特別研究報告書

評判情報に基づく言語サービスの選択

指導教員　　石田 亨 教授

京都大学工学部情報学科

後藤 真介

平成23年1月 31日

評判情報に基づく言語サービスの選択

後藤 真介

内容梗概

サービスコンピューティングとはサービスを基盤としたソフトウェア開発に関する研究領域であり，研究とビジネス両方から注目を浴びている分野の一つである．サービスコンピューティングにおいて，同等の機能を持つサービスのうち，どのサービスを実際に利用するかの選択は重要な課題となっている．有用なサービスを事前に知ることができれば，サービスを組み合わせた複合サービスの開発コストの削減が期待できる．これまで，サービス選択の指標として信頼性などを用いた客観的な評価であるサービス品質 (Quality of Service, QoS) が最も広く用いられてきた．機械翻訳サービスをはじめとする言語サービスでは，翻訳精度の人手による評価が QoS として用いられている．しかし，言語サービスにおいてサービスの有用性はユーザの外国語能力，あるいはタスクの難易度によって変化する．それぞれの外国語能力，難易度における言語サービスの有用性を人手によって評価することは大きなコストがかかってしまう．

そこで，本研究では QoS の代わりにサービスの有用性に関する評判情報を用いて，ユーザ，タスクに対して有用なサービスを選択することを提案する．評判情報をもとに，ユーザの能力とタスクの難易度，言語サービスの翻訳精度それぞれの順序関係を用いた背景知識を使い，ユーザのタスクの遂行にどのサービスが有用であるかの判定を行うことにより，サービスの選択を行う．本手法の実現にあたり，取り組むべき課題は以下の 2 つである．

評判情報に基づく順序関係取得の形式化

ユーザの能力，タスクの難易度，言語サービスの翻訳精度それぞれの間の順序関係が分かれば評判情報から有用なサービスを選択することができる．しかし，順序関係については評判情報や背景知識からは分からないため，順序関係を評判情報から取得する手法の形式化が必要である．

順序関係取得エンジンとサービス実行エンジンの統合

評判情報を用いた順序関係取得エンジンと，取得された順序関係を基にサービスの選択を行いサービスを呼び出す実行エンジンを統合することで，評判情報に基づくサービス選択プラットフォームを構築する必要がある．

本研究では，上記の問題を解決するために仮説推論を用いた順序関係取得手

法を提案する．仮説推論は，背景知識と観測結果を用いて，無矛盾の仮説集合を取得する推論方法である．仮説推論の枠組を用い，評判情報からユーザの能力，タスクの難易度，言語サービスの翻訳精度に関する順序関係を取得する．その後，取得された順序関係を用い，演繹推論によってサービス選択を行う．サービス選択は，ユーザの問い合わせからユーザとタスクを取得し，全てのサービスに対してそれぞれのサービスが有用であるかどうかを推論する．

このシステムのアーキテクチャは，評判情報からユーザ，タスク，サービスの相対情報の取得を行う仮説推論部，および取得した相対情報からユーザの問い合わせに対して有用なサービスを提案するサービス選択部からなる．このアーキテクチャの実世界での有効性を示すため，言語グリッド上での実装を行った．本研究の貢献は以下の通りである．

評判情報に基づく順序関係取得の形式化

ユーザ，タスク，サービス間の順序関係を仮説とし，それぞれの定義に基づく推論規則を背景知識と捉えることにより，サービス選択に必要な順序関係を取得する手法を仮説推論の枠組を用いて定式化した．

順序関係取得エンジンとサービス実行エンジンの統合

本研究では順序関係取得エンジンおよびサービス実行エンジンの二つのシステムを統合し，それぞれの関係を示した．そして，評判情報に基づくサービス選択のためのアーキテクチャを提案し，実装を行った．

Reputation-Based Selection of Language Services

Shinsuke GOTO

Abstract

Services computing, which is a new research field in the software development, is now attracting much attention in both research and business communities. In services computing, selecting the service from available services with the equivalent function is an important topic. If the right service can be found before the use, the cost of developing compound services can be reduced. To date, Quality of Service (QoS), which is the numerical value of service evaluation, is the most common factor of service selection. For the language services such as machine translation services, translation accuracy can be used as QoS metric. However, the usefulness of language services may vary according to the foreign language ability of the users, or the difficulty of the tasks.

In this study, I proposed to use reputation information for judging the usefulness of services instead of the QoS metrics. The reputations are defined as the information expressed by the user, task, and service. Based on reputation information, a system assumes what service is useful to carry out the task for the user. To realize this method, we face several problems.

Formalization of the order relation acquisition based on reputation information

If the order relation between the ability of the users, the difficulty of the tasks, and the accuracy of the language services are obtained, useful services could be selected. However, the order relation cannot be determined by reputation information and background knowledge only. Therefore, the formalization method to acquire the order relation from reputation information is needed.

Integration of order relation acquisition and service execution

In order to construct the service selection platform, we need to integrate the order relation acquisition engine, that is based on reputation information; and the execution engine which calls the service.

In this study, I propose the hypotheses acquisition method using the hypothetical reasoning in order to resolve the problem. Hypothetical reasoning is an

assumption method to obtain the consistent set of hypotheses from background knowledge and observation result. Firstly, the framework of hypothetical reasoning is used to obtain the order relation between the ability of users, the difficulty of tasks, and the accuracy of service from reputation information. Then, order relation obtained from hypothetical reasoning is used to recommend useful services by inference. Service selection system gets the information of the user and task from the query, then uses this to assume which service is useful.

The architecture of this system consists of the two components: the hypotheses acquisition component and the service selection conponent. The hypotheses aquisition component acquires the order relation information of users, task, and services from reputation information. The service selection component recommends useful services obtained from order relation information and reputation. We implemented this integrated system in Language Grid. Contributions of this research are as follows:

Formalization of the order relation acquisition method based on reputation information

Suppose that hypotheses are the order relation among users, tasks, and services, and background knowledge is the inference rule based on each definition, we formalized the order relation acquisition method necessary to service selection.

Integration of order relation acquisition and service execution

We showed the interaction between the order relation acquisition engine and the service execution engine. Moreover, we integrated the two components and then implemented the service selection platform to recommend services according to user's query.

評判情報に基づく言語サービスの選択

目次

第1章　はじめに

近年，Web サービスにおいて，サービスコンピューティングという概念が急速に発達している．サービスコンピューティングとは，迅速かつ柔軟なサービス開発を目標として開発された技術である．具体例として，機械翻訳サービスや専門用語対訳辞書サービスがコンポーネントとして存在する場合を考える．それぞれのサービスが独立して存在する場合，それぞれのサービスは別々に使われるのみである．しかし，これらが共通のインターフェースを持つ場合，機械翻訳サービスと対訳辞書サービスを組み合わせることが可能となる．そのため，一から開発を行うことなく辞書連携機械翻訳サービスを実現し，ドメインに特化したサービスを提供することができる．

言語グリッドプロジェクト [1] は，サービスコンピューティングの先駆的な研究を行うための基盤を提供している．機械翻訳，専門用語対訳辞書，形態素解析などのサービスがコンポーネントとして存在し，これらの多様な言語資源を組み合わせることによってユーザの環境に応じた複合サービスを迅速に作成することが可能となっている．

サービスコンピューティングにおいて，同等の機能を持つ複数のサービスからどのサービスを選ぶかは重要な課題である．既存のサービスから複合サービスを開発する際，最も有用なサービスを利用する前に知ることができれば，開発・評価コストの削減が期待できる．これまで広く用いられていたサービスの評価指標は，客観的なサービスの評価値を表す Quality of Service(QoS) であった，機械翻訳をはじめとする言語サービスにおいては，翻訳精度を数値化し，QoS として用いている．

しかし，翻訳サービスの有用性はユーザの外国語能力に依存する．翻訳精度が高い場合でも，外国語能力に優れたユーザにとっては機械翻訳は役に立たない可能性がある．例えば，TOEIC のスコアと英日機械翻訳の評価の間に相関関係があることが先行研究により示されている [2]．また，機械翻訳サービスは翻訳するタスクによって評価結果が違うことが知られている．Koehn ら [3] はそれぞれのタスクに対して 30 人以上の人が 200 から 300 文の機械翻訳文を評価することにより精度を判定している．しかしながら，このような人手による評価をそれぞれのユーザ，タスク，そしてサービスに対して行った場合コストは非常に大きくなってしまう．サービス連携を用いた場合，コストの問題はさらに大

