2016年度　修士論文

合成サービスを用いたソフトウェアの潜在要求発見の支援手法に関する研究

2017年1月30日(月)提出

指導：深澤　良彰　教授

早稲田大学　基幹理工学部
情報理工学科　深澤研究室

学籍番号：5115F044-5
鶴貝　康男

目 次

第 1 章　はじめに **1**
1.1 背景 1
1.1.1 ソフトウェア開発とは 1
1.1.2 近年の要求獲得における傾向 2
1.2 問題 4
1.3 モチベーション 4
1.4 提案 5

第 2 章　関連研究 **7**

第 3 章　手法 **11**
3.1 全ての合成サービスの導出 (ステップ 1) 12
3.2 フィルタリング (ステップ 2) 18
3.2.1 他のソフトウェアが実装済みの機能のフィルタリング 18
3.2.2 無駄な処理を含む組み合わせのフィルタリング . . 21
3.2.3 ユーザによる作成が困難なデータのの入力が必要な機能のフィルタリング 30
3.3 グループ化 (ステップ 3) 34

第 4 章　評価 **37**
4.1 データセット 37
4.2 フィルタリング数における本手法の有効性 (RQ1) 39
4.3 ユーザが必要とする機能の獲得支援における本手法の有効性 (RQ2) 41
4.4 他のソフトウェアに存在しない新しい特徴となる要求の獲得支援における本手法の有効性 (RQ3) 43
4.5 無駄なデータの循環の識別のための Web サービスの実行数削減における本手法の有効性 (RQ4) 44
4.6 考察 45

第 5 章　おわりに **49**
5.1　まとめ 49
5.2　今後の課題 50

第1章　はじめに

1.1　背景

1.1.1　ソフトウェア開発とは

ソフトウェアとは，現実世界においてユーザが抱える問題を解決し，そのユーザの要求を満たすために作成されるものである．要求とは，ユーザが問題を解決するために，ソフトウェアに求める事を指す．

ソフトウェアの開発工程は，大きく分けて，要求分析，設計，実装の3つの工程によって構成される．

要求分析は，ソフトウェアが実装する必要がある機能や性能を定義する工程である．要求分析には，更に，要求獲得と要件定義の2つの工程が存在する．要求獲得では，ユーザが現実世界において抱えている問題を洗い出し，その問題に対するユーザの要求を明確にする．そして要件定義で，それらのユーザの要求を満たすために，ソフトウェアが実装する必要がある機能や性能を，要件として定義する．これらの作業を行う人を，要求分析者と呼ぶ．

設計の工程では，要求分析の工程で定義された要件をプログラムとして実現する方法を考える．そして最後に実装の工程で，それらの要件を実現する実行可能なプログラムを作成する．

ここで，ソフトウェア開発が上手くいかない原因の多くは，要求に関係したものとなっている．様々なITベンダが回答したソフトウェア開発における失敗の原因の2位が「不完全な要求と仕様」であり，3位が「要求と仕様の変更」である[8]．また，ソフトウェア開発における中断の原因の1位が「不完全な要求」であり，5位が「要求と仕様の変更」である[8]．そして，要求獲得に問題があった場合，その修正に掛かるコストは全開発費の30パーセント〜50パーセント[9]と言われている．これらの事から，ソフトウェア開発工程において，要求獲得の工程が重要な意味を持つ事が分かる．

1.1.2 近年の要求獲得における傾向

要求獲得において，ユーザは始めから明確な要求を持っている訳では無く，多くの場合はユーザ自身が自分の抱える問題を十分に把握しておらず，それに対する要求も曖昧となっている．また，ユーザ自身が自分の持つ要求に気付いていない事もある．そこで，要求獲得では，ユーザが始めから持っている明確な要求を獲得する事に加え，曖昧な要求を明確にする事，そしてユーザ自身が気付いていない要求を獲得する事が必要となる．従来の要求獲得では，その内，主に曖昧な要求を明確にする事に主眼が置かれており，それを解決するために様々な手法が確立されている．代表的なものとして，資料収集，インタビュー [10] がある．資料収集は，開発する対象のソフトウェアに関する既存の資料を収集し，それらを分析する事によって要求を獲得する手法である．インタビューは，ユーザに予め定めた質問をし，その回答を分析する事によって要求を獲得する手法である．これらの手法は，要求分析者がユーザが抱えている曖昧な要求を理解し，それらを明確にするために使われる手法であり，一方で，ユーザが抱えていないユーザ自身が気付いていない要求を獲得する事の支援とはならない．

現在ソフトウェアの数は膨大な数になっており，同じドメインに属するソフトウェアを複数見つける事が出来る [11]．例えばホテルのドメインに所属するソフトウェアを，sercus[3] という Web アプリケーションの検索サイトで検索すると，memonda，Expedia，Priceline，楽天トラベル，ポンパレ，トルノス，フォートラベル，TripAdvisor，じゃらん，一休．com，イープラスクーポンと合計 11 個のソフトウェアを見つける事が出来る．この様に，ソフトウェアの顧客獲得競争は激しくなっているため，それぞれのソフトウェアは，他のソフトウェアに存在しない新しい特徴を身に付ける必要がある [13]．

そのため，近年，要求獲得における既存研究では，その新しい特徴を獲得するため，要求分析者の創造力を支援する事によって，ユーザ自身が気付いていない要求を獲得する事に主眼が置かれている [16][12]．

J.Robertson[16] は，現在膨大な数となっている他のソフトウェアとの顧客獲得競争に勝つために，要求分析者は発明家であるべきだ，と主張している．そこでは，要求獲得において発明を引き起こすための様々な考え方が紹介されている．その内の 1 つとして，既存の製品を 1 つのアイディアとして捉え，その製品をより良くするためにはどうするか?，あるいは，その製品から別の何かを作れないか？という考え方が提案されて

いる．この様な既存のアイディアを使う考え方は，創造力を支援する上で，基本となる考え方である [17].

N.Maiden ら [12] は，要求獲得の工程は，要求分析者の創造力によって解決される工程と見做すべきだと主張している．それによって，これから要求獲得の工程で，創造力を支援する技術や理論などが優先的に利用される様になり，より効果的な要求獲得が行われる様になる事を目指している．この研究の中で，N.Maiden ら [12] は，創造力を，Sternberg[14] の定義に基づき,「新しいもの，かつ必要なものを生産するための能力」と定義している．この創造力の定義は，要求獲得者の創造力を支援する他の様々な手法においても，頻繁に使われるものである．

この様に，近年の要求獲得における既存研究では，要求分析者の創造力を支援する手法が求められている事が分かる．

要求分析者の創造力を支援する手法として，既存研究が提案している手法は，大きく分けて，トップダウンアプローチとボトムアップアプローチに分かれる．

トップダウンアプローチとは，一般的な「ユーザは，何を実現したいのか？」という観点から要求を考える手法である [20].

T.Bhowmik ら [11] は，過去の開発に使われたドキュメントやコメントの単語を抽出し，それらの単語同士を組み合わせたものを，要求分析者にヒントとして提示する事によって,「ユーザは，何を実現したいのか？」という事を考える要求分析者の創造力を支援する手法を提案している．

しかし，この様なトップダウンアプローチにおける既存手法 [11] では，他のソフトウェアに存在しない新しい特徴となる要求を獲得出来るかどうか，はその支援をされた要求分析者の創造力に依存している．そのため，確実にそれらの新しい要求を獲得出来るとは限らず，他の手法との併用が必要となる．

一方でボトムアップアプローチとは,「現時点で何が実現出来るのか？」という観点から要求を考える手法である．現在，要求獲得が大変な作業であると認識されている 1 つの理由として，要求分析者が，その時点でソフトウェアが実現出来る事を分かっていない，という事が挙げられる．そのため，要求獲得において「現時点で何が実現出来るのか？」という情報を元に要求を考える事が必要である [18].

K.Zachos ら [21] は,「現時点で何が実現出来るのか？」という情報として，Web サービスを提示する手法を提案している．Web サービスとは，Web 上にある既存の特定の実行可能な機能の事を指す．Web サービスは，

他の Web サービスと組み合わせる事によって，新しい機能を表す事が可能である．例えば,「ホテルを地名から検索する」という Web サービスと「ホテルを予約する」という Web サービスを組み合わせる事によって,「地名から検索されたホテルのどれかを予約する」という新しい機能を表す事が出来る．この様な，Web サービスを組み合わせて出来る機能を表すものを合成サービスと呼ぶ．この手法では，要求分析者が Web サービスの組み合わせを人手で考え，そこから作成された合成サービスを元に要求を考える事も可能である．また，この手法におけるメリットとして，ある Web サービスを元に要求分析者が要求を発見した時，その要求の元となった Web サービスは実行可能な機能を表すため，それをそのままソフトウェアに実装するプログラムとして使う事が可能となる事が挙げられる．

1.2 問題

ボトムアップアプローチにおける既存手法 [18] では，要求分析者に提示される Web サービスは，単純な機能を表す事が多いため，そこから他のソフトウェアに存在しない新しい要求を見つける事は困難である．

またそれらを組み合わせて出来る合成サービスの機能から，新しい要求を見つけるとすると，それらの Web サービスの組み合わせ方は膨大な数になる可能性があり，それらを要求分析者が人手で組み合わせ，その中から新しい特徴を見つける事は，大変な時間が掛かる作業である．

1.3 モチベーション

じゃらん [1]，楽天トラベル [2] など，既存のホテルに関するソフトウェアに存在する，例えば「ホテルを地名から検索する」,「ホテルの部屋情報を取得する」,「ホテルの部屋を予約する」,「現在の予約状況を取得する」などの機能を含む，19 個の機能を考える．後に評価で詳細を述べるが，これらの機能の組み合わせ方は全部で「5929 個」存在する．

この時，この 5929 個の合成サービスの中には,「ホテルを地名から検索する」,「ホテルの部屋情報を取得する」,「ホテルの部屋を予約する」,「現在の予約状況を取得する」という 4 つの機能を組み合わせて出来る「現在ホテルの部屋を予約している期間で，地名から検索されたホテルの部屋をどれか予約する」という機能が存在する．この機能は，現在予約し

ているホテルの部屋を別のホテルの部屋に変更したい人にとって，ニーズがある機能と言える．そして，この機能は，既存のホテルの関するソフトウェアにおいて，見当たらない機能である．この様に，既存の機能を組み合わせる事によって，他のソフトウェアに存在せず，かつニーズが存在する機能が生まれる可能性がある．

しかし，要求分析者が人手で「5929 個」の合成サービスをチェックし，この様な機能を探し出す事は大変な時間が掛かる作業と言える．本手法では，この事を解決する支援を行う手法を提案する．

1.4 提案

本手法では，ボトムアップアプローチにおいて，要求分析者が新しい要求を発見するための支援を行う手法を提案する．そのために本手法では，既存手法において提示される様な単体の Web サービスを組み合わせて出来る合成サービスから，(1) 他のソフトウェアが実装済みの機能，(2) 同じ機能を表す複数の組み合わせの中で，より悪い性能を表す組み合わせ，(3) ユーザに掛かる手間が大きい機能，をフィルタリングし，最後にそれらを要求分析者が 1 つの合成サービスを理解する過程に沿って，グループ化する．

本手法においてフィルタリングの対象としている (1)〜(3) の 3 つの機能もしくは組み合わせは，以下に示す理由によって，定義したものである．

(1) 本手法の目的は，他のソフトウェアに存在しない新しい特徴の発見を支援する事である．ここで，他のソフトウェアが実装済みの機能は，他のソフトウェアに存在しない新しい特徴とは言えないため，提示されない事が好ましい．よって，他のソフトウェアが実装済みの機能を表す合成サービスをフィルタリングの対象としている．

(2) 合成サービスは，単体の Web サービスと同じく，実行可能な機能を表す．そのため既存手法と同じく，本手法において要求分析者が獲得した要求の元となる合成サービスは，そのままソフトウェアに実装するプログラムとして使う事が可能である．ここで，同じ機能を表す合成サービスが複数ある場合，その中でより性能が良い合成サービスが提示される事が好ましい．そのために，その中でより性能が悪い合成サービスをフィルタリングの対象としている．

(3) ユーザに掛かる手間が大きい機能は，ユーザが必要としない可能性が高い機能と言える．例えば「ホテルを予約する」という機能は，ユー

ザが「ホテル ID」というデータを与える必要がある．この時，ユーザは，その「ホテル ID」がどの様なフォーマットで表現されるのか，そして自身の求めるホテルがそのフォーマットにおいてどの様に表現されるのか，という事を調べる必要があり，これはユーザにとって大きな手間であると言える．この時，既存のホテルに関するソフトェアにおいて「ホテルを予約する」という単体の機能は存在せず，主に「ホテルを検索する」という機能を組み合わせて使われており，その単体の機能はユーザが必要としない機能であると言える．ここでユーザが必要としない機能は，その機能に対するユーザの要求が存在しない事を表す．そのため，この様なユーザの手間が大きい機能が表す合成サービスをフィルタリングの対象としている．

また，本手法におけるグループ化の役割は，以下の通りである．フィルタリングの結果残った合成サービスは，未だに数が多い場合がある．本手法において，要求分析者は提示された合成サービスが「何が実現出来るのか？」という事を考える．そのために要求分析者は，それぞれの合成サービスの機能を理解する必要がある．しかし，この作業は，提示される合成サービスの数が多い場合，時間が掛かる作業となる．この様な合成サービスの機能への理解を支援する方法として，要求分析者が 1 つの合成サービスを理解する過程に沿ったグループ化を行う．グループ化の方法については，後の手法の説明にて述べる．

