buf, BUFSIZE, fp)!=NULL) {
    linenum++;
    switch(buf[0]) {
    case 'T': /* 積荷票の場合*/
      process_tsumini(buf, fp);     /* 積荷処理 */
      resolve_husoku();             /* 在庫不足を解決*/
      break;
    case 'S': /* 出庫依頼票の場合*/
      reqnum++; /*依頼番号をインクリメント*/
      process_irai(reqnum, buf, fp); /*依頼処理*/
      break;
    case '#': case '¥n': case '¥r': /*コメント空行*/
      break;
    default:
      fprintf(stderr, "Invalid Request: line%5d:%s¥n",
linenum, buf);

    }
  }
  free_empty_container(); /*1ファイルごとに, 空コンテナを
削除*/
}

/************************************************/
/* main                                         */
/* メイン関数. コマンドライン引数から, 入力ファ   */
/* イルを取得, ファイルごとに処理を行う.          */
/************************************************/

int main(int argc, char *argv[]) {
  FILE *fp;
  int i;

  if(argc > 1){
    for(i=1; i<argc; i++){
      if((fp = fopen(argv[i], "r")) == NULL){
            /* ファイルが開けなかった場合は*/
        fprintf(stderr, "%s: No such file or
directory¥n",argv[i]);
        continue; /* エラーを表示して次のファイルの処理に
移る */
      }
      process_liquor(fp); /*酒の処理*/
      fclose(fp);
    }
  }
  else{
    /* コマンドラインから何も渡されなかった場合, 標準入力
から受け取る */
    process_liquor(stdin);
  }

  free_container(souko); /*後片付け*/
  return 0;
}
```