きくなる．言語グリッド [1] には，英日間翻訳に絞った場合でも，5 個の機械翻訳サービスが存在する．さらにユーザ自身で対訳辞書を作成できるため，辞書連携翻訳サービスは無数に存在しうる．このような膨大な数のサービスに対し，それぞれのタスクの文章に対する評価を人手で行うことは現実的ではない．

本研究では，ユーザの特性やサービスについての十分な情報が得られていない場合に，有用と考えられるサービスを推測する枠組を考える．客観的な情報の代わりとしてユーザのサービスに対する評判情報を用いる．評判情報は，ユーザへのアンケートで比較的容易に収集することが期待できる．

表 1: 評判情報の例

User	Service	Task	Reputation
Alice	Translution	Chat	Useful
Bob	Translution	Chat	Useless
Bob	Google Translate	News	Useful

評判情報の例を表 1 に示す．本研究では，ユーザの評判を「ユーザの能力とタスクの難易度によって変わる閾値を言語サービスの翻訳精度が超えていればその言語サービスは有用であると判断する」と定義する．そのため，ユーザ，タスク，サービスそれぞれの順序関係が判明すれば，評判情報を用いてそれぞれのユーザにとってのサービスの有用性を演繹推論によって判定することができる．

しかし，評判情報を直接サービス選択の手法として用いることはできない．その理由として二つの問題が挙げられる．まず，客観的な評価を行う QoS と比較して一貫した情報が得られないという問題がある．また，同等の機能を持つサービスを同じユーザが全種類使うとは限らないため，十分な量の情報が得られない可能性がある．

本研究ではこれらの問題を解決するために，順序関係を仮説として取得することにより，サービス選択に必要な情報を取得するというアプローチをとった．評判情報と矛盾しない一貫した順序関係を取得する手法として仮説推論を用いることを提案する．仮説推論とは，観測された事実と背景知識の集合から，これらに矛盾しない仮説集合を得るための推論手法である．本研究では，仮説推論を用いた仮説取得手法，そして得られた仮説を用いた演繹推論によるサービ

ス推薦手法について説明する．

第2章　サービス選択

この章では，まずサービスコンピューティングの概要を説明する．その後，サービスコンピューティングで重要視されているサービス選択手法として最も広く用いられているQoS中心のアプローチ，そしてユーザ指向QoSのアプローチについて説明する．

2.1　サービスコンピューティング

サービスコンピューティングとは，ソフトウェア実装に関する新たな枠組であり，現在研究及びビジネスにおいて活動が盛んな分野となっている[4]．サービスコンピューティングを用いることにより，ソフトウェア開発の初期コストの軽減，そしてコンポーネントの再利用性の向上を図ることができる．

さらに，サービスコンピューティングはWebにおける集合知の実現に大きな役割を果たす．辞書や用例対訳，そして機械翻訳などの言語資源をWebサービスとしてラッピングすることにより，これらの資源を連携させることが可能となる．サービスコンピューティングに基づく言語資源の連携は異文化コラボレーションの様々な分野のツールを開発することを容易にする．

サービスコンピューティングにおいて重要視されている研究内容として，サービスの選択手法がある．これは二つの観点から重要である．まず，サービスのユーザにとって，様々なサービスを試行錯誤して選ぶことは大きな労力を必要とする．特に，複合サービスとなるとサービスの数は爆発的になるので，全てのサービスを試すのは不可能に近い．また，サービスの種類によっては試用することができず，サービスを評価するのにコストがかかってしまう．もう一つ，複合サービスの開発者にとって，出来るだけ有用なサービスをコンポーネントとして用いたい，という願望を持つのは自然である．あるサービスによって複合サービス全体の有用性が落ちてしまうことが考えられるためである．．これまで，このような目的で行われているサービス選択手法は，Quality of Service(QoS)に基づく選択が中心であった．主に，それぞれのサービスの特徴を数値あるいはベクトルとして扱い，機械的な計算によりサービスの有用性を測定する手法である．

2.2 QoS に基づくサービス選択

QoS とは元来ネットワーク分野で用いられてきた言葉であり，客観的な指標を基にサービスの有用性を数値化している．QoS を Web サービスの選択に適用させた例として，Zeng ら [5]，Liu ら [6] の先行研究がある．[5] では価格，耐久性，評判情報，信頼性，可用性を QoS の要素として数値化し，それぞれに重みを付けた総和をサービスの QoS としている．例えば，あるサービス s の QoS$q(s)$ が (価格，耐久性，可用性，信頼性，評判情報) の 5 次元ベクトルとして

$$q(s) = (q_{price}(s), q_{du}(s), q_{av}(s), q_{re}(s), q_{rep}(s))$$

と表され，ユーザの選好が

$$P = (p_{price}, p_{du}, p_{av}, p_{re}, p_{rep})$$

とあらわされるとすれば，サービスのユーザに対する評価値 $Score$ は

$$Score(s, P) = q_{price}(s) * p_{price} + q_{du}(s) * p_{du} + q_{av}(s) * p_{av} + q_{re}(s) * p_{re} + q_{rep}(s) * p_{rep}$$

となる．

しかし，このような QoS 評価を用いたサービス選択には，二つの問題がある．まず一つ目の問題は，サービスの QoS と実際のサービスの有用性が異なっている可能性である．言語サービスにおいて例をあげると，翻訳サービスの有用性はユーザの外国語能力に依存する．外国語能力に優れたユーザにとっては，誤訳の多い機械翻訳サービスは役に立たない．また，翻訳サービスの QoS はタスクによって大きく変化する．旅行コーパスで訓練を行った機械翻訳サービスを医療応用に用いれば，高い QoS は期待できない．これは，サービスの有用性は客観的なサービス品質だけで判断することはできないことを表す．これらの問題を回避するため，ユーザの能力に応じた QoS 評価も提案されている．その一例として，ユーザ指向 QoS がある．

2.3 ユーザ指向 QoS

Bramantoro ら [7] は，同じサービスに対してそれぞれのユーザによって QoS が異なる場合の QoS の求め方をユーザ指向 QoS として提案した．この論文では，それぞれのユーザにとっての QoS を以下のように定義している．

$$QoS(R) = Q_1(R) +, ..., +Q_m(R)$$

ここで，R はサービスであり，m は QoS の要素数，$Q_1(R)...Q_m(R)$ はそれぞれの要素の評価関数をあらわす．この $Q_1(R)...Q_m(R)$ を求める際にそれぞれのユーザの選好や特徴を数値化して用いている．この際，前述の一般的な QoS による評価手法と比較した場合の利点は，サービスの領域に特化した QoS の要素を加えることができる点である．Bramantoro らの多言語チャットにおける例では，翻訳サービスの評価の場合はユーザの外国語能力や翻訳サービスの精度などを用いている．

しかし，ユーザ指向 QoS によってサービス選択を行うためには，ユーザの選好や，それぞれの選好に対する QoS 評価を全てあらかじめ調査する必要がある．例えば，ユーザごとの翻訳サービスの有用性を図る場合，翻訳サービスの QoS である 5 段階評価と，ユーザの外国語能力との比較を行うことにより，ユーザの能力ごとの QoS を取得する．しかし，一般に翻訳サービスの 5 段階評価には大きなコストがかかる．Koehn ら [3] が実施した機械翻訳の精度の評価には，それぞれのタスクに対して 30 人以上の人が 200 から 300 文の機械翻訳文を評価することが必要となっている．また，ユーザ指向 QoS の取得法については触れられていない．ユーザが新たに QoS の要素を加え，それがまだ評価されていない要素である場合，追加の評価が必要となる．この評価を行う際に，迅速かつコストを抑えた開発というサービスコンピューティングの利点が失われてしまう．

本研究ではユーザ情報や QoS 評価が十分に得られない場合に有用なサービスを選択することを目的としている．これは QoS の代わりにサービスを選択する指標を定義する必要があると同時に，その指標が数値によるものでない場合は得られた指標を用いたサービス選択を行う必要があるということを意味する．

第 3 章　評判情報を用いたサービス選択手法

3.1　評判情報の概観

本研究では，矛盾を含む評判情報から一貫した仮説を取得するために，評判情報を仮説推論によって定式化することを試みる．評判情報を観測とし，全ての評判を証明するために必要な仮説集合を背景知識から求める．そのためには，評判情報，仮説集合，背景知識の定式化が必要となる．