第2章　関連研究

既存研究において，現在まで，要求分析者の創造力を支援する様々な手法が提案されている．

「ユーザは，何を実現したいのか？」という観点から要求を考えるトップダウンアプローチとして，現在，以下の様な手法が提案されている．

M.A.Boden[17] は，創造力を支援する方法を，ExploratoryCreativity，CombinatorialCreativity，TransformationalCreativity という 3 つの種類に大分している．ExploratoryCreativity とは，自身の持つ思考のスタイルに沿って，新しいアイディアを探す方法を指す．CombinatorialCreativity とは，既存のアイディアを組み合わせて，新しいアイディアを表す組み合わせを作る方法を指す．TransformationalCreativity とは，既存の物事が持つ制限を見つけ，それを解消する新しいアイディアを考える方法を指す．

R.B.Svensson ら [15] は，これらの ExploratoryCreativity，CombinatorialCreativity，TransformationalCreativity それぞれにおける代表的な手法とブレインストーミングを比較し，どの方法が最も要求分析者の創造力を支援する事が出来たか，を調査している．調査の方法として，それぞれの方法に対して要求を考えるワークショップを開催し，そこで獲得された要求をそれぞれ評価する．評価の方法は，Sternberg[14] の創造力の定義に基づき，獲得された要求に対して「新しさ」と「必要さ」を 5 段階評価でアンケートを行うというものである．

T.Bhowmik らは [11]，これらの方法の中の CombinatorialCreativity に着目し，既存のアイディアを組み合わせる事を自動化する手法を提案している．この手法では，まず開発する対象のソフトウェアのユーザを類似度によってグループ分けする．次に，それぞれのグループに属するユーザが持つ過去の開発のドキュメントやコメントを取得し，そこに含まれる単語を全て抽出する．そして最後に，グループ同士の単語を 1 つずつ組み合わせ，単語のペアを作成する．これらの作業を自動で行い，要求分析者に，単語のペアをヒントとして提示する事によって，その創造力

を支援している．

一方で,「現時点で何が実現出来るのか？」という観点から要求を考えるボトムアップアプローチとして，現在，以下の様な手法が提案されている．

K.Zachos ら [20] は,「現時点で何が実現出来るのか？」という情報として Web サービスを提示する手法を提案している．この手法では，要求分析者は，開発する対象のソフトウェアへの要求を曖昧な文書として入力すると，その文書の意味に近い機能を持つ Web サービスの説明文が提示される．要求分析者は，それらの説明文を読み，最初に入力した曖昧な要求をより詳細なものに落とし込んだり，もしくはそれらを元に新しい要求を獲得する事が可能となる．

また，K.Zachos ら [21] は，この手法の発展として，最初に入力する曖昧な文書の意味の解析をより正確にし，同じドメインだけで無く，他のドメインの似た Web サービスも提示出来る様にした手法を提案している．これによって，要求分析者に提示される Web サービスの数が増え，要求を考えるための探索空間を広げる事が可能となる．

また，K.Zachos ら [19] は，これらの Web サービスを使う手法と要求獲得において頻繁に使われる手法の 1 つであるユースケースを使う手法を比較した結果，ユースケースを使う手法が獲得出来なかった要求を Web サービスを使う手法によって獲得出来た事を示している．

また，既存研究において，合成サービスを作成するための支援を行う様々な手法が提案されている．

V.Gabrel ら [24] は，合成サービスにおける Web サービスの組み合わせ方のルールを，線型計画法によって記述したものを提案している．Web サービスは，その Web サービスが必要とするインプットのデータの型とその Web サービスが返すアウトプットのデータの型の情報を持っており，それらの情報を元に合成が行われる．その合成における V.Gabrel ら [24] が提示しているルールの 1 つとして，合成サービスが必要とするインプットのデータはどこかから必ず与えられている必要がある，というものがある．Web サービスの実行にはインプットのデータの入力が必須であり，合成サービスの実行においても同様にそのインプットのデータを揃える必要がある．

R.Tang ら [30] は，Web サービスの組み合わせ方におけるパターンというものを定義し，それらを合成サービスの実行ログから取得する手法を提案している．Web サービスの組み合わせ方には，様々な合成サービス

で頻繁に使われる組み合わせ方というものが存在する．この手法では，これらの組み合わせ方を実行ログを使う事によって識別し，Web サービスの合成におけるパターンとして導出している．実行ログとは，合成サービスに含まれる個々の Web サービスの実行された順番を記録したものである．合成サービスは，その振る舞いを定義した BPEL ファイルなどを IBM WebSphere Process Server[7] などのサーバにデプロイする事によって，Web 上に公開する事が可能である．この時，様々なユーザによって合成サービスは使われ，その実行ログはサーバ上に保存されているため，それらの実行ログをサーバから取得する事が可能である．

ここで，これらの合成サービスを作成するための支援を行う手法では，Web サービス同士が組み合わせる事が可能であるか，という情報が表現された SDG[22][23] というグラフが既に存在している事が前提となっている．

SDG とは，Q.Liang ら [22] が提案した Web サービス同士が組み合わせる事が可能であるか，という情報を表現したグラフである．SDG では，それぞれの Web サービスが持つインプットのデータの型とアウトプットのデータの型の情報を調べ，ある Web サービスが必要とするインプットのデータの型を *input*，別の Web サービスが返すアウトプットのデータの型を *output* とした時，*output* と *input* が一致しており，*output* を *input* として使う事が可能であるものを，Web サービス同士が組み合わせる事が可能であるという依存関係として表現している．

Z.Gu ら [23] は，Q.Liang ら [22] が提案した SDG を派生させた SDG+ というグラフを提案している．従来の SDG では，データの型における包含関係を考慮した依存関係，似た様なデータの型を考慮した依存関係，特定の Web サービスのみとの依存関係、の 3 つが表現されておらず，SDG+ はこれらの情報を組み込んだ新しい SDG となっている．包含関係とは，一方のデータがもう一方のデータを包含する関係の事を指す．例えば，「ホテル情報」というデータが「ホテル ID」「ホテル名」「住所」というデータを包含する時，これらのデータは包含関係を持っており，「ホテル情報」という上位のデータから「ホテル ID」という下位のデータを取得する事が可能である．似た様なデータの型による依存関係とは，例えば「住所」と「地域」の様な，意味は似ているがデータの型が違うもの同士の依存関係の事を指す．特定の Web サービスとの依存関係とは，Web サービスにはデータの型に関わらず特定の Web サービスとのみ依存関係があるものが存在し，その Web サービスとの依存関係の事を指す．例えば，「ID」

というデータの型は同じ型を持つデータでも違うフォーマットを持つ場合がある．この場合，同じフォーマットを持つデータを扱う特定の Web サービスとのみ依存関係が生じる．

第3章　手法

図 3.1 は，本研究が提案する手法のオーバービューを示す．

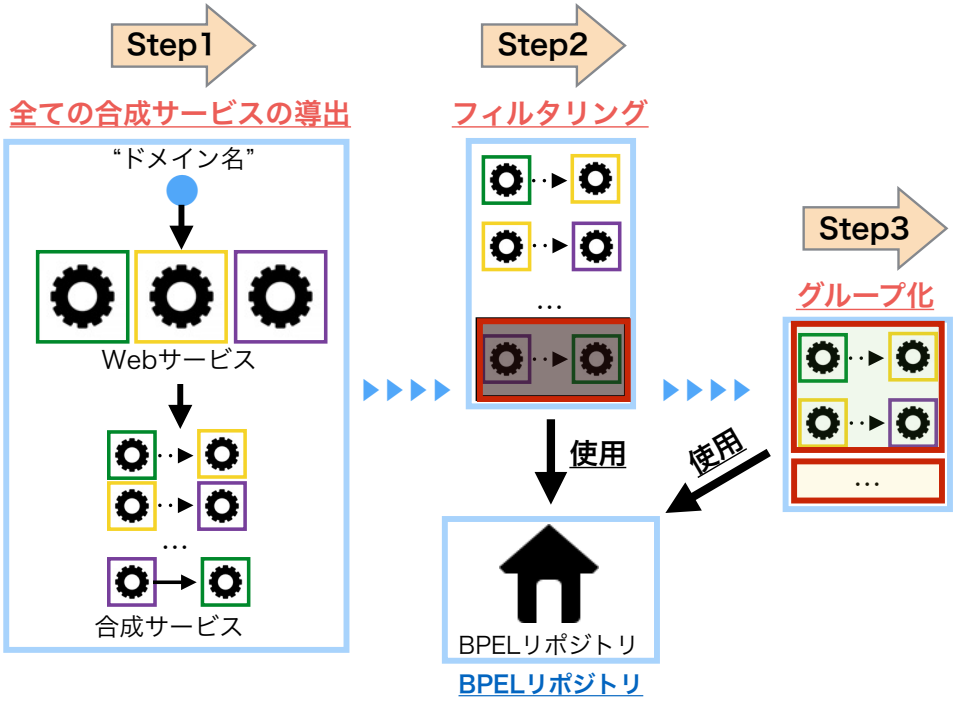


図 3.1: オーバービュー

本手法は，全ての合成サービスの導出，フィルタリング，グループ化，という 3 つのステップによって構成される．

ステップ 1 の全ての合成サービスの導出の段階では，開発する対象のソフトウェアのドメインの属する Web サービスの全ての組み合わせ方を網羅し，それぞれに対する合成サービスを作成する．開発する対象のソフトウェアのドメインに属する Web サービスを取得するために，UDDI[6] や ProgramableWeb[5] などの Web サービスが登録されているリポジトリを使う．この時，それぞれの Web サービス同士が組み合わせる事が可能

であるかどうか，という情報は SDG[22] というグラフによって表現されており，この SDG を元に全ての組み合わせ方を網羅し，それぞれに対する合成サービスを作成する．

ステップ 2 のフィルタリングの段階では，ステップ 1 で導出された合成サービスの集合から，他のソフトウェアが実装済みの機能，同じ機能を表す組み合わせが複数ある時に，より性能が悪い組み合わせ，ユーザに掛かる手間が大きい機能をフィルタリングする．この時，より性能が悪い組み合わせとして，無駄な処理を含む組み合わせをフィルタリングし，ユーザに掛かる手間が大きい機能として，ユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする．

ステップ 3 のグループ化の段階では，ステップ 2 でフィルタリングされた結果残った合成サービスの集合に対して，要求分析者がそれぞれの合成サービスの機能を理解する時に掛かる手間を減らす様に，グループ化を行う．

3.1 全ての合成サービスの導出 (ステップ 1)

このステップは，開発する対象のソフトウェアのドメインに属する全ての Web サービスの集合を取得し，それらの全ての組み合わせ方を網羅し，それぞれの組み合わせ方に対して合成サービスを導出するステップである．

このステップは，開発する対象のソフトウェアのドメインに属する Web サービスの集合の取得，そして，それらの全ての組み合わせ方の導出，という 2 つの工程によって構成される．

まず開発する対象のソフトウェアのドメイン情報から，そのドメインに属する Web サービスの集合を取得する．現在 Web サービスが登録されているリポジトリとして，UDDI[6] や ProgramableWeb[5] など幾つかのリポジトリが存在する．これらのリポジトリでは，ドメイン情報を入力として与える事によって，そのドメインに属する Web サービスを検索する事が出来る．そのため，その検索にヒットする全ての Web サービスの集合を取得する．

次に，それらの全ての組み合わせ方を導出する．ここで，Web サービスはインプットのデータの型とアウトプットのデータの型というものの情報を持っている．Web サービスを利用する方法は，Web サービスが必要とするインプットのデータを与え，それに対して Web サービスが返す

アウトプットのデータを受け取る，というものである．この時，組み合わせる事が可能な Web サービスとは，ある Web サービスが必要とするインプットのデータの型を *input*，別の Web サービスが返すアウトプットのデータの型を *output* とした時，*output* と *input* が一致しており，*output* を *input* として使う事が可能なものを指す．例えば,「ホテルを地名から検索する」という Web サービスが返す「ホテル ID」という型を持つアウトプットのデータを,「ホテルを予約する」という Web サービスが必要とする「ホテル ID」という型を持つインプットのデータとして使う事が可能な時，これらの Web サービスは組み合わせる事が可能であると言える．

この様な，Web サービス同士が組み合わせる事が可能かどうか，という情報を表現するものとして，SDG(Service Dependency Graph)[22][23] というグラフが存在する．

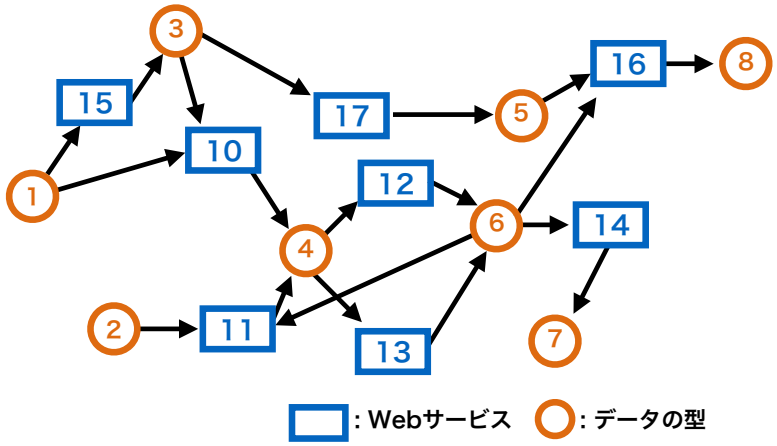


図 3.2: Service Dependency Gprah

SDG の例を，図 3.2 に示す．四角で囲まれた数字が示す頂点が Web サービスを表し，丸で囲まれた数字が示す頂点がデータの型を表す．データの型とは，データそのものでは無く，それらをまとめた総称の事を指す．例えば,「1111」や「2222」という 4 桁の数字の羅列を総称したものとして,「ホテル ID」というデータの型を指す．頂点を結ぶ矢印は，ある Web サービスが矢印の先に示す型を持つデータをアウトプット出来るという依存関係と，ある Web サービスが矢印の元に示す型を持つデータがインプットとして必要となるという依存関係を表す．