客観的な QoS 評価と比較して，評判情報には以下の問題点がある．

1. 異なるユーザは，同じサービスに対しても異なる評価を下す可能性がある

2. 同じユーザが全てのサービスを使う場合は少なく，一貫した評価が得られにくい

それぞれの問題の具体例を図に示す．1. の問題として，表 2 では，同じサービス，同じタスクに対して Alice は有用であると判定し，Bob は有用でないと判定している．そのため，一般的に Translution が有用であると判定することはできない．2. の具体例を表 3 に示す．表 3 では Alice が Translution を有用だと判定し，Bob は Google Translate を無用と判定している．しかし，この情報だけから単純に J-server の方が Translution より有用性が高いと単純に評価することはできない．

表 2: 評判情報の問題点:ユーザ間の矛盾

User	Service	Task	Reputation
Alice	Translution	Chat	Useful
Bob	Translution	Chat	Useless

表 3: 評判情報の問題点:情報不足

User	Service	Task	Reputation
Alice	Translution	Chat	Useful
Carol	Google Translate	Chat	Useless

このような問題を解決し，ユーザに有用なサービスを選択するために，本研究では仮説推論を導入する．仮説推論とは，背景知識，事実から証明できない観測情報を証明するために仮説を用い，仮説集合によって観測情報が証明できた場合，仮説集合は実際に正しいだろう，と考える論理の枠組である．まずは仮説推論の枠組を用いるため，評判情報を一階述語論理を用いて定式化する．一階述語論理とは，オブジェクト同士の関係を扱うのに適した論理体系である．例えば，J-Server がサービスであることを一階述語論理では `service(J-server)` と表す．以下では，評判情報，及びこの論文で用いる用語を定義していく．まず，ユーザのサービスに対する評価は，第 1 章の記述と同様，以下のように定義する.「ユーザの能力とタスクの難易度によって変わる閾値を言語サービスの

翻訳精度が超えていればその言語サービスは有用であると判断する」

この定義に存在していないサービス，ユーザ，タスク以外の情報は有用性の判断に対して影響を及ぼさないと仮定する．また，サービス，ユーザ，タスクのそれぞれを以下のように定義する．

サービス

機械翻訳サービスには，評価指標として精度がある．例えば，機械翻訳サービスとして「Google Translate」,「Translution」,「J-server」が存在し，精度が高ければそれだけユーザはサービスが有用であるという判断を下しやすい．

ユーザ

サービスを利用するユーザには，それぞれサービスを用いずに自力でタスクを遂行する能力があると仮定する．ユーザは自分の能力とサービスの能力を比較してサービスの有用性判定を行う．ユーザの能力が低い場合，サービスの機能が有用であるという判断を下しやすい．

タスク

タスクは，サービスの提供する機能を用いる目的を表す．翻訳サービスでは,「チャット翻訳」,「ニュース記事翻訳」などが例として挙げられる．タスクにはそれぞれ難易度があり，タスクの難易度が容易であればユーザは有用だという判断を下しやすい．

評判情報の例として,「Alice はチャット翻訳に Translution を用いた結果，有用であると判定した」という文がある．これを，$useful(Alice, chat, Translution)$ と記述する．

仮説の例としては,「Alice は Bob よりも能力が高い」,「J-server は Google 翻訳よりも精度が高い」などの相対情報がある．ユーザの評価の理由として，このような相対情報が存在していると本研究では考える．これは，以下のように記述できる．

$higher\_ability(Alice, Bob), higher\_accuracy(J-server, Google\ Translate)$

これらの定義のもとに，以下では仮説推論を用いた順序関係取得法について議論する．

3.2 評判情報からの仮説取得手法

3.1 に挙げた定義のもとに，サービス，ユーザ，タスクの間の順序関係を取得する方法として，本研究では仮説推論を用いる．一般的に，仮説推論は真か偽か不明な事柄を真と考えて議論を進め，矛盾なく説明が達成できれば立てた仮説は正しかった，と考える形式の推論である [8]．今回は論理表現を基礎とする仮説推論を用いる．仮説推論は，以下の要素によって定式化される．

Σ　いつでも成り立つ背景知識 (事実) の集合

H　否定される可能性のある知識 (仮説) の集合

O　実際の観測結果の集合

仮説推論の動作の概要は以下の通りである．

まず，Σ から O が演繹的に証明できるか確かめる．Σ からだけでは O が証明できないとき，次の条件を満たす解仮説 $H' \subset H$ を求める．

$$\begin{cases} H' \cup \Sigma \vdash O \\ H' \cup \Sigma \ is\ consistent \end{cases}$$

上の式は，H' と Σ から O が証明できることを表し，下の式は H' と Σ が無矛盾であることを示している．すなわち，背景知識だけから O を証明できないとき，仮説集合の中から背景事実と矛盾しない部分集合 H' を切り出し，Σ と合わせて O を証明する．単純なバックトラックによる推論機構を用いた仮説推論の流れを示す．まず，与えられた観測結果から出発して，後ろ向きに推論を行う．観測結果が証明されない場合，選ばれた仮説を格納する仮説ボックスに，仮説集合から新たに仮説を選ぶ．新たに仮説を選ぶことができなくなった場合，バックトラックにより別の仮説が選択される．この結果，観測結果が証明できた場合は仮説ボックスに格納された仮説の集合が一つの解になる．この操作を，強制的なバックトラックをすることにより，全ての仮説の組み合わせに対して行う．

評判情報を仮説推論の枠組に当てはめると，以下のように (Σ, H, O) を定義できる．

Σ　順序推定のための推論規則，無矛盾性制約，ドメイン固有の知識

H　ユーザの能力，タスクの難易度，サービスの QoS における相対情報

O　アンケートによって取得された評判情報

背景知識である Σ のうち，推論規則は以下の６つである．これは，ユーザ，タスク，サービスに対しての定義を根拠としている．

- 自分と同等もしくはそれ以上の能力のユーザが，同じタスクで有用と判断したサービスは，自分にとって有用である.（他ユーザを用いた類推）
- 自分未満の能力のユーザが，同じタスクで無用と判断したサービスは，自分にとって無用である.（他ユーザを用いた類推）
- ある人が有用と判断したサービスよりも，品質が高いとされているサービスは，同じタスクでその人にとって有用である.（QoS の向上）
- ある人が無用と判断したサービスよりも，品質が低いとされているサービスは，同じタスクでその人にとって無用である.（QoS の低下）
- ある人があるタスクで有用と判断したサービスは，そのタスクよりも容易なタスクにおいてもその人にとって有用である.（タスクの容易化）
- ある人があるタスクで無用と判断したサービスは，そのタスクよりも困難なタスクにおいてもその人にとって無用である.（タスクの困難化）

また，無矛盾性制約として，それぞれのユーザの能力，タスクの難易度，サービスの QoS についての順序関係は無矛盾とする．すなわち，二人のユーザがいた場合，その能力関係は片方が能力が高いか，どちらとも言えない，のどちらかしかあり得ないということである.

さて，このようにしてサービスに対する評判情報を仮説推論に用いる枠組が整ったのであるが，仮説推論をそのまま評判情報からの順序関係取得に適用させる場合，以下のような問題がある.

1. 評判情報の数が十分でない場合，どのような仮説集合に対しても評判情報の証明に失敗する可能性がある
2. 前述した推論規則に基づいて証明を行う場合，ある評判情報を証明する際に他の評判情報が知識として扱われていない

これらの問題を解決するため，以下のアプローチをとった.

1. 評判情報の証明に失敗した場合，少なくとも無矛盾である仮説集合を取得する
2. 一つの評判情報を証明する際，他の評判情報は全て正しいとして知識集合に代入する

それぞれのアプローチについて詳しく説明する．1. は仮説推論を用いた証明に失敗した評判情報を知識として仮説推論を行い，矛盾が発生しない仮説集合を解とする．この際，観測情報を空集合にすることによって，知識とした評判情報と仮説が矛盾していないかを確かめることができる．実際のアルゴリズム

では，一つ一つの評判情報に対して無矛盾の仮説を取得することをせず，証明に失敗したら空集合を返している．そして全ての仮説推論が終わった後に，仮説集合が無矛盾であるかどうかのチェックを行っている．2. は評判情報の数だけ仮説推論を行うことを意味する．その後，それぞれの仮説推論によって得られた仮説集合を結合させることによって，それぞれの評判情報を証明することができる仮説集合を取得する．

これらを踏まえた，サービス選択に用いる仮説取得アルゴリズムを以下では説明する．仮説取得アルゴリズムは，評判情報，仮説集合および背景知識を入力として，おのおのの評判情報を証明可能な無矛盾の仮説集合のリストを出力とするアルゴリズムである．仮説取得を行う際の前提として，評判情報は全て正しいとする．さらに，背景知識と仮説集合以外に評判情報を証明できる述語は存在しないとする．この考え方は閉世界仮説といい，真であると宣言されていない述語は全て偽となる [9].

最初に，自然言語での処理の概要を記述する．

1. 評判情報の集合 R から評判情報 r_i を一つ選ぶ
2. 1. で選ばれなかった評判情報 $R - \{r_i\}$ ともともとの入力である背景知識の和集合を 3. で用いる背景知識とする
3. r_i を観測情報とし，2. で更新された背景知識および仮説集合 H を用いて仮説推論を行う．
4. 仮説推論の結果を P_i として保存する．P_i には r_i を証明できる複数の仮説集合が含まれる
5. 全ての r_i に対して 1.〜4. を繰り返す
6. CH を空集合とする．CH はそれぞれの観測情報を証明できる無矛盾の仮説集合を表す
7. A に $P_1, ..., P_n$ の直積集合を代入する
8. A の要素 a_k に含まれる仮説集合を全て結合させた仮説集合 PH に対し，制約知識に反していないかを調べる．反していなければ CH に PH を追加する
9. 全ての a_k に対して 8. を繰り返す.
10. CH を返して終了

Algorithm 1 仮説取得アルゴリズム $acquire\_hypotheses(K, H, R)$

K /* 背景知識の集合 */
H /* 仮説集合 */
$R = \{r_1, ..., r_n\}$ /* 評判情報の集合 */
CH /* 無矛盾かつ各々の評判情報を証明可能な仮説集合の集合 */
P_i /* 評判情報 r_i を証明可能な仮説集合の集合 */
$p_i \subset H$ /* 評判情報 r_i を証明可能である仮説集合の一つ．$p_i \in P_i$ */
$IC \subset K$ /* 制約知識 */
A /* $P_1, ..., P_n$ の直積集合．各々の評判情報を証明可能である仮説集合の組の集合を意味する */
$a_k = \{p_1, ..., p_n\} \in A$ /* 各々の評判情報を証明可能である仮説集合の n-組 */
PH /* 各々の評判情報を証明可能である仮説集合 */
for all r_i **in** R **do**
    $P_i \leftarrow \text{abduction}(K \cup (R - \{r_i\}), H, r_i)$
end for
$A \leftarrow \prod_{i=1}^{n} P_i$
$CH \leftarrow \emptyset$
for all a_k **in** A **do**
    $PH \leftarrow \emptyset$
    for all p_i **in** a_k **do**
        $PH \leftarrow PH \cup p_i$
    end for
    if $check\_consistency(K \cup R, PH, IC)$ **then**
        $CH \leftarrow CH \bullet PH$
    end if
end for
return CH

Algorithm 2 無矛盾性をチェックする関数 $check\_consistency(K, H, IC)$

1: **if** $K \cup H \vdash IC$ **then**
2: **return** $false$
3: **end if**
4: **return** $true$

Algorithm1 において，$acquire_hypotheses$12 行目の abduction(K, H, O) は仮説推論の本体であり，背景知識 K，仮説集合 H，観測情報 O から無矛盾かつ O を証明可能な仮説集合のリスト $P = \{H'_1, ..., H'_m\}, H'_i \subset H$ を出力する．O を証明可能な仮説集合がない場合は空集合を出力する．また，$acquire_hypotheses$22 行目の $\bullet$ は左辺に右辺の要素を追加した集合を返す処理を意味する．また，$check\_consistency$ 1 行目の $\vdash$ は，左辺から右辺が証明可能であることを表す．

3.3 仮説を用いたサービス選択手法

3.2 では，評判情報を基に無矛盾の仮説集合を取得する方法について説明した．本研究の目標はサービスの選択を行うことであるので，仮説集合の取得だけでなく，取得した仮説集合を基にサービス選択を行う必要がある．演繹推論を用いてサービス選択を行う例に Baldoni[10] がある．これは複合サービスが利用可能，利用不可能の基準でサービス選択を行っている．また，Sohrabi[11] では，ユーザの選好が完全に判明している場合に対して複合サービスの選択を行っている．

本研究では利用可能なサービスの集合に対して有用，有用でないかの判断を行うアルゴリズムを提案する．前提として仮説取得アルゴリズムによって得られた無矛盾の仮説集合が一つに定まっているとする．また，仮説取得部における前提と同様に，評判情報が全て正しいとする．

Web サービス選択アルゴリズムは，ユーザの問い合わせに対して，有用なサービスの集合を返す．ユーザの問い合わせは，ユーザ自身の情報と，ユーザの遂行しようとするタスクについての情報から構成されている．サービス選択のために必要なデータは，評判集合，無矛盾な仮説の集合，背景知識，サービス集合である．まず，処理の概要を自然言語で記述する．

1. サービスを一つ選ぶ．

2. 選んだサービスとユーザの問い合わせであるユーザ，能力に対して，サービスの有用性を判定する．
3. サービスが有用である場合，そのサービスを有用サービス集合に追加する．
4. 利用可能な全てのサービスに対して 1.〜3. を行う．
5. 有用サービス集合をユーザに返して終了．

実際のアルゴリズムを Algorithm3 に示す．ここで$\vdash$は，仮説取得部のアルゴリズムと同様に左辺から右辺が証明できることを表す．サービスの有用性を判定する際，サービスが有用であることの証明を試みる．有用であることが証明された場合，ユーザ，タスク，サービスの 3 つ組に対して有用であることを返す．証明されなかった場合，有用でないことを推論結果として返す．

Algorithm 3 Web サービス選択アルゴリズム $service\_selection(Q)$

K /* 背景知識の集合 */
CH /* 無矛盾かつ各々の評判情報を証明可能な仮説集合のリスト */
R /* 評判情報の集合 */
$Q = (u, t)$ /* ユーザの問い合わせ．ユーザ情報 u と，タスク情報 t からなる */
$S = \{s_1, ..., s_n\}$ /* サービス集合 */
US /* 有用なサービス集合 */
BK /* サービス有用性判定のために用いる知識の集合 */
$US \leftarrow \emptyset$
$BK \leftarrow K \cup CH \cup R$
for all s_i **in** S **do**
  if $BK \vdash \mathrm{useful}(s_i, u, t)$ **then**
    $US \leftarrow US \bullet s_i$
  end if
end for
return US

以上で，評判情報からサービス選択を行うための枠組が整った．ここで，3.2 節および 3.3 節で定義した仮説取得法およびサービス選択法を用いて，サービス選択を行う具体例を以下に示す．日英翻訳サービスに，Translution，Google

Translate，J-Server の 3 種類の選択肢があり，Alice は英語のニュース記事の翻訳に使える機械翻訳サービスを探しているとする．この時，表 4 のような評判情報が得られていると仮定する．ここから有用なサービスを選択する．

表 4: 評判情報

User	Service	Task	Reputation
Alice	Translution	Chat	Useful
Bob	Translution	Chat	Useless
Bob	Google Translate	Chat	Useful
Carol	Google Translate	Chat	Useless
Carol	Translution	News	Useless
Carol	J-Server	Chat	Useful

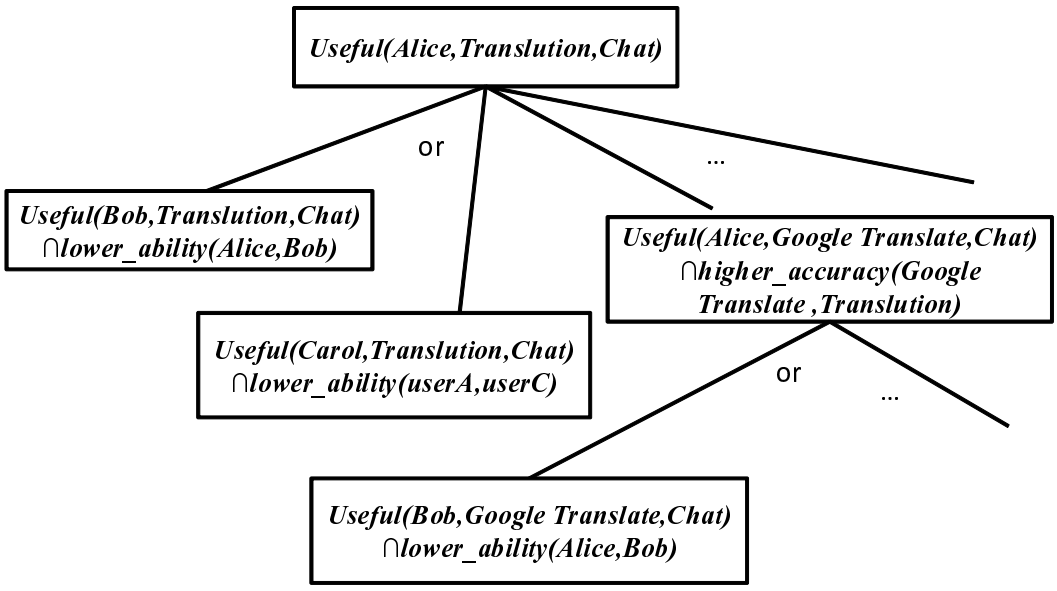

図 1: 仮説推論の実行過程

まず，仮説を取得する過程を説明する．一番目の評判情報を説明する過程は以下図 1 のように示される．$useful(Alice, Translution, Chat)$ を証明するため，まずユーザに関する推論規則を用い，$useful(Bob, Translution, Chat) \cap lower\_ability(Alice, Bob)$ を証明しようと試みる．しかし，知識としての評判