SDG は，G=(X，U) という有向グラフによって表現される．X は，S という Web サービスを表す頂点の集合と D というデータの型を表す頂点

の集合によって表現される．$i \in S$ の時，$s(i)$ は i という頂点によって表される Web サービスを指す．$i \in D$ の時，$d(i)$ は i という頂点によって表されるデータの型を指す．また，U は，2 種類の依存関係を表す有向辺の集合によって表現される．1 つが $i \in S$，$j \in D$ を満たす i，j に関して，$s(i)$ が $d(j)$ をアウトプットする，という依存関係を表す有向辺 (i, j) であり，もう 1 つが $i \in D$，$j \in S$ を満たす i，j に関して，$d(i)$ を $s(j)$ がインプットする，という依存関係を表す有向辺 (j, i) である．SDG において，$i \in S$ かつ $j \in S$，もしくは $i \in D$ かつ $j \in D$ を満たす i，j に関して，(i, j) が成り立つ事は無い [22]．ここで $i \in S$，$j \in D$，$k \in S$ を満たす i，j，k に関して，$s(i)$ が $d(j)$ をアウトプットとして導出し，また $d(j)$ を $s(k)$ がインプットとして使う事が可能な場合，$s(i)$ と $s(k)$ は組み合わせる事が可能であり，その組み合わせは $\{(i, j), (j, k)\}$ という依存関係によって SDG 上に表現される．例えば，図 3.2 の SDG 上にある $\{(15, 3), (3, 10)\}$ という依存関係は，$s(15)$ と $s(10)$ を $d(3)$ を介して組み合わせる事が可能である事を表現している．

この様に Web サービスの全ての組み合わせ方は，SDG 上に依存関係として全て表現されており，Web サービスの全ての組み合わせ方を網羅するためには，この SDG 上の全てのサブグラフを網羅すれば良い．

この全てのサブグラフを網羅するには，SDG 上のそれぞれの Web サービスに対して，それらが必要とするインプットのデータの型に対して依存関係がある Web サービスを次々と組み合わせていき，その過程で導出される全ての組み合わせを網羅すれば良い．この時，それぞれの Web サービスに対して別の Web サービスを組み合わせる方法は，以下のアルゴリズム 1 によって表現する事が出来る．このアルゴリズムでは，SDG 上に存在する全ての Web サービスの集合を All WebServices，それぞれの Web サービスを WebService，Web サービスが必要とするインプットのデータが複数ある時，それぞれのインプットのデータの型に対して依存関係がある Web サービスと組み合わせた合成サービスの集合を PastCompositeWebServices，Web サービスが必要とするインプットのデータが複数ある時，その中の 1 つのインプットのデータの型に対して依存関係がある Web サービスと組み合わせた合成サービスを NowCompositeWebService として定義する．

アルゴリズム 1 では，まず SDG 上に存在する All WebServices を走査し，それぞれの WebService に対して，その WebService が必要とするインプットのデータの型に対して依存関係がある WebService2 を探す．そして，その様な WebService2 を発見した場合，その Web サービスの組み

Algorithm 1 Web サービスが必要とするインプットのデータの型に対して依存関係がある Web サービスを組み合わせる方法

```
for WebService in All WebServices do
  if Inputs of WebService.Size == 1 then
    for WebService2 in All WebServices do
      if Output of WebService2.contains(Input) then
        WebService combine WebService2 //組み合わせ方のパターン (1)
      end if
    end for
  else {Inputs of WebService.Size > 1}
    PastCompositeWebServices ⇐ null
    for Input in Inputs of WebService do
      for WebService2 in All WebServices do
        if Output of WebService2.contains(Input) then
          WebService combine WebService2 //組み合わせ方のパターン (2)
          NowCompositeWebService ⇐ WebService and WebService2
          if Input's Count is more than 1 then
            for PastCompositeWebService in PastCompositeWebServices do
              PastCompositeWebService combine NowCompositeWebService //組み合わせ方のパターン (2)
              PastCompositeWebServices.Add(PastCompositeWebService and NowCompositeWebService)
            end for
          end if
          PastCompositeWebServices.Add(NowCompositeWebService)
        end if
      end for
    end for
  end if
end for
```

合わせ方には，組み合わせる対象の Web サービスが必要とするインプットのデータの型が 1 つしか無い場合と複数ある場合の 2 つのパターンが存在する．1 つしか無い場合は，その 1 つのインプットのデータの型に対して依存関係がある WebService2 をシンプルに組み合わせれば良い．一方で複数ある場合は，まずそれぞれのインプットのデータの型に依存関係がある WebService2 を組み合わせ，それぞれ NowCompositeWebService を作成する，それらは PastCompositeWebServices として保持する．そして，過去に作成した PastCompositeWebServices と現在作成している NowCompositeWebService を組み合わせたものを導出する，という方法を使う．

このアルゴリズムを，新しく組み合わせた Web サービスに対して依存関係がある Web サービスが存在しなくなるまで繰り返し，その過程で導出した全ての組み合わせを，SDG 上の全てのサブグラフとして取得する．

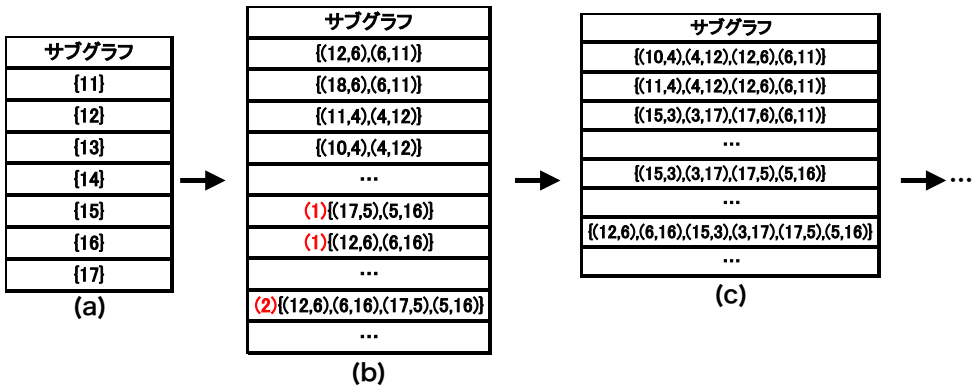


図 3.3: SDG 上の全てのサブグラフの導出過程

アルゴリズム 1 を，図 3.2 の SDG を使って説明する．SDG 上の全てのサブグラフを導出する過程を，図 3.3 に示す．図 3.3(a)，(b)，(c) は，SDG 上の全てのサブグラフを導出する過程を，幾つかのステップに分けた時に，それぞれのステップにおいて導出されるサブグラフを表す．

まず，図 3.3(a) は，SDG 上に存在するそれぞれの単体の Web サービスを表す．

そして，図 3.3(b) は，図 3.3(a) のそれぞれの Web サービスが必要とするインプットのデータの型に対して依存関係がある Web サービスを組み合わせたものを表す．この時，必要とするインプットのデータの型が 1 つしか無い場合と複数ある場合があり，それぞれ組み合わせ方のパターン

がある．

1つしか無い場合は，その1つのインプットのデータの型に依存関係がある Web サービスをシンプルに組み合わせたものを導出する．例えば，図 3.3(a) の $s(11)$ は，$d(6)$ というインプットのデータの型のみを必要とする．この場合，この $d(6)$ というインプットのデータの型に対して依存関係がある $s(12)$ をシンプルに組み合わせる事によって，図 3.3(b) の {(12, 6), (6, 11)} という組み合わせを導出する．

一方で複数ある場合は，まずそれぞれのインプットのデータの型に依存関係がある Web サービスを組み合わせたものを導出し，次に，その組み合わせたもの同士を組み合わせたものを導出する．例えば，図 3.3(a) の $s(16)$ は，$d(5)$ と $d(6)$ という 2 つのインプットのデータの型を必要とする．この場合，まず一方の $d(5)$ というインプットのデータの型に対して依存関係がある $s(17)$ を組み合わせ，図 3.3(b)(1) に示す {(17, 5), (5, 16)} を導出し，次に $d(6)$ というインプットのデータの型に対して依存関係がある $s(12)$ を組み合わせ，図 3.3(b)(1) に示す {(12, 6), (6, 16)} を導出する．そして，その後にその組み合わせたもの同士を組み合わせた図 3.3(b)(2) に示す {(12, 6), (6, 16), (17, 5), (5, 16)} を導出する．

図 3.3(c) は，図 3.3(b) で新しく組み合わせた Web サービスに対して，今までと同様に，そのインプットのデータの型に依存関係がある Web サービスを組み合わせたものを表す．例えば，図 3.3(b) の {(12, 6), (6, 11)} において新しく組み合わせた Web サービスは $s(12)$ である．この $s(12)$ が必要とするインプットのデータの型は $d(4)$ であり，それに対して依存関係がある $s(10)$ という Web サービスを組み合わせると，図 3.3(c) の {(10, 4), (4, 12), (12, 6), (6, 11)} が導出される．

ここで，インプットのデータの型に対して依存関係のある Web サービスを次々と組み合わせていくと，再度同じ Web サービスを組み合わせる事になる場合がある．例えば，図 3.2 の SDG であれば，$s(12)$ が必要とするインプットのデータの型に対して依存関係のある Web サービスを次々と組み合わせていくと，{(12, 6), (6, 11), (11, 4), (4, 12)} が導出され，ここでは再度 $s(12)$ を新しく組み合わせる事になっている．ここに以前組み合わせた Web サービスを再度組み合わせていくと，同じ処理が永遠と繰り返されてしまうため，本手法では，この様な場合には以前組み合わせた事があるものとは別の依存関係がある Web サービスを組み合わせる事とする．例えば，先述した {(12, 6), (6, 11), (11, 4), (4, 12)} であれば，$s(12)$ が必要とするインプットのデータの型である $d(4)$ に対して依存関係があ

る Web サービスには，$s(11)$ と $s(10)$ の 2 つが存在する．この時，$s(11)$ は以前組み合わせた事があるため，別の依存関係がある $s(10)$ を組み合わせる．

最後に合成サービスにおけるルールとして，個々の Web サービスが必要とするインプットのデータは必ずどこかから与えられている必要がある [24] というものがあり．本手法もこのルールを元に合成サービスを導出する．そのためこのステップで導出した合成サービスに関しても，個々の Web サービスが必要とするインプットのデータがどこから与えられるのか，という事を定義する必要がある．この時，合成サービスに対するインプットのデータの与え方は 2 種類あり，Web サービスが返すアウトプットのデータを使う方法とユーザが与えるデータを使う方法が存在する．ここで SDG では，ユーザが与えるデータを使うという依存関係は表現されておらず，このステップで導出した全ての合成サービスは，他の Web サービスが返すアウトプットのデータを使うという依存関係のみしか表現されていない．そこで本手法では，これらの合成サービスに対してユーザが与えるインプットのデータとして，Web サービスが返すアウトプットのデータを使うと表現されているデータ以外の全てのデータとする．例えば，図 3.2 の SDG の $\{(12, 6), (6, 11)\}$ という合成サービスであれば，$s(11)$ はインプットとして $d(2)$ と $d(6)$ という 2 つの型を持つデータが必要となる．この時，$d(6)$ に関しては $s(12)$ が返すアウトプットのデータを使う事が表現されている一方で，$d(2)$ に関しては何も表現されていない．この時，$\{(12, 6), (6, 11)\}$ という合成サービスに対して，ユーザは $d(2)$ という型を持つデータを与える必要がある事とする．

3.2 フィルタリング (ステップ 2)

このステップは，ステップ 1 で導出された全ての合成サービスから，他のソフトウェアによって実装済みの機能，無駄な処理を含む組み合わせ，そしてユーザによる作成が困難なデータの入力が必要な機能をフィルタリングするステップである．

3.2.1 他のソフトウェアが実装済みの機能のフィルタリング

ここでは，他のソフトウェアが実装済みの機能を持つ合成サービスを識別し，それらをフィルタリングする方法について説明する．

本手法の目的は，他のソフトウェアに存在しない新しい特徴を発見するための支援を行う事である．しかし，他のソフトウェアが実装済みの機能とは，本手法の目的を満たしておらず，フィルタリングされる事が好ましい機能と言える．

本手法における他のソフトウェアが実装済みの機能を表す合成サービスをフィルタリングする方法は，以下のアルゴリズム 2 によって表現する事が出来る．このアルゴリズムでは，ステップ 1 で導出された全ての合成サービスを All CompositeWebServices，それぞれの合成サービスを CompositeWebService，BPEL リポジトリに存在する全ての BPEL ファイルを All BPELFiles，BPEL ファイルを BPELFile として定義する．