情報に $useless(Bob, Translution, Chat)$ が存在するため，矛盾を起こしてしまう．同様に $useful(Carol, Translution, Chat) \cap lower\_ability(Alice, Carol)$ の証明にも失敗する．そこで，サービスに関する推論規則を用いて $useful(Alice, GoogleTranslate, Chat) \cap higher\_accuracy(GoogleTranslate, Translution)$ の証明を試みる．この際，$useful(Alice, GoogleTranslate, Chat)$ はまだ現れていないため，もう一度展開され，$useful(Bob, GoogleTranslate, Chat) \cap higher\_accuracy(GoogleTranslate, Translution) \cap lower\_ability(Alice, Bob)$ となる．$useful(Bob, GoogleTranslate, Chat)$ は評判情報として存在するため真となり，仮説 $higher\_accuracy(Google\ Translate, Translution) \cap lower\_ability(Alice, Bob)$ が選択される．同様のプロセスを全ての評判情報に対して行うことで，それぞれの評判情報を説明できる仮説集合が取得される．それぞれの仮説集合を全て結合されることで，仮説集合 $higher\_accuracy(GoogleTranslate, Translution) higher\_accuracy(J - Server, GoogleTranslate) \cap lower\_ability(Alice, Bob) \cap lower\_ability(Bob, Carol)$, $easier\_task(Chat, News)$ が得られる．

そして，サービスの選択を行う過程を示す．

Alice は News のタスクに対して有用なサービスを求めているため，$useful(Alice, GoogleTranslate, News), useful(Alice, Translution, News), useful(Alice, J - Server, News)$, のそれぞれの証明を試みる．結果として，J-server は $useful(Carol, J - Server, News) \cap lower\_ability(Alice, Bob) \cap lower\_ability(Bob, Carol)$ から有用性が証明できるものの，Translution, Google Translate は有用であるかどうか判定できないため有用でないという出力となる．

第 4 章　サービス選択のアーキテクチャ

第 3 章で説明したアルゴリズムに基づく，サービス選択システムの実際の構成を説明する．このシステムは，二つのコンポーネントからなる．一つは，評判情報から無矛盾の仮説を取得する仮説取得部である．もう一つは，仮説取得部で得られた無矛盾の仮説を用い，ユーザの問い合わせに対して有用なサービスを推薦するサービス選択部である．まず，システム全体の構成図を図 2 に示す．ここでは，ユーザが二つの役割を果たしている．一つは，評判情報を入力し，仮説取得のために必要な情報の追加を行っている．もう一つは，ユーザが問い

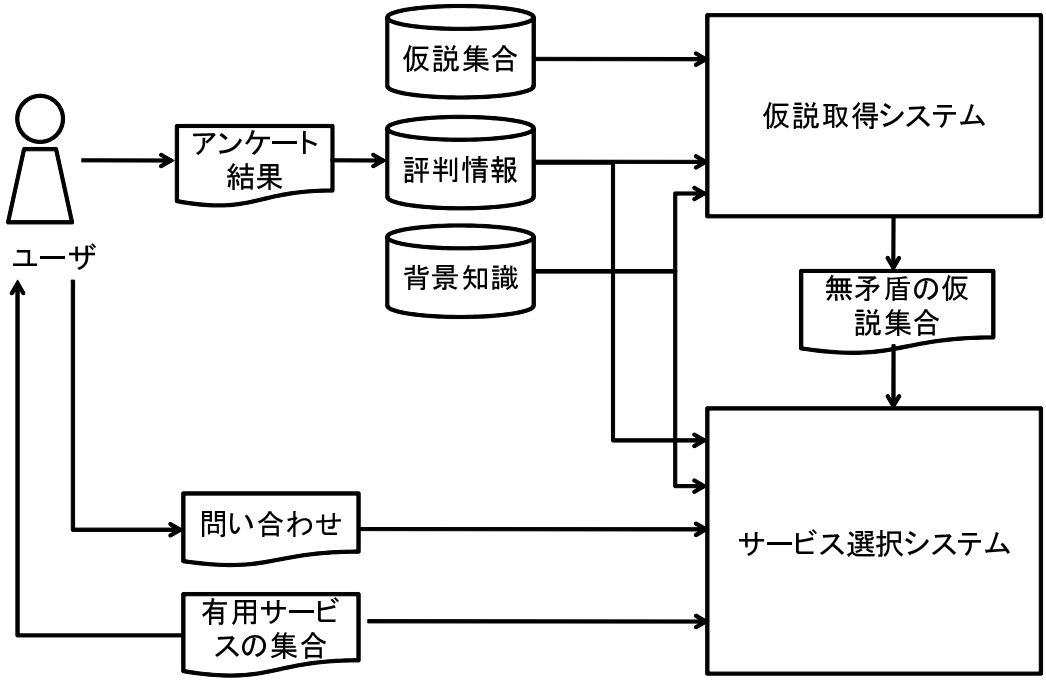

図 2: システム全体の構成図

合わせを行うことにより，サービス選択部から有用なサービスを取得している．ユーザの問い合わせに対して有用であると推論できるサービス集合を提供する．これらは互いに作用することで，より有用なサービスの選択を行う．ユーザはまず，サービス選択部を実行し，その選択のもとにサービスの実行を行う．その後，サービスの評価を評判情報として仮説取得部の評判情報を更新する．仮説取得システム，サービス選択システムそれぞれの詳細は以下で述べる．

4.1 仮説推論部

仮説取得部のシステム構成図を図 3 に示す．仮説取得部は，3.2 節で説明した仮説取得アルゴリズムを用い，無矛盾の仮説集合を出力する．ここで，仮説取得システム枠内の処理がアルゴリズムの本体であり，その中で仮説推論が用いられている．サービス評価者は，サービスを利用した後にアンケートフォームからサービスの評判情報を入力する．アンケートで入力される評判情報は，第 3 章節の定義のように,「ユーザ，タスク，サービス」の 3 つ組と，それに対する有用性で構成される．ユーザからアンケート結果が入力されるたび，評判情報のデータベースにデータが追加され，全ての評判情報を入力として仮説取得システムが起動する．仮説取得システムの内部での処理は 3.2 節で説明したア

ルゴリズムに従っている．まず全ての評判情報に対して，それぞれの評判情報を証明できる仮説集合を仮説推論によって取得する．その後，それぞれの評判情報を証明できる仮説集合を入力として，無矛盾性のチェックを行う．これにより，無矛盾の仮説集合が仮説推論を用いて更新される．ここでの背景知識であるが，図 3 にある推論規則及び制約知識はサービスの領域に関係なく同じである．ドメイン固有の知識は，例えばサービスやタスク，ユーザの集合である．また，サービスの領域に特化した規則を追加で入力しようとする場合も，ドメイン固有の知識として追加する必要がある．

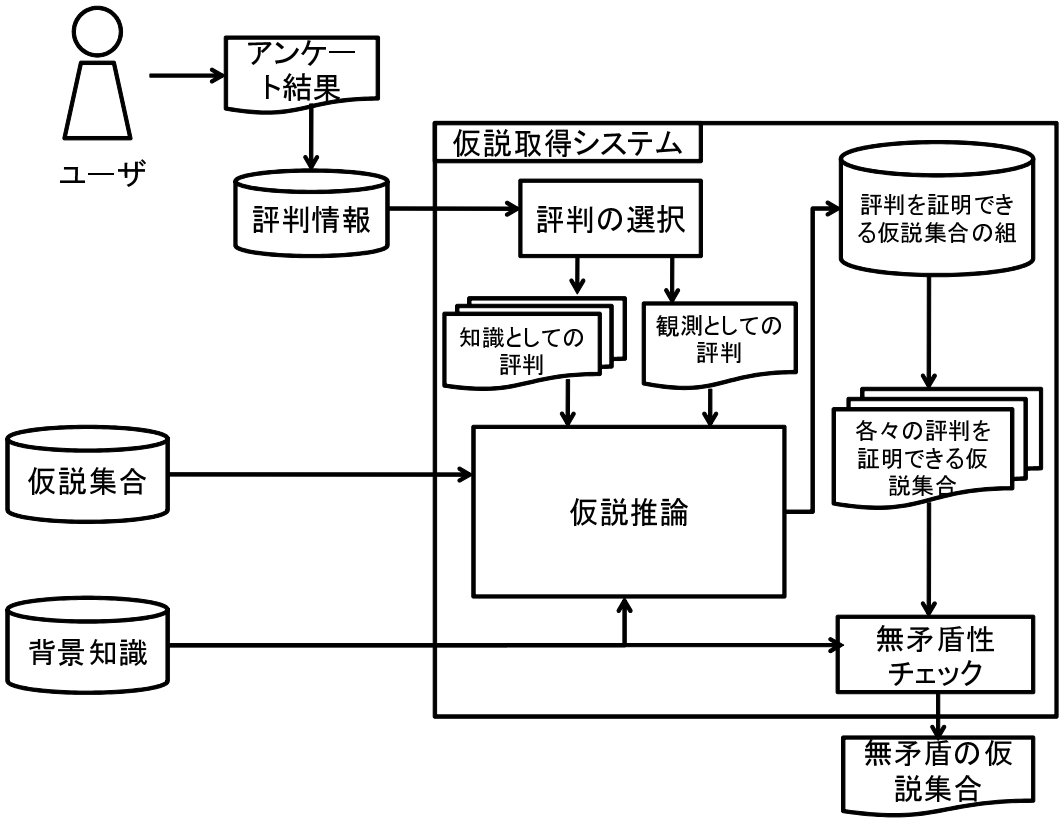

図 3: 仮説取得システムの構成図

4.2 サービス選択部

サービス選択部のシステム構成図を図 4 に示す．サービス選択部は，ユーザの問い合わせに対して起動され，ユーザにとって有用なサービスを選択するモジュールである．このシステムは一般的な論理プログラミングの枠組に基づい

ている．無矛盾かつ各々の評判情報を証明できる仮説集合は，仮説取得部から得られる出力となる．サービス選択者は，タスクを指定し，有用なサービスはどれであるかを問い合わせる．問い合わせに対し，サービス選択システムはまずユーザ情報を取得する．データベースである背景知識，推論規則，評判情報は仮説取得部と変わらない．これらのデータベースと，ユーザの問い合わせから，ユーザのプロファイルとタスク情報を取得する．次に，サービス集合から評価対象のサービスを選択し，取得されたユーザとタスクの情報を用いて，サービスに対して「ユーザ，タスク，サービス」の 3 つ組の有用性判定を行う．有用性判定を行う手法は，一般的な論理プログラミングの演繹推論に基づいている．この有用性判定を全てのサービスに対して行う．その後，有用であると判定された全てのサービスをユーザに返す．その有用なサービスを用いて，ユーザはサービスを実際に呼び出し，実行する．

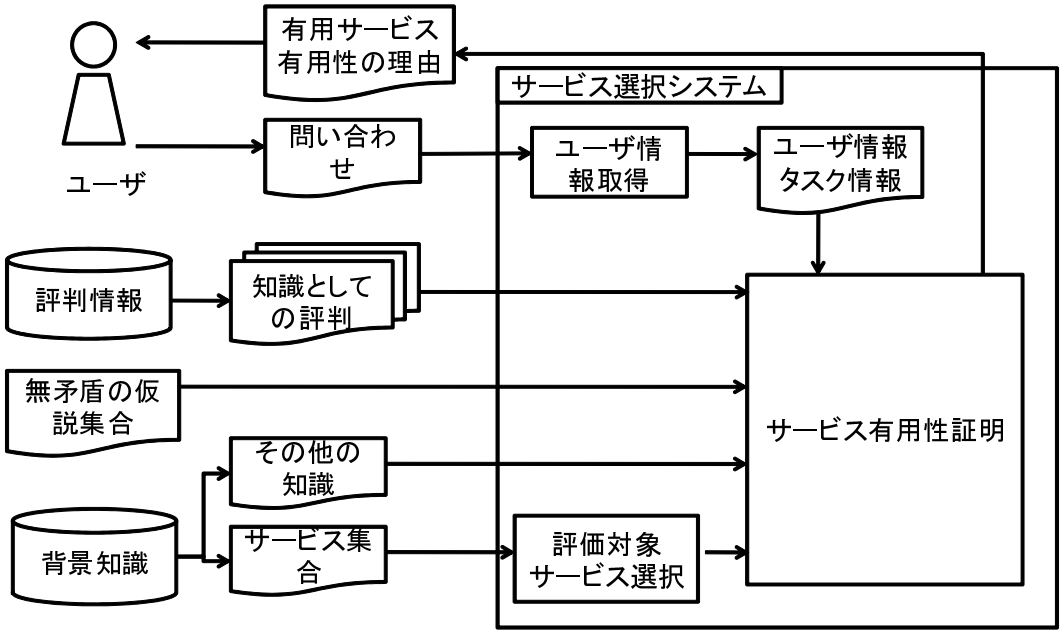

図 4: サービス選択システムの構成図

第 5 章　機械翻訳サービス推薦における応用例

これまでで説明してきた仮設推論を用いたサービス選択の枠組を実世界に応用させることを考える．今回，言語グリッド内において機械翻訳サービスの推

薦機能の実装を行った．まずは言語グリッドと Playground の概要について説明し，その後言語グリッド Playground における実際の実装例について説明する．

5.1 言語グリッド

言語グリッドとは,「辞書や機械翻訳などの言語資源を言語サービスとして登録し、共有可能にするインターネット上の多言語サービス基盤」[1] を指す．サービスコンピューティングにおいては，ラッピングされた言語サービスがそれぞれの Web サービスのコンポーネントとなっており，実際の Web ツールとしての利用法を公開している[1)]．これらのコンポーネントを組み合わせることにより,「医療領域に特化した日英翻訳サービス」などの複合サービスをコストを抑えて実現することができる．言語グリッドには全部で 91 のサービスが提供されており，日英翻訳サービスの場合単体で 5 つのサービスが存在する．専門用語辞書や形態素解析サービスと組み合わせた複合機械翻訳サービスになると，その数は大きなものになる．そのため，これらのサービスの中から有用なサービスを実際に用いる前に発見することができれば，ユーザにとって大きな助けとなる．

5.2 Playground 内での実装

今回，機械翻訳サービスの推薦機能を，言語グリッド Playground に実装した．Playground とは，ユーザコミュニティによってそれぞれ異なる多言語環境において，ユーザ自身が容易にカスタマイズすることのできるインターフェースである [12]．Playground で提供しているサービスとして，辞書連携翻訳，ユーザによる対訳辞書作成，形態素解析などがある．ユーザはこれらのサービスを組み合わせたり，サービスに対するインターフェースを拡張することで，自由に言語サービスを作成，利用することが可能となる．これらが実際に使われている場面として，川崎市や富士見中学校での活用例がある．今回は，機械翻訳サービスの取得及び実行に Playground のインターフェースを利用した．本研究における実装では，評判情報の取得から仮説取得システムまでの実装は，ローカルで実行さる．仮説推論本体の実装には PrologICA を用いた．PrologICA は Prolog によって実装された仮説推論システムのための拡張であり，知識を記述するこ

[1)] Language Grid Service Manager : http://langrid.org/service_manager/overview

とのみで仮説推論の実行を可能にする [13].