Algorithm 2 他のソフトウェアが実装済みの機能を持つ合成サービスのフィルタリング方法

```
for CompositeWebService in All CompositeWebServices do
  Get(All BPELFiles)
  for BPELFile in All BPELFiles do
    if BPELFile == CompositeWebService then
      Filtering(CompositeWebService) // フィルタリング
    end if
  end for
  if CompositeWebService.Size == 1 then
    Filtering(CompositeWebService) //フィルタリング
  end if
end for
```

アルゴリズム 2 では，まず BPEL リポジトリに存在する全ての BPEL ファイル All BPELFiles を取得する．

BPEL ファイルとは，BPEL(Business Process Execution Language) という合成サービスを表現する事が可能な言語によって記述されたファイルの事を指す．BPEL ファイルは，IBMWebSphereProcessServer[7] などの BPEL の実行エンジンにデプロイする事によって，1 つのソフトウェアとして Web 上に公開する事が可能である．そのため，BPEL ファイルは，他のソフトウェアが実装済みの機能を表していると言える．BPEL では，Web サービスに関する操作を示す要素が幾つか定義されており，それらの要素を使う事によって，合成サービスを表現する事が出来る．

図 3.4 に，BPEL ファイルの例を示す．この BPEL ファイルは,「ホテルを検索する」という Web サービスと「ホテルを予約する」という Web サービスを組み合わせた合成サービスを表現している．BPEL ファイル

```
…
<sequence>
…

    <invoke operation="ホテルを検索する" inputVariable="InputData1"
outputVariable="OutputData1"></invoke>

    <assign>
        <copy>
            <from>$OutputData1.ホテルID</from>
            <to>$InputData2.ホテルID</to>
        </copy>
    </assign>

    <invoke operation="ホテルを予約する" inputVariable="InputData2"
outputVariable="OutputData2"></invoke>

…
</sequence>
…
```

図 3.4: BPEL ファイルの例

では，この 2 つの Web サービスの依存関係を invoke 要素と assign 要素によって表現する事が出来る．

BPEL における invoke 要素では，どの Web サービスを呼び出すか，という事を表現する事が出来る．invoke 要素は，operation 属性，inputVariable 属性，outputVariable 属性という属性を持っており，oeration 属性によって，どの Web サービスを呼び出すか，という事を表現する事が出来る．また inputVariable 属性では，その Web サービスが必要とするインプットのデータの型，outputVariable 属性では，その Web サービスが返すアウトプットのデータの型を表現する事が出来る．図 3.4 の BPEL ファイルの例では，2 つの invoke 要素を使い，それぞれ operation 属性で定義された「ホテルを検索する」と「ホテルを予約する」という Web サービスを呼び出している．

また，BPEL における assign 要素によって，どのデータをどのデータとして使うか，という事を表現する事が出来る．assgin 要素は，from 要素と to 要素という子要素を持っており，これらの要素によって，from 要素によって定義されたデータを to 要素によって定義されたデータとして使う，という事を表現する事が出来る．図 3.4 の BPEL ファイルの例では，assign 要素によって OutputData1 の「ホテル ID」という型を持つデータを，InputData2 の「ホテル ID」という型を持つデータとして使う，という事を表現している．この時，OutputData1 とは,「ホテルを検索する」を呼び出している invoke 要素の outputVariable 属性によって定義された「ホテルを検索する」が返すアウトプットのデータの型である．また InputData2 は,「ホテルを予約する」を呼び出してる invoke 要素の

inputVariable 属性によって定義された「ホテルを予約する」が必要とするインプットのデータの型である．よって，この BPEL ファイルが表す一連の流れは,「ホテルを検索する」が返すアウトプットの「ホテル ID」を「ホテルを予約する」が必要とする「ホテル ID」として使い,「検索されたホテルのどれかを予約する」という合成サービスを作成するという事を表現している事が分かる．この様にして，BPEL ファイルは，合成サービスを表現する事が出来る．

BPEL リポジトリとは，過去に様々な開発者によって開発された BPEL ファイルが登録されているリポジトリの事を指す．BPEL ファイルは様々な開発者によって開発され，開発した BPEL ファイル自体を公開したいと思う開発者は，それらを様々なオープンソースリポジトリに登録する．例えば，Github[4] というオープンソースリポジトリにおいて BPEL ファイルを検索すると，215238 個の BPEL ファイルを見つける事が出来る．この様な BPEL ファイルが登録されているオープンソースリポジトリの事を，本手法では BPEL リポジトリと呼ぶ．

この様な BPEL リポジトリに存在する全ての BPEL ファイル All BPELFiles を使い，ステップ 1 で導出された合成サービス All CompositeWebServices の中に，All BPELFiles に含まれるそれぞれの BPELFile と同じ機能を表すものが存在すれば，その対象の CompositeWebService をフィルタリングする．

また，その対象の CompositeWebService が単体の Web サービスを表す場合，これも別でフィルタリングする．単体の Web サービスが表す機能は，その Web サービス自体という既存のソフトウェアが実装済みの機能と言える．そのため，単体の Web サービスを表す合成サービスも同様にして，他のソフトウェアが実装済みの機能としてフィルタリングする．

3.2.2 無駄な処理を含む組み合わせのフィルタリング

ここでは，同じ機能を表す複数の組み合わせがあった時に，より悪い性能を持つ組み合わせとして，無駄な処理を含む組み合わせが表す合成サービスを識別し，それらをフィルタリングする方法について説明する．

合成サービスはそれ自体が実行可能な機能であるため，本手法において要求分析者が獲得した要求の元となる合成サービスは，そのままソフトウェアに実装するプログラムとして使う事が可能である．この時，同じ機能を表す組み合わせが複数あった時，要求分析者にとっては，その

中で最も良い性能を持つ組み合わせのみが提示される事が好ましい．そのため，それらの組み合わせを残すために，その中でより悪い性能を持つ組み合わせが表す合成サービスをフィルタリングする．

合成サービスにおける性能は，QoS(Quality of Service) によって表現される．QoS とは，Web サービスの性能を数値で表したものであり，その QoS の要素として代表的なものには，応答時間，スループットなどが挙げられる．合成サービスにおける QoS は，その合成サービスに含まれる個々の Web サービスが持つ QoS の足し算によって表現される．例えば，ある 3 つの Web サービスを組み合わせる事によって出来る合成サービスにおいて，それぞれの Web サービスの応答時間が「3 秒」「1 秒」「2 秒」であるとすると，その合成サービス全体の応答時間は，それぞれの足し算の結果である「6 秒」となる．この時，QoS には良し悪しがあり，例えば応答時間であれば，早ければ早いほど良い QoS を表し，遅ければ遅いほど悪い QoS を表す．本手法では，この様な合成サービスが持つ性能を表す QoS に着目し，より悪い QoS を持つ合成サービスを，より悪い性能を持つ合成サービスとして識別する．

ここで，複数の合成サービスが同じ機能を持つケースとして，合成サービスが無駄な処理を含むケースがある．無駄な処理とは，本手法では，その処理があっても無くても合成サービス全体の機能に影響が無い処理と定義する．そのため，無駄な処理を含む合成サービスは，そこから無駄な処理を除いた機能と同じ機能を持つ事となる．また，無駄な処理を含む合成サービス全体の QoS は，無駄な処理に含まれる Web サービスが持つ QoS の分だけ全体の QoS が悪くなると言える．よって，無駄な処理を含む合成サービスとは，同じ機能を持つ合成サービスが複数存在し，その中で，より悪い QoS を持つ合成サービスであると言える．

ステップ 1 で導出された合成サービスは，Web サービスの全ての組み合わせ方を網羅する事によって導出しており，その中には無駄な処理を含む組み合わせも含まれてしまう．そのため，この様な無駄な処理を含む組み合わせをフィルタリングする．

本手法では，無駄な処理として，無駄なデータの循環というものを定義し，それらを識別する．

本手法において，データの循環とは，合成サービスに含まれる各 Web サービスにおいて，以前に導出されたデータと同じ型を持つデータが再度導出された時，そのデータを元に起こる循環の事を指す．

図 3.5 に，SDG 上で表現されるデータの循環の例を示す．この SDG 上

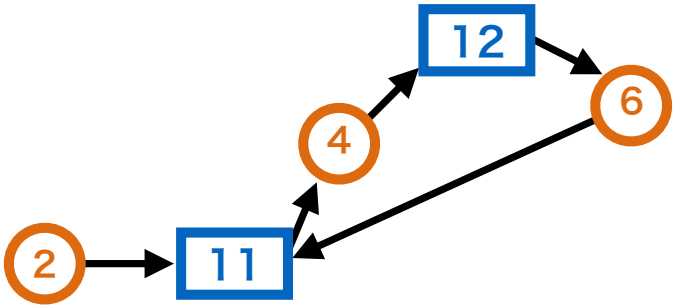


図 3.5: データの循環

の $\{(12, 6), (6, 11), (11, 4), (4, 12)\}$ という合成サービスを考えると，まず最初の $s(12)$ はユーザがインプットとして $d(4)$ という型を持つデータを与える必要がある．この時，ユーザは $d(4)$ という型を持つデータを導出したと言える．次に，そのまま各頂点を見ていくと，同じく $d(4)$ という型を持つデータを $s(11)$ が再度導出している事が分かる．この時，この合成サービスを SDG 上で見ると，$d(4)$ というデータの型を元にして各頂点を結ぶ矢印が循環している事が分かる．この様な循環の事を本手法ではデータの循環と呼ぶ．

ここでデータの循環には，合成サービス全体の機能に対して，無駄なデータの循環と無駄では無いデータの循環という 2 種類が存在する．無駄なデータの循環とは，合成サービス全体の機能に影響が無いデータの循環の事を指し，無駄では無いデータの循環とは，合成サービス全体の機能に影響を与えるデータの循環の事を指す．そのため，本手法では，無駄な処理を表すものとして，無駄なデータの循環を識別する．

無駄なデータの循環は，その循環の元となるデータが常に同じであり，かつその循環に含まれる Web サービスが現実世界に与える効果を持たない場合に起こる．Web サービスが現実世界に与える効果とは，その Web サービスが現実世界の状態を変化させる事を指す．例えば，「クレジットカードで支払いを行う」という Web サービスが現実世界に与える効果は，「クレジットカードの利用金額を，支払い金額分増やす」という現実世界におけるクレジットカードの状態を変化させる事である．

無駄なデータの循環の例を，図 3.6 に示す．この例に示す合成サービスは，「預金額を取得する」「円をドルに変換する」「ドルを円に変換する」という 3 つの Web サービスを組み合わせたものである．ここで，この合成サービスでは「円によって表された預金額」という型を持つデータを

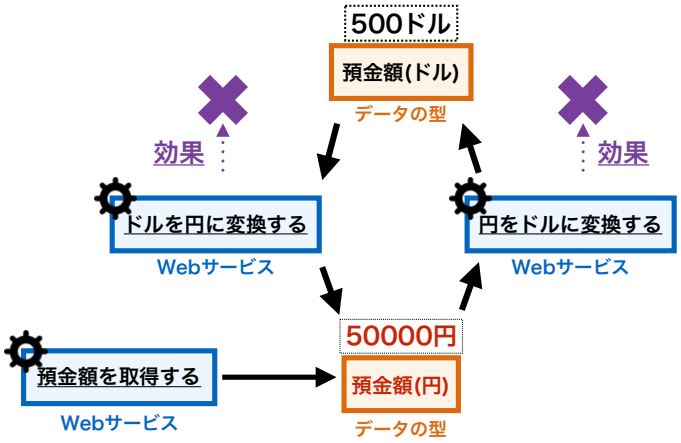


図 3.6: 意味を持たないデータの循環を含む合成サービス

元にデータの循環が起こっている事が分かる．

この合成サービスにおいて，データの循環の元となる「円によって表された預金額」は常に同じ「50000 円」を表す．これはそのデータが「10000 円」であっても「100000 円」であっても，常に同じ「10000 円」「100000 円」というデータとなる．そして，このデータの循環に含まれる「円をドルに変換する」と「ドルを円に変換する」という Web サービスは，現実世界に与える効果を持っていない．

この時,「預金額を取得する」「円をドルに変換する」「ドルを円に変換する」という合成サービス全体の機能によって実現出来る事は,「50000 円」というデータを獲得する事である．一方で，そこからデータの循環を除いた「預金額を取得する」という機能によって実現出来る事は，同様に「50000 円」というデータを獲得する事である．よって，この様なデータの循環は，合成サービス全体の機能に影響が無い処理と言える．

一方で，無駄では無いデータの循環は，データの循環に含まれる Web サービスが現実世界に与える効果を持つ，もしくはデータの循環の元となるデータが常に同じとは限らない場合に起こる．

データの循環に含まれる Web サービスが現実世界に与える効果を持つ場合に起こる無駄では無いデータの循環の例を，図 3.7 に示す．この例に示す合成サービスは,「ソースコードをテストする」,「ソースコードをアップロードする」という 2 つの Web サービスを組み合わせたものである．ここで，この合成サービスでは「ソースコード」という型を持つデータを元にデータの循環が起こっている事が分かる．