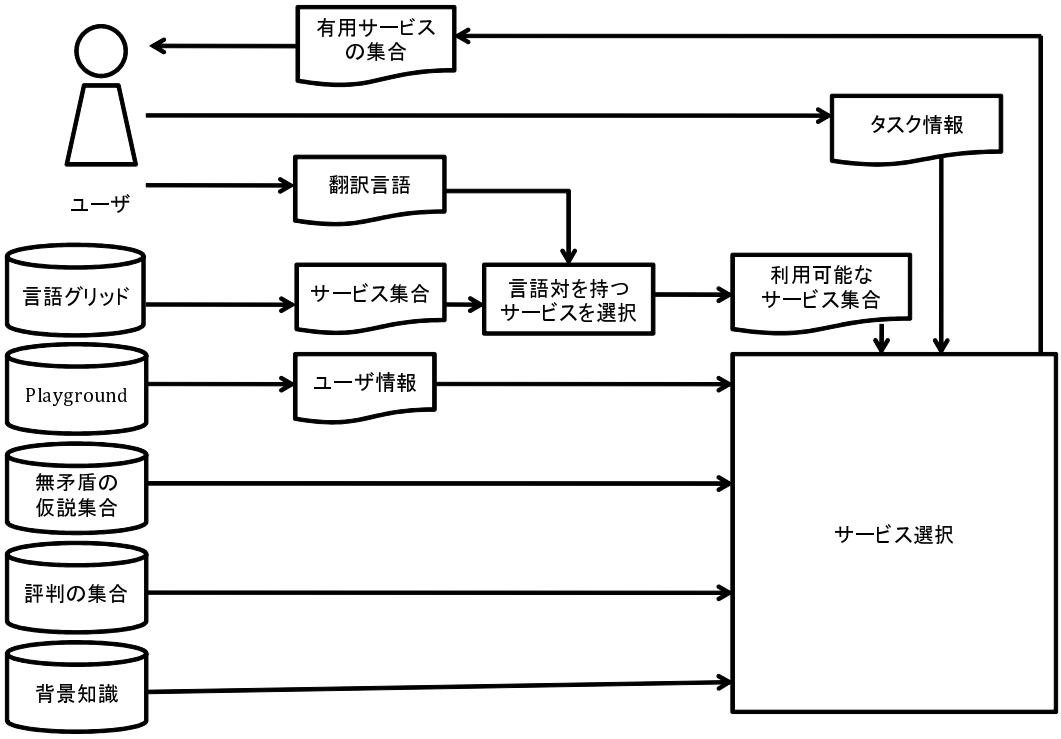

図 5: Playground における実装のシステム構成図

実装したシステム構成図を図 5 に示す．第 4 章で説明した一般的なサービス選択部といくつかの部分で違いがある．まず，Playground ではユーザのログイン情報を保持するので，ユーザ情報は Playgroud から取得している．次に，演繹推論でサービス選択を行う前にユーザの翻訳しようとした言語が翻訳できる言語対であるかどうかを確認するため，全てのサービスからユーザの指定した言語対で翻訳可能なサービスのみを演繹推論によるサービス選択の入力とする．また，背景知識については，サービス集合が言語グリッドから呼び出されるため，推論規則のみが記述されている．他の，無矛盾の仮説集合，評判情報の集合については第 4 章で説明したものと同じである．Playground で追加された処理に，言語対を持つサービスを選択，という処理がある．これは，言語グリッドから提供されるサービスの情報を用い，ユーザが指定した言語対を翻訳可能言語対に持つ言語サービスを出力する処理である．もう一つ追加された処理に，評判情報の入力という部分がある．これは，推薦されたサービスを利用した後，ユーザがタスクとサービスに対して実際に有用であったかを入力し，Playground の

サーバに保存する処理である．この処理は，サービス推薦機能を用いることなく機械翻訳を実行したユーザも用いることができる．

5.3 評価

Playground には，現在自動でサービス選択を行うシステムは用意されていない．そこで，本研究で用いた枠組を用いることにより，どのサービスが有用であるか分からない場合でも有用なサービスを選択することができると考えられる．

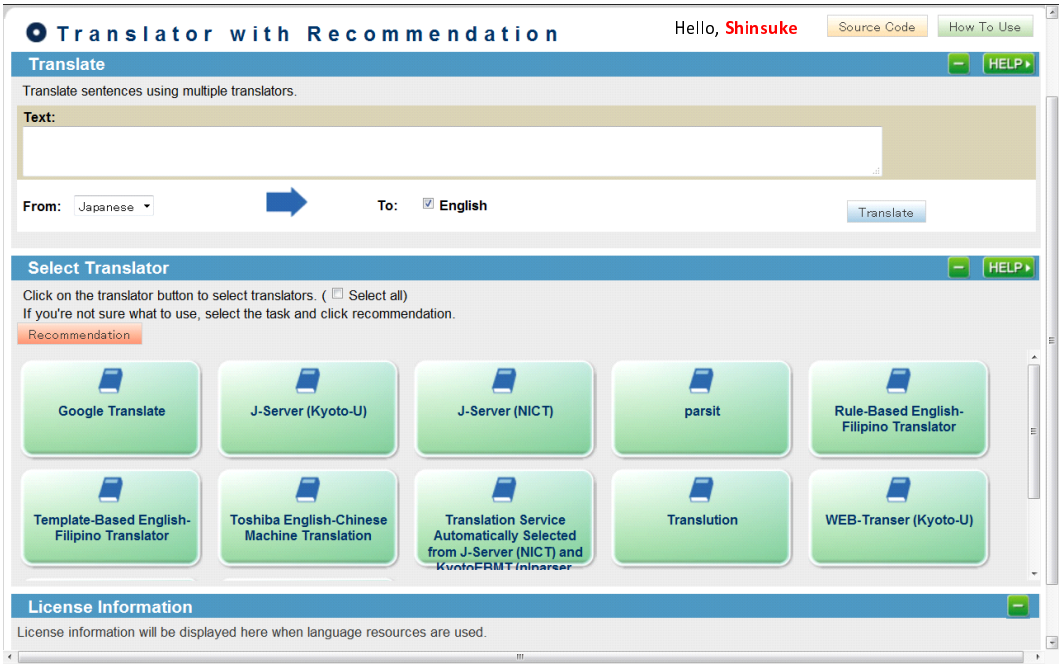

図 6: サービス推薦システム：初期画面

図 6，図 7 は，機械翻訳サービス推薦システムの実際の画面である．まず，機械翻訳サービスの一覧が図 6 に表示されている．この状態から，機械翻訳サービスと翻訳言語対を選択すれば，翻訳サービスの呼び出し，実行を行うことができる．しかし，機械翻訳サービスの種類は多く，初めて訪れたユーザはどのサービスを利用すればよいのか分からない．そこで，タスクおよび翻訳言語を指定し，Recommendation ボタンを押すとサービス推薦が実行される．推薦を行ったサービスで翻訳を行った後，ユーザは翻訳結果に対して Useful, Useless の評価を行うことが可能になっている．また，サービス推薦後に表示される Reason ボタンをクリックすると，有用なサービスが有用であると示された理由が推論

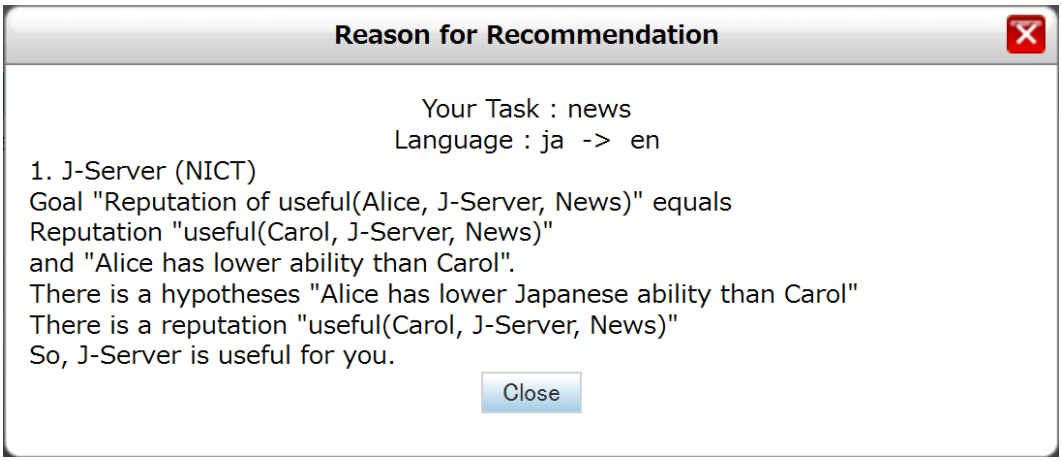

図 7: サービス推薦システム：推薦理由表示画面

規則を用いて表示される．図 7 は，Alice がニュースをタスクとして選択した際，J-Server が有用であることを示している．推論理由として，Alice にとってニュースのタスクを行う際に J-Server が有用であることは，推論規則から「Carol が J-Server をニュースに使ったときに有用である」かつ「Alice は Carol より言語能力が低い」ということと同義である．そして，仮説ベースに「Carol が J-Server をニュースに使ったときに有用である」があり評判として「Alice は Carol より言語能力が低い」があることから，Alice にとってニュースのタスクを行う際に J-Server が有用であることが推論できると示している．