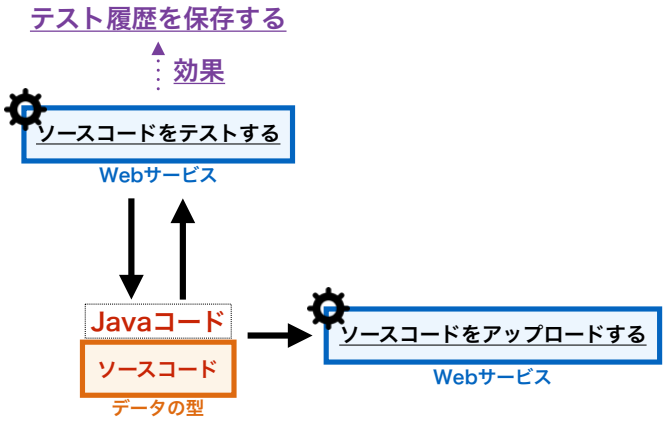


図 3.7: データの循環に含まれる Web サービスが現実世界に与える効果を持つ場合の例

この合成サービスにおいて，データの循環に含まれる「ソースコードをテストする」という Web サービスは「テスト履歴を保存する」という現実世界に与える効果を持っている．この時，この合成サービスから，このデータの循環を除いてしまうと，合成サービス全体の機能から「テスト履歴を保存する」という機能が除かれてしまう事になる．よって，この様なデータの循環は，合成サービス全体の機能に影響を与える処理と言える．

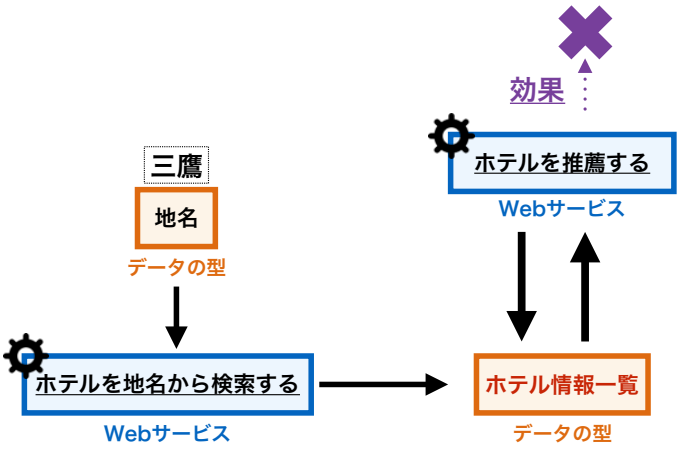


図 3.8: データの循環の元となるデータが常に同じとは限らない場合の例

次に，データの循環の元となるデータが常に同じとは限らない場合に起こる無駄では無いデータの循環の例を，図 3.8 に示す．この例に示す合成サービスは,「ホテルを地名から検索する」「ホテルを推薦する」という 2 つの Web サービスを組み合わせたものである．ここで，この合成サービスでは「ホテル情報一覧」というデータの型を元にデータの循環が起こっている事が分かる．

この合成サービスにおいて,「ホテルを地名から検索する」が返す「ホテル情報一覧」は「三鷹に存在するホテル情報一覧」を表し,「ホテルを推薦する」が返す「ホテル情報一覧」は，その中から選出された「ユーザに合うホテル情報一覧」を表す．この時,「三鷹に存在するホテル情報一覧」と「ユーザに合うホテル情報一覧」は，そこに含まれるホテル情報が違う可能性があり，これらのデータは常に同じとは限らない．この時，この合成サービスから，このデータの循環を除いてしまうと，合成サービス全体の機能から「ユーザに合うホテル情報一覧」を選出するという機能が除かれてしまう事になる．よって，この様なデータの循環は，合成サービス全体の機能に影響を与える処理と言える．

この様にデータの循環には，無駄なデータの循環と無駄では無いデータの循環が存在する．本手法では無駄なデータの循環を含む組み合わせのみを，無駄な処理を含む組み合わせとしてフィルタリングする．

本手法における無駄な処理を含む合成サービスをフィルタリングする方法は，以下のアルゴリズム 3 によって表現する事が出来る．このアルゴリズムでは，ステップ 1 で導出された全ての合成サービスを All CompositeWebServices，それぞれの合成サービスを CompositeWebService，合成サービスに含まれる各 Web サービスを WebService，合成サービスに含まれる各 Web サービスにおいて導出されるデータの集合を PastData，データの循環を DataCycle として定義する．

アルゴリズム 3 では，まずステップ 1 で導出された全ての合成サービス All CompositeWebServices を走査し，それぞれの CompositeWebService に対して，データの循環 DataCycle が含まれているか，という事をチェックする．このデータの循環の識別の方法は，対象の CompositeWebService に含まれる各 WebService において，以前に導出されたデータと同じ型を持つデータが再度導出されるかどうか，によって識別する．そして，もし DataCycle が含まれているならば，その DataCycle が無駄なデータの循環なのか，という事をチェックする．そして，もしその DataCycle が無駄なデータの循環であれば，その DataCycle を含む対象の CompositeWeb-

Algorithm 3 無駄な処理を含む合成サービスのフィルタリング方法

```
for CompositeWebService in All CompositeWebServices do
  CompositeWebServiceHasDataCycle ⇐ false
  for WebService in CompositeWebService do
    PastData.Add(Inputs of WebService)
    if WebService is Last of CompositeWebSerivce then
      if PastData.Contains(Output of WebService) then
        CompositeWebServiceHasDataCycle ⇐ true
        if DataCycle is not meaningful then
          Filtering(CompositeWebService) //フィルタリング
        end if
      end if
    else {WebService is not Last of CompositeWebSerivce}
      if PastData.Contains(Output of WebService for Next WebService)
      then
        CompositeWebServiceHasDataCycle ⇐ true
        if DataCycle is not meaningful then
          Filtering(CompositeWebService) //フィルタリング
        end if
      end if
    end if
    PastData.Add(Output of WebService)
  end for
end for
```

Service をフィルタリングする．ここで，対象の DataCycle が無駄なデータの循環なのか，を識別する方法について，以下に説明する．

無駄なデータの循環を識別するために必要となる情報は，データの循環に含まれる Web サービスが現実世界に与える効果を持つか，そしてデータの循環の元となるデータが常に同じとなるか，である．

まず，データの循環に含まれる Web サービスが現実世界に与える効果を持つか，という事を識別するために，本手法では，WSDL ファイルという WSDL(Web Service Description Language) によって記述されたファイルを使用する．WSDL とは，Web サービスの振る舞いを定義する事が可能である言語である．WSDL では，Web サービスが必要とするインプットのデータの型，Web サービスが返すアウトプットのデータの型の情報に加え，Web サービスが現実世界に与える効果の情報を記述する事が出来る．この情報を使い，データの循環に含まれる Web サービスが現実世界に与える効果を持つか，という事を識別する．

次に，データの循環の元となるデータが常に同じであるか，という事を識別するために，本手法では，データの循環を含む合成サービスを逐次実行し，実際にそのデータの循環の元となるデータを観測し，それが常に同じになるかどうか，を調べる．

ここで Web サービスを実行するためには，その Web サービスにおいてユーザが入力する必要があるインプットのデータを生成する必要がある．現在，Web サービスのテストケースとなるデータを自動生成する手法が幾つか提案されており [25][26]，これらの手法を使う事によって，それらのデータを自動で生成する．

Web サービスの実行には，その処理に要する時間が掛かるというコストが存在する．もしデータの循環を含む合成サービスの数が膨大になる場合，そこに含まれる Web サービスを実行する処理全てに掛かる時間も膨大な時間となる．そのため，Web サービスを実行する数は出来る限り少なくする事が求められる．本手法では，この事を解決するため，Web サービスを実行する数を必要最低限に抑える方法を新しく提案している．その方法について，以下に説明する．

Web サービスの実行数を削減する方法

本手法が提案する Web サービスを実行する数を削減する方法は，データの循環に含まれる Web サービスのみを実行し，かつそのデータの循環

の元となるデータが常に同じであるか，という識別の結果を，同じデータの循環を含む合成サービスに再利用する方法である．

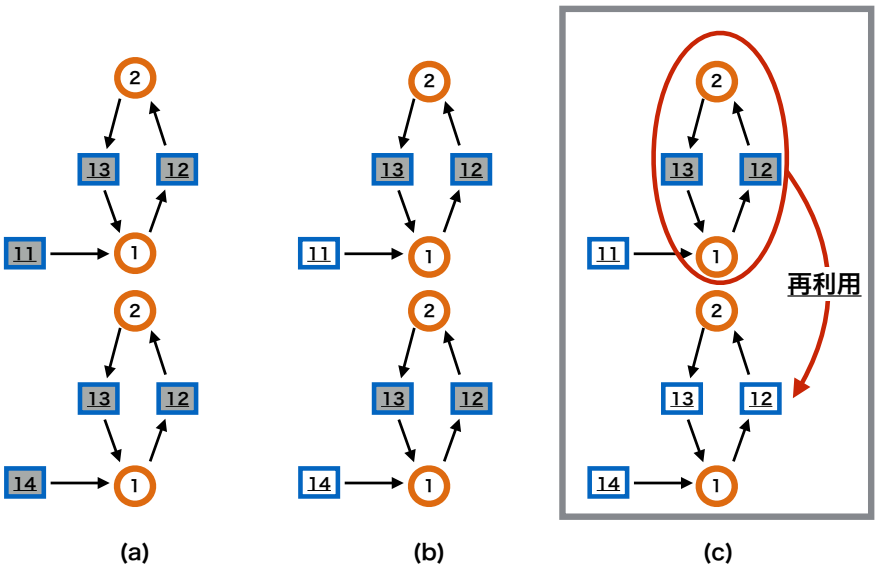


図 3.9: データの循環を含む合成サービスの実行の方法

図 3.9 に，本手法が提案する方法とそれ以外の方法との比較を示す．図 3.9(c) が本手法が提案する方法を表し，図 3.9(a) と (b) がそれ以外の方法を表す．それぞれの方法を，図 3.9(a)，(b)，(c) それぞれの上段と下段に示す {(11, 1), (1, 12), (12, 2), (2, 13), (13, 1)} と {(14, 1), (1, 12), (12, 2), (2, 13), (13, 1)} という 2 つの合成サービスに対して適用している．これらの合成サービスは $d(1)$ というデータの型を元に起こる同じデータの循環を含む．四角で囲まれた数字が表す頂点が Web サービスを表し，丸で囲まれた数字が表す頂点がデータの型を表す．また，背景が黒く塗り潰されている Web サービスは，それぞれの方法において実行される Web サービスを表す．

まず，図 3.9(a) に示す方法は，データの循環を含む合成サービスに含まれる全ての Web サービスを実行する方法である．この方法は，全ての Web サービスを愚直に実行する最もシンプルな方法であり，最も Web サービスを実行する数が多くなる方法となる．

次に，図 3.9(b) に示す方法は，データの循環に含まれる Web サービスのみを実行する方法である．データの循環の元となるデータが常に同じであるか，を判別するためには，データの循環に含まれる Web サービス

のみを実行すれば十分である．この方法では，図 3.9(a) に示す方法に比べ，データの循環に含まれる Web サービスのみを実行するため，個々の合成サービスにおける Web サービスの実行の数を減らす事が出来る．今回の例では，この方法を使う事によって，$s(11)$ と $s(14)$ の実行をせずに済む事になる．

最後に，図 3.9(c) に示す本手法が提案する方法は，データの循環に含まれる Web サービスのみを実行し，かつそのデータの循環の元となるデータが常に同じであるか，という識別の結果を，同じデータの循環を含む合成サービスに再利用する方法である．同じデータの循環であれば，その元となるデータが常に同じであるか，という情報は，常に同じとなる．そのため，同じデータの循環を含む他の合成サービスに，その情報を再利用する事が可能である．この方法では，図 3.9(b) の方法に比べ，同じデータの循環を含む合成サービスが複数あった時に，それらの中の 1 つの合成サービスのみを実行すれば良く，その他の合成サービスに含まれる Web サービスの実行を丸ごとを削減する事が可能である．今回の例では，まず上段の合成サービスで $d(1)$ を元に起こるデータの循環の $d(1)$ が常に同じであるか，という識別を行う．そして，その結果を同じデータの循環を含む下段の合成サービスに再利用する事によって，その下段の合成サービスに含まれる $s(14)$，$s(12)$，$s(13)$ の実行をせずに済む事になる．

3.2.3 ユーザによる作成が困難なデータのの入力が必要な機能のフィルタリング

ここでは，ユーザに掛かる手間が大きい機能として，ユーザによる作成が困難なデータの入力が必要な機能を持つ合成サービスを識別し，それらをフィルタリングする方法について説明する．

ユーザに掛かる手間が大きい機能は，ユーザが必要としない可能性が高い機能と言えるため，フィルタリングする．合成サービスにおけるユーザに掛かる手間は，ユーザが合成サービスに与える必要があるインプットのデータを用意する手間と表現する事が出来る．そのため，ユーザに掛かる手間が大きいデータとして，ユーザによる作成が困難なデータを識別する．

ここで，Web サービスが必要とするインプットのデータには，ユーザが即座に識別出来ないフォーマットを持つものが存在する．例えば，「ホテルを予約する」という Web サービスであれば，予約する対象のホテル

を識別するために「ホテル ID」という型を持つデータがインプットとして必要となる．ここで，この「ホテル ID」が持つフォーマットは，Web サービスを提供している組織によって，同じ「ホテル ID」という型を持つデータであっても，違うフォーマットを持つ可能性がある．これは,「ホテルを予約する」という機能に対して，その機能を Web サービスとして提供する組織の実装方法によって，扱うホテル情報のデータセットが異なるためである [23].

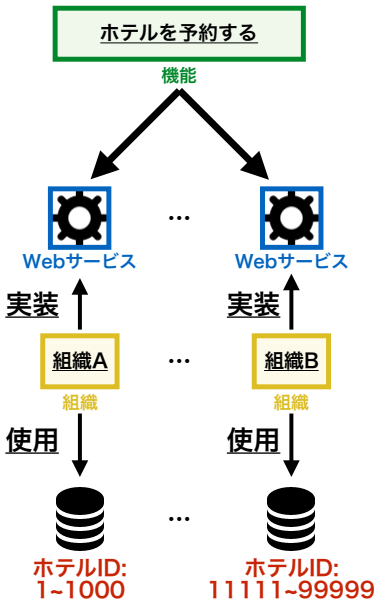


図 3.10: データのフォーマットの違いが起こる理由

同じ型を持つデータであっても，違うフォーマットを持つ可能性がある理由を，図 3.10 に示す．この例では,「ホテルを予約する」という機能に対して，組織 A は「ホテル ID」が 1〜1000 の数字というフォーマットで表現されるデータセットを使って Web サービスの実装を行っており，一方で組織 B は「ホテル ID」が 11111〜99999 という 5 桁の数字というフォーマットで表現されるデータセットを使って Web サービスの実装を行う．この様に，ある機能に対して，その機能を Web サービスとして提供する組織の実装方法によって，同じ型を持つデータであっても違うフォーマットを持つ可能性がある．

この時，これらのデータをユーザが与える必要があるとすると，ユーザは，そのデータがどの様なフォーマットを持つのか，また，自身が求め

るデータは，そのフォーマットでどの様に表現されるのか，という事を調べる必要がある．例えば,「ホテルを予約する」という Web サービスにおいてユーザが「ホテル ID」というデータを与える必要があるとすると，ユーザは対象の「ホテル ID」がどの様なフォーマットを持つのか，そして，自身の予約したいホテルは対象の「ホテル ID」が持つフォーマットでどの様に表現されるのか，という事を自身で調べる必要がある．これは大変な時間が掛かる作業であり，もしかしたらそれらの情報を手に入れる事が出来ない可能性もあり，それらのデータをユーザが作成する事は困難であると言える．そのため，ユーザによる作成が困難なデータとして，この様なユーザが即座に識別出来ないフォーマットを持つデータを識別する．

本手法におけるそれらのデータの入力が必要な機能を持つ合成サービスを識別する方法は，以下のアルゴリズム 4 によって表現する事が出来る．このアルゴリズムでは，ステップ 1 で導出された全ての合成サービスを All CompositeWebServices，それぞれの合成サービスを CompositeWebService，合成サービスに含まれる各 Web サービスを WebService，BPEL リポジトリに存在する全ての BPEL ファイルを All BPELFiles，BPEL ファイルを BPELFile，BPEL リポジトリに存在する全ての BPEL ファイルに含まれる Web サービスが必要とするインプットのデータの型の集合を All Inputs，BPEL リポジトリに存在する全ての BPEL ファイルに含まれる Web サービスに対して，ユーザが与える必要があるデータの型の集合を All UserInputs，そして，ユーザが即座に識別出来ないフォーマットを持つデータの集合を TargetData として定義する．

アルゴリズム 4 では，まずユーザが即座に識別出来ないフォーマットを持つデータ TargetData を識別する．この識別には，BPEL リポジトリに存在する BPEL ファイル All BPELFiles を使い，All Inputs の中で All UserInputs に含まれていないものを，TargetData として識別する方法を使う．これは即ち，All BPELFiles に存在する BPELFile 全てにおいて他の Web サービスが返すアウトプットを使っているデータの事を表す．この識別の方法を採用した理由を，以下に説明する．

ユーザが即座に識別出来ないフォーマットを持つデータは，他の同じデータセットを使って実装された Web サービスによって作成される事を想定しているデータと考えられる．例えば,「ホテルを予約する」という Web サービスであれば，そのインプットとして必要な「ホテル ID」は，ユーザが与える事を想定したものではなく，同じデータセットを使って

Algorithm 4 ユーザが即座に識別出来ないフォーマットを持つデータの入力が必要な機能が表す合成サービスのフィルタリング方法

```
for BPELFile in All BPELFiles do
  for WebService in BPELFile do
    All Inputs.Add(Inputs of WebService)
    All UserInputs.Add(UserInputs of WebService)
  end for
end for
for Input in All Inputs do
  if !All UserInputs.contains(Input) then
    TargetInputs.Add(Input)
  end if
end for
for CompositeWebService in All CompositeWebServices do
  for WebService in CompositeWebServices do
    for UserInput in UserInputs of WebService do
      if TargetInputs.contains(UserInput) then
        Filtering(CompositeWebService) //フィルタリング
      end if
    end for
  end for
end for
```

実装された別の Web サービス，例えば「ホテルを検索する」など，が返すアウトプットの「ホテル ID」を使う事を想定したものであると考えられる．

この時，BPEL リポジトリに存在する BPEL ファイルの全てにおいて，他の Web サービスが返すアウトプットを使っているデータは，それら全ての BPEL ファイルの開発者が，ユーザが与えるよりも他の Web サービスが与える必要があると判断したデータであり，他の特定の Web サービスによって作成される事を想定しているデータである可能性が高い．そのため，この様な PEL リポジトリに存在する BPEL ファイルの全てにおいて他の Web サービスが返すアウトプットを使っているデータを，ユーザが即座に識別出来ないフォーマットを持つデータとして識別する．

そして，ステップ 1 で導出された全ての合成サービス All CompositeWebServices を走査し，それぞれの CompositeWebService に対して，TargetData をユーザが入力する必要があるか，を識別する．そして，もしそれが必要な場合，対象の CompositeWebService をフィルタリングする．

3.3 グループ化 (ステップ 3)

このステップは，ステップ 2 でフィルタリングされた結果残った合成サービスを，要求分析者が 1 つの合成サービスの機能を理解する過程に沿ってグループ化を行うステップである．

ステップ 2 でフィルタリングされた結果残った合成サービスは，未だに数が多い場合がある．本手法では，要求分析者は提示された合成サービスが「何が実現出来るのか？」という事を考える．そのためには，要求分析者はそれぞれの合成サービスが持つ機能を理解する必要がある．ここで，提示される合成サービスの数が多い場合，その理解をする作業は時間が掛かる作業となる．そのため，この理解を支援する事が必要であり，本手法ではグループ化によって，その支援を行う．

ここで，要求分析者が 1 つの合成サービスの機能を理解する過程は，組み合わさっている個々の Web サービスの機能を順番に理解していくものと考えられる．例えば,「ホテルを地名から検索する」「ホテルを予約する」「クレジットカードで支払いを行う」という Web サービスが順番に組み合わさって出来る合成サービスであれば，要求分析者はまず「ホテルを地名から検索する」という機能を理解し，次に「地名から検索されたホテルのどれかを予約する」という「ホテルを地名から検索する」に

「ホテルを予約する」を組み合わせた機能を理解し，最後に全体の「地名から検索されたホテルのどれかを予約し，その支払いを行う」という機能を理解する，という過程を踏むと考えられる．この時，要求分析者は 1 つの合成サービスを理解する過程で，複数の機能を理解する事になると言える．この事を本手法では，1 つの合成サービスに複数の機能が含まれている，と表現する．

要求分析者にとって，この様な複数の機能を含む合成サービスの機能を理解し，そこからしばらくして，そこに含まれる機能を持つ合成サービスが改めて提示され，その機能を再度理解しようとする事は二度手間となる作業である．そのため，複数の機能を含む合成サービスと，そこに含まれる機能を持つ合成サービスはまとめて提示される事が好ましい．よって，本手法では，最も多くの機能を含む合成サービスと，そこに含まれる機能を表す合成サービスを同じグループとしてまとめて，要求分析者に提示する．

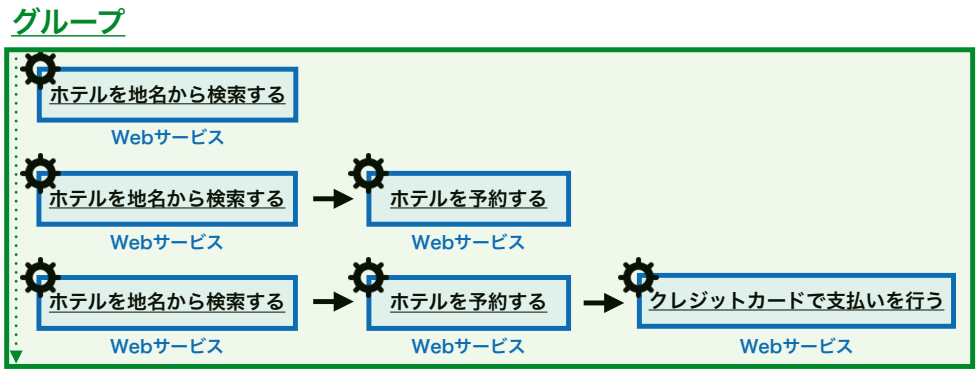


図 3.11: グループ化の例

グループ化の例を，図 3.11 に示す．この例は，先述した例の「地名から検索されたホテルのどれかを予約し，その支払いを行う」という機能を持つ合成サービスと，そこに含まれる「ホテルを地名から検索する」，「地名から検索されたホテルのどれかを予約する」という機能を持つ合成サービスを表す．本手法におけるグループ化では，これらの合成サービスを同じグループとしてまとめる．

また，それぞれのグループの合成サービスを，より少ない機能を含む合成サービスから順番にソートする事によって，要求分析者が 1 つの合成サービスの機能を理解する過程を再現し，その理解の支援を行う．例えば，図 3.11 に示す例であれば，最初に最も少ない機能を含む「ホテル

を地名から検索する」を提示し，次に「地名から検索されたホテルのどれかを予約する」を提示し，最後に最も多くの機能を含む「地名から検索されたホテルのどれかを予約し，その支払いを行う」を提示する，という矢印の順番に提示を行う．

第4章　評価

要求分析者が他のソフトウェアに存在しない新しい要求を獲得する事に対する本手法の有効性と無駄なデータの循環の識別のために本手法が提案したWebサービスの実行数を必要最低限に抑える方法の有効性を評価する．前者をRQ1〜RQ3によって評価し，後者をRQ4によって評価する．RQ1〜RQ4を以下に示す．

- RQ1: 本手法によって，どれだけの数の合成サービスをフィルタリングする事が可能であるか？
- RQ2: 本手法によって，ユーザが必要とする機能を表す合成サービスを過不足無く提示する事が可能であるか？
- RQ3: 本手法によって，他のソフトウェアに存在しない新しい要求を獲得する事が可能であるか？
- RQ4: 意味を持たないデータの循環の識別のため，本手法が提案した方法は，どれだけWebサービスの実行の数を削減する事が可能であるか？

4.1　データセット

本手法の有効性を評価するために，19個の「ホテル」というドメインに属するWebサービスを対象として，実験を行う．その対象となる19個のWebサービスの一覧を，図4.1に示す．ここでは，それぞれのWebサービスに対して1〜19の番号が割り当てられており，それぞれの番号の横にそれぞれのWebサービスが持つ機能が英語と日本語によって表現されている．例えば,「1」という番号が割り当てられた一番上のWebサービスは，その番号の横に書かれた「対象のホテルの全部屋データを取得する」という機能を持つ事が表現されている．

1：【findAllRooms（対象のホテルの全部屋データを取得する）】
2：【findAllHotels（保存されている全ホテルデータを取得する）】
3：【findHotelsByCity（指定した地名に存在するホテルを探す）】
4：【findHotelsByKeyword（指定したキーワードに名前が近いホテルを探す）】
5：【checkRoomAvailability（対象のホテルの部屋が対象の日にちに泊まれるかどうか、を返す）】
6：【getPrice（対象のホテルの部屋で対象の期間泊まるとどれくらいの値段になるか、を調べる）】
7：【reserveRoom（対象のホテルの部屋を対象の日にちで予約する）】
8：【pay（対象の予約の支払いをクレジットカードで行う）】
9：【cancelReservation（対象の予約を取り消す）】
10：【addHotel（ホテルデータを追加する）】
11：【addRoom（部屋データを追加する）】
12：【confirmReservation（対象の予約が正しく行われているか、を確認する）】
13：【getAllCurrentReservation（対象のユーザーの現在の全予約データを取得をする）】
14：【calculateHotelPreference（ホテルの好みを計算する）】
15：【recommendHotels（好みに合うホテルを指定したホテルのリストの中から推薦する）】
16：【findReservationById（予約のIDから予約データを探す）】
17：【findHotelsById（ホテルのIDからホテルデータを探す）】
18：【findRoomsById（部屋のIDから部屋データを探す）】
19：【getPresentLocation（現在位置を取得する）】

図 4.1: データセット

また，それぞれの Web サービスにおいて，その Web サービスが必要とするインプットのデータの型とその Web サービスが返すアウトプットのデータの型を定義している．その定義の 1 つを図 4.2 に示す．この例では，番号「3」の「指定した地名に存在するホテルを探す」という Web サービスは，インプットとして必要なデータの型として「地名」，またアウトプットとして返すデータの型として「ホテル情報一覧」を持つ事が定義されている．

3：【findHotelsByCity（指定した地名に存在するホテルを探す）】
・インプット：[CityName（地名）]
・アウトプット：[HotelInfos（ホテル情報一覧）]

図 4.2: インプットとアウトプットのデータの型の定義

また，Web サービスで定義されるデータには階層構造があるものが存在し，階層構造において上位のデータから下位のデータを取得する事が可能である．階層構造とは，一方のデータがもう一方のデータを包含する関係の事を指す．例えば，「ホテル情報」というデータが「ホテル ID」「ホテル名」「広さ」「評価」というデータを包含する時，これらのデータは包含関係を持っており，「ホテル情報」という上位のデータから「ホテ

ル ID」という下位のデータを取得する事が可能である．このデータセットの Web サービスにおいても，それぞれの Web サービスが持つデータ同士の階層構造を定義している．その定義の 1 つを図 4.3 に示す．

1.<HotelInfos(ホテル情報一覧)>
 1.1.<HotelInfo(ホテル情報)>
  1.1.1.<HotelId(ホテルID)>
  1.1.2.<HotelName(ホテル名)>
  1.1.3.<Breadth(広さ)>
  1.1.4.<Rate(評価)>

図 4.3: データの階層構造の定義

これらの Web サービスの機能は，既存の「ホテル」に関するソフトウェアを調べ，それぞれのソフトウェアが持つ機能を，Web サービスという単位で切り出したものである．そのため，これらの個々の機能は，現実世界における実際に存在する機能を表している．

これらの Web サービスを使い，情報学を専攻する早稲田大学の大学院生 10 名を対象とし，「これらの Web サービスを組み合わせて，合成サービスを作って下さい.」という合成サービスを作成して貰うアンケートを行った．その結果，合計 97 個の合成サービスを獲得する事が出来た．これらの 97 個の合成サービスは，情報学を専攻する早稲田大学の大学院生という開発者によって作成された BPLE ファイルと捉える事が出来るため，これらを本手法における BPEL リポジトリに存在する BPEL ファイルとして使う．

4.2 フィルタリング数における本手法の有効性 (RQ1)

本手法によって，どれだけの数の合成サービスをフィルタリングする事が可能であるか？という事を評価する．

この事を評価するために，対象のデータセットの Web サービスにおける全ての組み合わせ方の数と，そこから本手法で提案した 3 つの方法によってフィルタリングする事が出来た合成サービスの数を導出する．

表 4.1 に，他のソフトウェアが実装済みの機能のフィルタリング，無駄な処理を含む組み合わせのフィルタリング，ユーザによる作成が困難な

表 4.1: RQ2 に対する評価

フィルタリングの方法	合成サービスの数
全ての合成サービスの数	5929
(a) 他のソフトウェアが実装済みの機能のフィルタリング	43
(b) 無駄な処理を含む組み合わせのフィルタリング	5436
(c) ユーザによる作成が困難なデータの入力が必要な機能のフィルタリング	3543
(a)+(b)+(c) によるフィルタリング	5722

データの入力が必要な機能のフィルタリング，というそれぞれの方法によってフィルタリング出来た合成サービスの数を示す．

まず，対象のデータセットの「ホテル」というドメインに属する 19 個の Web サービスの全ての組み合わせ方を網羅すると，その合成サービスの数は「5929 個」となる．次に，本手法によって，それらの中から，他のソフトウェアが実装済みの機能が「43 個」，無駄な処理を含む組み合わせが「5436 個」，ユーザによる作成が困難なデータの入力が必要な機能が「3543 個」識別された事が分かる．そして，それら全てをフィルタリングすると，合計「5722 個」の合成サービスがフィルタリングされ，最終的に「207 個」の合成サービスが残った事が分かる．

よって，本手法を使わなければ要求分析者が「5929 個」の合成サービスをチェックしなければならなかったものを，本手法によって，そのチェックするべき合成サービスを「207 個」まで削減出来た事が分かる．「5929 個」の合成サービスをチェックする事は現実的では無いが，「250 個」であれば現実的であると考えられる．

表 4.2: RQ2 に対する評価

	グループの数
グループ化無し	207
グループ化あり	158

また，本手法で提案したグループ化の方法によって作成されたグループの数を，表 4.2 に示す．グループ化によって，フィルタリングの結果残った「207 個」の合成サービスが「158 個」のグループにまとめられている事が分かる．このグループ化の有効性を評価するために，同じく情報学を専攻する早稲田大学の大学院生 5 名を対象として，「このグループ化は，それぞれの合成サービスの機能を理解する支援になったか？」という「はい」と「いいえ」の 2 択から成るアンケートを行った．その結果，5 名中 5 名全てが「はい」という回答となった．そのため，本手法におけるグ

ループ化は，合成サービスの機能を理解するための支援となったと考えられる．

4.3 ユーザが必要とする機能の獲得支援における本手法の有効性 (RQ2)

本手法によって，ユーザが必要とする機能を表す合成サービスを過不足無く提示する事が可能であるか？という事を評価する．

この事を評価するために，本手法によって最終的に提示された合成サービス，本手法におけるユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法によってフィルタリングされた合成サービスに対して，情報学を専攻する早稲田大学の大学院生 10 人を対象として,「あなたがホテルに関するソフトウェアを開発しようと思った時，その合成サービスが表す機能は必要ですか?」というアンケートを行い，それらを 5 段階評価で回答して貰った．5 段階評価は，1: 「そう思う」，2: 「少し思う」，3: 「どちらでもない」，4: 「あまり思わない」，5: 「思わない」という 1 に近ければ近いほど必要という事を表す 5 つの指標を使う．

この結果から，本手法の有効性を示すために，Precisition と Recall を導出する．Precisition とは，本手法によって最終的に提示された合成サービスが，正解データとなる合成サービスをどれだけ含んでいるか？という事を表す指標である．Recall とは，本手法によって獲得出来た合成サービスが，正解データをどれだけフィルタリングせずに済んだか？という事を表す指標である．それぞれの結果は 0〜1 の値によって導出され，1 に近ければ近いほど良い結果を示す．そのため Precisition と Recall が，1 に近ければ近いほど，本手法の有効性を示す事になる．

本手法によって最終的に提示された合成サービスを RetrievedCWS，正解データとなる合成サービスを CorrectCWS とした時，Precisition と Recall を以下の式によって導出する事が出来る．ここで，それぞれの合成サービスに対するアンケートの回答の 5 段階評価を平均し，それが 3 未満となった合成サービスの機能を，ユーザが必要な機能と判断したものとして，本手法における正解データとして扱う．

$$Precisition = \frac{RetrievedCWS \cap CorrectCWS}{RetrievedCWS}$$

$$Recall = \frac{RetrievedCWS \cap CorrectCWS}{CorrectCWS}$$

Precisition と Recall の計算結果を，図 4.4 に示す．本手法における Precisition の結果は「0.54」，Recall の結果は「0.69」となった．本手法によって提示された合成サービスには，ユーザが必要とする機能が「111 個 (/207 個)」含まれており，本手法のユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法によってフィルタリングした機能には，ユーザが必要としない機能が「193 個 (/247 個)」含まれている．この結果から，本手法によって提示された合成サービスには，ユーザが必要とする機能が半数以上含まれており，また本手法のユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法は，7 割近い割合でユーザが必要としない機能をフィルタリング出来ている事が分かる．

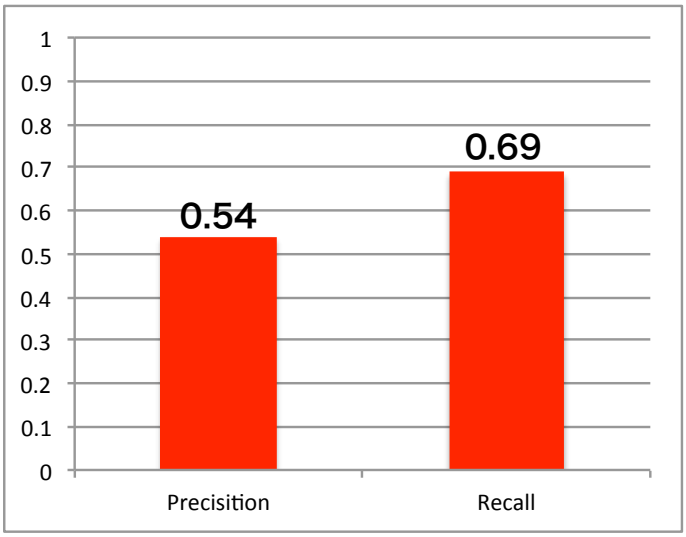


図 4.4: Precisition と Recall

4.4 他のソフトウェアに存在しない新しい特徴となる要求の獲得支援における本手法の有効性 (RQ3)

本手法によって，他のソフトウェアに存在しない新しい特徴となる要求を獲得する事が可能であるか？という事を評価する．

この事を評価するために，同じく情報学を専攻する早稲田大学の大学院生 10 名を対象とし，本手法によって最終的に提示された合成サービス，本手法におけるユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法によってフィルタリングされた合成サービスに対して,「これらの機能を使って，どの様な要求を実現出来ると思いますか？」というアンケートを行った．

その結果，本手法によって提示された合成サービスの中の RQ2 に対する評価でユーザが必要とする機能とした「111 個」の中で,「71 個」の機能を元にした要求が他のホテルに関するソフトウェアに存在しない新しい要求を示した．この「71 個」の要求は，主に「7 個」の似た要求にまとめる事が出来る．以下に，その「7 個」の要求を示す．

- 「現在予約しているホテルの部屋を、別のホテルの部屋に変更する」
- 「現在予約しているホテルの部屋を、別の現在予約しているホテルの部屋に変更する」
- 「現在予約しているホテルの部屋と同じ部屋を、新しく予約する」
- 「過去に泊まったホテルの部屋と同じ部屋を、新しく予約する」
- 「過去に泊まったホテルの部屋が、別のホテルの部屋でも予約出来たのか、を調べる」
- 「過去の予約を、後日支払う」
- 「過去の予約の証明書を再度発行する」

例えば，本稿のモチベーションで挙げた「ホテルを地名から検索する」,「ホテルの部屋情報を取得する」,「現在のホテルの予約状況を取得する」,「ホテルを予約する」という Web サービスを組み合わせて出来る「現在ホテルの部屋を予約している期間で，地名から検索されたホテルの部屋

のどれかを予約する」という機能を持つ合成サービスから,「現在予約しているホテルの部屋を、別のホテルの部屋に変更する」という要求を獲得する事が出来ている．この要求は，他のホテルに関するソフトウェアに存在しない新しい要求である．

次に本手法におけるユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法によってフィルタリングされた合成サービスにおいて,「50 個」の機能を元にした要求が他のホテルに関するソフトウェアに存在しない新しい要求を示した．しかし，この「50 個」の要求は，主に先述した「7 個」の要求にまとめる事が可能であり，これはユーザが即座に識別出来ないフォーマットを持つデータは，他の特定の Web サービスが作成する事が想定されるため，要求分析者の頭の中でその想定が補完され，その補完された合成サービスから要求が獲得された為である考えられる．その想定が補完された合成サービスは，本手法によって提示された合成サービスの中に含まれているものである．そのため，今回の実験においては，本手法におけるユーザによる作成が困難なデータの入力が必要な機能をフィルタリングする方法によって，その機能ならではの新しい要求の元となる機能を持つ合成サービスをフィルタリングせずに済んだ事が分かる．

4.5 無駄なデータの循環の識別のための Web サービスの実行数削減における本手法の有効性 (RQ4)

意味を持たないデータの循環の識別のため，本手法が提案した方法は，どれだけ Web サービスの実行の数を削減する事が可能であるか？という事を評価する．

この事を評価するために，本手法が提案した方法とそれ以外の方法において，意味を持たないデータの循環を識別するために実行された Web サービスの数を導出する．

表 4.3: RQ4 に対する評価

Web サービスの実行の方法	Web サービスの実行の数
全ての Web サービスを実行する方法	39894
データの循環に含まれる Web サービスのみを実行する方法	12944
本手法が提案する方法	58

その結果を，表 4.3 に示す．データの循環を含む合成サービスに含まれる全ての Web サービスを実行した場合，実行される Web サービスの数は「39894 個」である．これは，1 個の Web サービスに平均 1 秒の処理時間が掛かるとすると，意味を持たないデータの循環の識別のために，合計「11 時間 4 分 54 秒」掛かる事になる．次に，データの循環に含まれる Web サービスのみを実行した場合，実行される Web サービスの数は「12944 個」である．これは，1 個の Web サービスに平均 1 秒の処理時間が掛かるとすると，合計「3 時間 35 分 44 秒」掛かる事になる．最後に，本手法が提案する方法を使用した場合，実行される Web サービスの数は「58 個」である．これは，1 個の Web サービスに平均 1 秒の処理時間が掛かるとすると,「58 秒」で処理が終わる事になる．

よって，本手法が提案する方法によって，意味を持たないデータの循環を識別するために実行される Web サービスの数を，大幅に削減出来ている事が分かる．

4.6 考察

RQ1 に関して，それぞれのフィルタリングの方法について，考察する．まず他のソフトウェアが実装済みの機能のフィルタリングは,「43 個」の合成サービスをフィルタリングする事に成功した．ここに含まれる合成サービスは，予備実験の時に情報学を専攻する 10 人の大学院生が作成した合成サービスの機能を持つものである．予備実験の結果，作成された合成サービスが「97 個」である事に対して，フィルタリングしたものが「43 個」である事の理由としては，その「97 個」の中に同じ機能を持つものが多く含まれていたからだと考えられる．

次に，無駄な処理を含む組み合わせのフィルタリングでは,「5929 個」の合成サービスをフィルタリングする事に成功した．このフィルタリングの方法が一番多くの合成サービスをフィルタリング出来ている事になるが，これは同じ無駄なデータの循環を含む合成サービスが多く存在していたからだと考えれる．一方で，無駄では無いデータの循環を含む合成サービスは，フィルタリングされずに残っている．これは，本手法の対象のフィルタリングの方法が，正確に無駄なデータの循環と無駄では無いデータの循環を識別出来ていると言える．

最後に，ユーザによる作成が困難なデータの入力が必要な機能のフィルタリングでは,「3543 個」の合成サービスをフィルタリングする事に成

功した．ここで，ユーザが即座に識別出来ないフォーマットを持つデータとして，本手法は「ホテル ID」「部屋 ID」「予約 ID」「部屋 ID」「ホテル情報一覧」「予約情報一覧」「ホテルの好み」を識別している．この中では，まずユーザが即座に識別出来ないフォーマットを持つデータとして代表的なものであるホテルに関する「ID」を表すデータを，全て取得出来ている事が分かる．続いて,「ホテル情報一覧」「予約情報一覧」「ホテルの好み」なども「ID」と同様に，どの様なフォーマットで提供されるデータなのか，そして，自分の与えたいデータが，そのフォーマットにおいてはどの様に表現されるのか，を即座に識別する事は困難なため，ユーザが即座に識別出来ないフォーマットを持つデータと言える．

次に，RQ2 に関して，考察する．本手法に対する Precisition は「0.54」であり，本手法によって提示された機能の「54%」がアンケートの結果によってユーザが必要とする機能となっているが，一方で「46%」がアンケートの結果によってユーザが必要としない機能となっている．これは，本手法が，確実にユーザが必要としないと考えられる機能のみをフィルタリングしている為であると考えられる..

ここで，アンケートの結果ユーザが必要とする機能は全部で「161 個」存在し，その内本手法によって提示された機能には「111 個」のユーザが必要とする機能が含まれていた．もし本手法によってフィルタリングを行わなければ「5929 個」の機能を全てをチェックして，これらの「161 個」のユーザが必要とする機能を獲得出来る事になる．これは割合で言えば「2%」である，そして，このチェックに掛かる時間は，要求分析者が 1 つの機能をチェックするために 1 分掛かるとして,「98 時間 49 分 0 秒」の時間を要する．一方で，本手法によってフィルタリングを行えば「207 個」の機能をチェックして,「111 個」のユーザが必要とする機能を獲得する事が出来る．この割合は Precisition に示されている通り「54%」である．そして，このチェックに掛かる時間は，同じく要求分析者が 1 つの機能をチェックするために 1 分掛かるとしても,「3 時間 27 分 0 秒」の時間だけで済む事になる．よって，本手法によってフィルタリングされてしまったユーザが必要とする機能は確かに存在するが，それ以上に要求分析者がチェックする時間を大きく削減し，更にがユーザが必要とする機能を含む割合を増加させた事は，本手法の有効性を示していると考えられる．

また，本手法に対する Recall は「0.69」であり，ユーザが必要とする機能の「69%」を獲得する事が出来ている．しかし，一方で，ユーザの必要とする機能の「31%」をフィルタリングしてしまっている事になる．

この原因は,「ホテル ID」「部屋 ID」「予約 ID」「部屋 ID」「ホテル情報一覧」「予約情報一覧」のデータの入力が必要な機能に対しては，ユーザは軒並み必要としないという回答になった一方で,「ホテルの好み」だけに関しては，人によって回答が分かれるという結果になった事が挙げられる．人によっては「ホテルの好み」は比較的容易にフォーマットが識別出来るデータであると判断したと考えられる．現在本手法では，ユーザが即座に識別出来ないフォーマットを持つデータの識別のために，BPEL リポジトリの BPEL ファイルの全てにおいて Web サービスのアウトプットを使っているデータを使う方法を使用しているが，これは BPEL リポジトリの BPEL ファイルの量に精度が大きく依存している．もし「ホテルの好み」を他の特定の Web サービスが作成する事を想定されたデータではないと判断した人が「ホテルの好み」の入力が必要となる BPEL ファイルを作成していれば,「ホテルの好み」の入力が必要な合成サービスはフィルタリングされずに残っており，より精度が上がったと考えられる．

次に，RQ3 に関して，考察する．本手法によって提示された合成サービスの機能において，アンケートの結果ユーザが必要とする機能と識別された「111 個」の機能から，新しい要求を表すものは「71 個」獲得出来ており，それらは主に「7 個」の新しい要求にまとめられる．これは，本手法が他のソフトウェアに存在しない新しい要求を獲得する能力を有している事を示す．しかし一方で残りの「40 個」は他のホテルに関するソフトウェアに存在する機能であった．現在本手法では現在 BPEL リポジトリに存在する BPEL ファイルの機能を，他のソフトウェアが実装済みの機能としてフィルタリングしている．これは，他のソフトウェアが実装済みの機能をフィルタリング出来るかどうか，が BPEL リポジトリに存在する BPEL ファイルの量に大きく依存している事を表している．そのため，BPEL リポジトリに存在する BPEL ファイルの量が十分であれば，他のソフトウェアに存在する機能としてフィルタリングされる合成サービスの数は増え，より多くの割合で新しい要求の元となる機能を獲得出来ると考えられる．

最後に，RQ4 に関して，考察する．本手法が使う無駄なデータの循環の識別の方法は，他の方法に比べて，大幅に実行する Web サービスの数を減らす事が出来た．これは，同じデータの循環を含む合成サービスが多く含まれていた事が原因として挙げられるが，その様な事態は現実世界でも十分に考えられる．そのため，無駄なデータの循環と無駄では無いデータの循環を正確に，必要最低限の Web サービスの実行によって識

別出来た事は，本手法の有効性を示していると言える．

第5章　おわりに

5.1　まとめ

本稿では，合成サービスを用いた他のソフトウェアに存在しない新しい要求の発見を支援する手法を提案した．既存手法では，人手で単体のWebサービスを組み合わせて合成サービスを作成する事によって新しい要求を獲得していたが，これらのWebサービスの全ての組み合わせ方は膨大な数になる可能性があり，それらを人手で組み合わせ，それらの中から人手で新しい要求を探す事は膨大な時間が掛かる作業である．そこで本手法では，それらの合成サービスの中から3つの機能もしくは組み合わせを対象としてフィルタリングを行い，提示される合成サービスの数を減らす手法を提案した．また提示される合成サービスを，要求分析者が1つの合成サービスの機能を理解する過程に沿ってグループ化を行う事によって，要求分析者がそれぞれの合成サービスの機能を理解する支援を行う手法を提案した．

本手法の有効性を評価するために，19個のホテルに関するWebサービスをデータセットとし，実験を行った．その結果，そのデータセットにおけるWebサービスの組み合わせ方は全部で「5929個」あり，そこから本手法によって「207個」まで提示する合成サービスの数を削減する事に成功した．また，この「207個」の合成サービスの機能の中に，新しい要求の元となる機能が「7個」(似た要求を全て含めると「71個」) 含まれており，この「7個」の新しい要求を発見するために，本手法を使わなければ「5929個」の合成サービスをチェックする必要があった事に対して，本手法によって「207個」のチェックで済む事となった．これは，本手法によって新しい要求の発見のために不要な機能を大きく削減出来た事を示していると考えられる．

5.2 今後の課題

本手法における今後の課題は 3 つある．

まず更なる他のソフトウェアが実装済みの機能の識別の方法を考える事である．他のソフトウェアが実装済みの機能として本手法が識別した機能は「43 個」であった事に対して，最終的に提示された機能の中に他のソフトウェアが実装済みの機能が「40 個」含まれていたため，これらの機能を更にフィルタリングする事が求められる．そのため，更なる他のソフトウェアが実装済みの機能の識別の方法を追加する事が必要となる．

次に，更なるフィルタリングの方法を考える事である．本手法に対する Recall は「0.57」であるが，これは BPEL リポジトリの BPEL ファイルの量が増えれば，精度は上がると考えられる．そのため，フィルタリングの方法自体に問題があるとは言えない．一方で，Precisition は「0.69」であり，これは未だにユーザが必要としない機能が半分近くフィルタリング出来ずに残っている事を意味している．そのため，今後の課題としては，本手法が提案した 3 つのフィルタリングの方法に加え，今回フィルタリング出来なかったユーザが必要としない機能をフィルタリングする，更なるフィルタリングの方法を追加する事が必要となる．

最後に，更なるグループ化の方法を考える事である．本手法によって提示された合成サービスは似た機能を持つものが存在する．例えば，合成サービスにおいて「ホテルを地名から検索する」と「ホテルをキーワードから検索する」のみが異なり，それ以外は全て同じ場合，ほとんど同じ機能を表す．評価のアンケートを見ると，この様な似た機能を持つ合成サービスを元にした要求はほぼ同じとなっているため，これらの要求をグループ化する方法が求められる．

参考文献

[1] じゃらん, http://www.jalan.net/]

[2] 楽天トラベル, http://travel.rakuten.co.jp/

[3] sercus [サーカス] — みんなで作る Web サービスキュレーションサイト, http://sercus.jp/

[4] Github, https://github.com/

[5] ProgrammableWeb - APIs, Mashups and the Web as Platform, http://www.programmableweb.com/

[6] UDDI — Online community for the Universal Description, Discovery, and Integration, http://uddi.xml.org/

[7] IBM WebSphere Process Server, http://www-01.ibm.com/software/integration/wps/

[8] T.Clancy, The Standish Group Report Chaos, The Standish Group, 1995.

[9] W.Barry, P.Papaccio, Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, 1988.

[10] G.Kotonya, I.Sommerville, Requirements Engineering : Processand Techniques, John Wiley and Sons, 2002.

[11] T.Bhowmik, N.Niu, A.Mahmoud, J.Savolainen, Automated Support for Combinational Creativity in Requirements Engineering , Requirements Engineering Conference (RE), 2014.

[12] N.Maiden, S.Jones, K.Karlsen, R.Neill, K.Zachos, A.Milne , Requirements Engineering as Creative Problem Solving: A Research Agenda for Idea Finding, Requirements Engineering Conference (RE), 2010.

[13] F.Patterson, M.Kerrin, G.Gatto-Roissard, P.Coan, Everyday innovation : How to enhance innovative working in employees and organizations, NESTA research report, 2009.

[14] J.R.Sternberg, Handbook of creativity. New York, Cambridge University Press, 1999.

[15] R.B.Svensson, M.Taghavianfar, Selecting Creativity Techniques for Creative Requirements: An Evaluation of Four Techniques using Creativity Workshops, Requirements Engineering Conference (RE), 2015.

[16] J.Robertson, Eureka! How analysts should invent their requirements, IEEE Software, 2002.

[17] M.A.Boden, The Creative Mind, Abacus, 2004.

[18] J.Robertson, Requirements analysts must also be inventors, IEEE Software, 2005.

[19] K.Zachos, N.Maiden, X.Zhu, S.Jones, Does Service Discovery Enhance Requirements Specification: A Preliminary Empirical Investigation, Proceedings SOCCER Workshop, 2006.

[20] S.Jones, N.Maiden, K.Zachos, X.Zhu, How Service-Centric Systems Change the Requirements Process, Proceedings REFSQ2005 Workshop, CAiSE2005, 2005.

[21] K.Zachos, N.Maiden, Inventing Requirements from Software: An Empirical Investigation with Web Services, Requirements Engineering Conference (RE), 2008.

[22] Q.Liang, Y.Su, AND/OR Graph and Search Algorithm for Discovering Composite Web Services, International Journal of Web Services Research (IJWSR), 2005.

[23] Z.Gu, J.Li, B.Xu, Automatic Service Composition Based on Enhanced Service Dependency Graph, ICWS, 2008.

[24] V.Gabrel, M.Manouvrier, C.Murat, Optimal and Automatic Transactional Web Service Composition with Dependency Graph and 0-1 Linear Programming, ICSOC, 2014.

[25] X.Bai, W.Dong, W.Tsai, Y.Chen, WSDL-based automatic test case generation for Web services testing, Service-Oriented System Engineering, 2005.

[26] S.Basu, C.Pautasso, L.Zhang, X.Fu, Automatic WSDL-guided Test Case Generation for PropEr Testing of Web Services, ICSOC, 2013.

[27] N.Maiden , C.Ncube , S.Robertson , Can requirements be creative? Experiences with an enhanced air space management system , Software Engineering Conference , 2007.

[28] M.Michalko , Thinkertoys , Ten Speed Press, 2006.

[29] J.Lemos, C.Alves, L.Duboc, G.N.Rodrigues, A Systematic Mapping Study on Creativity in Requirements Engineering, SAC, 2012.

[30] R.Tang, Y.Zou, An Approach for Mining Web Service Composition Patterns from Execution Logs, ICWS, 2010.

謝辞

本論文の執筆にあたり，指導教員として様々なご指導を賜りました，早稲田大学 基幹理工学部 情報理工学科の深澤良彰教授に深謝致します．また，ご指導，ご助言いただいた国立情報学研究所の本位田真一教授，本研究及び論文執筆の細部にわたりご指導いただいた高橋竜一特任助教にも深謝致します．

また，研究生活の面でともに研究に励み，研究への助言を始めとした様々なご協力をいただいた早稲田大学大学院 基幹理工学研究科 情報理工学専攻 深澤研究室，鷲崎研究室の皆様．東京大学大学院 情報理工学系研究科 コンピュータ科学専攻及び創造情報専攻 本位田研究室の皆様，電気通信大学大学院 情報システム学研究科 大須賀・田原研究室の皆様にも感謝いたします．

業績

鶴貝康男, 高橋竜一, 深澤良彰, “ソフトウェア開発における潜在要求発見の支援手法”, 情報処理学会第 79 回全国大会, 愛知県, 名古屋大学東山キャンパス, 2017 年 3 月 16〜18 日.