今回の研究の枠組を用いてサービス推薦を行った利点として，以下の 2 点が挙げられる．まず，定量的な QoS 評価を行わなくても，サービスの推薦を行うことができるという点である．コストが莫大になる言語サービスの定量的な評価を行うことなく，それぞれのユーザにとって有用なサービスを示すことに成功している．また，一般的な手法で自分でサービスを選択し，翻訳を行った場合でも有用性についてのユーザ視点での評価を入力することにより，後にサービス推薦機能を使ってサービスの選択を行う際に役立てることが可能になっている．

もう一つの利点は，QoS だけでなく，それぞれのユーザ，タスク間の順序関係に関する仮説を用いてサービスの推薦理由を示すことができているという点である．ユーザにとって，QoS の数値によるサービスの評価というものはサービスの実際の有用性にはつながらない．そこで，サービスが有用である理由を，他のユーザの評判情報を用いた推論機構によって明らかにし，ユーザが必要で

ある場合，どのサービスがどういった理由で有用であるかを見ることができる．

これらの利点は，機械翻訳サービスに限らず，タスクによってサービスの難易度が異なり，主観的な評価にコストが大きくかかるサービスにおいて応用可能である．例えば，対訳辞書サービスにおいては，語彙の量や，収録されている単語の種類によってサービスの有用性が異なる．仮説推論を用いたサービス選択の枠組を対訳辞書サービスに実装すれば，タスクに応じた対訳辞書の有用性が示されることとなる．

第6章　おわりに

サービスコンピューティングが今後さらに発展していくために，サービスを自動評価する手法は必要不可欠である．今後も人手による評価のみに頼り続けた場合，サービスの爆発的な増加に人手の評価が追いつかない，という事態が発生する可能性がある．

本研究では，厳密な QoS 評価の代わりに，評判情報を用いたサービス有用性判定の枠組を提案した．そして，ユーザの評判情報の持つ問題である，ユーザ間の矛盾とデータの不足の両者に対応する手法として仮説推論が有効に作用することを示した．

仮説推論を用いた評判情報からの無矛盾な仮説集合，及び仮説集合と評判情報を用いたサービス選択により，QoS を調査することなく，サービスを選択することができる．評判情報はサービスがタスクに対してユーザの能力より有能にこなすことができるかどうか，という定義のもとに，サービス，ユーザ，タスクそれぞれの順序関係を求めることでサービス評価の指標を取得した．取得した順序関係は，演繹推論によってそれぞれのサービスおよびユーザの指定したタスクに対してサービスが有用であるかどうか，の判定を行った．

本研究の貢献は以下の 2 点である．

評判情報に基づくサービス選択手法の定式化

仮説推論を用いたアプローチをとることにより，サービス選択に必要な順序関係を取得する手法を仮説推論の枠組を用いて定式化した．また，順序関係から有用なサービスを選択するアルゴリズムの提案を行った，

サービス選択のための統合アーキテクチャ

仮説推論を用いた評判情報からの仮説取得エンジンと，ユーザの問い合わ

せによって実行されるサービス選択の統合アーキテクチャを提案した．これらは言語サービスだけでなく，タスクごとの評価にコストがかかり，ユーザの評価が能力によって異なる場合のサービスにおいて適用可能となる．

今後，実世界において実際に評判情報に基づくサービス選択手法を適用させることを考えた場合，他の手法によるサービス選択との比較が必要となる．評判情報を用いる手法は人手による厳密な評価に比べて精度が落ちることが予想されるため，どれくらい厳密な評価に近いサービス選択ができるのかを調査する必要がある．また，この研究の拡張として，複合サービスに対するサービス選択が考えられる．サービスコンピューティングの一つの利点が，サービス間の連携の容易化であるため，複合サービスを評価する手法は重要となる．評判情報に基づく評価を複合サービスに拡張させた場合，新たな推論規則などの提案が必要とされるだろう．

謝辞

本研究を行うにあたり，熱心なご指導，ご助言を賜りました石田亨教授に深謝申しあげます．また，日頃より時間を惜しまず様々なご助言とご協力をいただきました独立行政法人 情報通信研究機構 研究員 村上陽平博士に感謝申しあげます．最後に，普段からお世話になっている石田・松原研究室の皆様に心より感謝いたします．

参考文献

[1] Ishida, T.: Language grid: an infrastructure for intercultural collaboration, *Applications and the Internet, 2006. SAINT 2006. International Symposium on*, pp. 5 pp. –100 (2006).

[2] Fuji, M., Hatanaka, N., Ito, E., Kamei, S., Kumai, H., Sukehiro, T., Yoshimi, T. and Isahara, H.: Evaluation Method for Determining Groups of Users Who Find MT Useful, *MT Summit VIII: Machine Translation in the Information Age*, pp. 103–108 (2001).

[3] Koehn, P. and Monz, C.: Manual and automatic evaluation of machine translation between European languages, *Proceedings of the Workshop on Statistical Machine Translation*, StatMT '06, Stroudsburg, PA, USA, As-

sociation for Computational Linguistics, pp. 102–121 (2006).

[4] *Services Computing*, Springer Berlin Heidelberg (2007).

[5] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. and Sheng, Q. Z.: Quality driven web services composition, *Proceedings of the 12th international conference on World Wide Web*, WWW '03, New York, NY, USA, ACM, pp. 411–421 (2003).

[6] Liu, Y., Ngu, A. H. and Zeng, L. Z.: QoS computation and policing in dynamic web service selection, *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, New York, NY, USA, ACM, pp. 66–73 (2004).

[7] Bramantoro, A. and Ishida, T.: User-Centered QoS in Combining Web Services for Interactive Domain, *Semantics, Knowledge and Grid, 2009. SKG 2009. Fifth International Conference on*, pp. 41 –48 (2009).

[8] Poole, D., Goebel, R. and Aleliunas, R.: *Theorist: A logical reasoning system for defaults and diagnosis*, Springer-Verlag, chapter 13, pp. 331–352 (2011).

[9] Reiter, R.: *On closed world data bases*, Morgan Kaufmann Publishers Inc., pp. 300–310 (1987).

[10] Baldoni, M., Baroglio, C., Martelli, A. and Patti, V.: Reasoning about interaction protocols for customizing web service selection and composition, *Journal of Logic and Algebraic Programming*, Vol. 70, No. 1, pp. 53 – 73 (2007). Web Services and Formal Methods.

[11] Sohrabi, S., Prokoshyna, N. and McIlraith, S.: Web Service Composition Via Generic Procedures and Customizing User Preferences, *The Semantic Web - ISWC 2006* (Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M. and Aroyo, L.(eds.)), Lecture Notes in Computer Science, Vol. 4273, Springer Berlin / Heidelberg, pp. 597–611 (2006).

[12] Sakai, S., Gotou, M., Murakami, Y., Morimoto, S., Morita, D., Tanaka, M. and Ishida, T.: Language grid playground: light weight building blocks for intercultural collaboration, *Proceeding of the 2009 international workshop on Intercultural collaboration*, IWIC '09, New York, NY, USA, ACM, pp. 297–300 (2009).

[13] Ray, O.: ProLogICA: a practical system for Abductive Logic Programming, *in Proceedings of the 11th International Workshop on Non-monotonic Reasoning*, pp. 304–312 (2006).

付録：ソースコード

本研究の実装において，仮説取得システム，及びサービス推薦システムの実装には java を用いた．推論システムには Prolog を用い，SWI-Prolog における Prolog-java インターフェースを用いて両者を実装した．PrologICA の仮説推論を呼び出す関数は demo であり，demo は第一引数を観測情報として第二引数に仮説を証明できる仮説集合を代入する処理である．

A.1　仮説取得システムのソースコード

仮説取得システムは，バッチコマンドで Main.java を複数回実行させることによって，テキストファイルとして仮説集合を取得している．

ソースコード 1: Main.java

```
package abduction;

import java.io.*;
import java.util.ArrayList;

import jpl.Term;

public class Main {

   /**
    * @param args reputationsFileName, abductionfilename, repsize,
      filename
    */
   public static void main(String[] args) {

      Reputation repSet = new Reputation(args[0]);
      ArrayList<String> rep = repSet.getAllRep();
      ReputationProver prover1 = new ReputationProver(repSet,rep.
        get(Integer.valueOf(args[2])),args[1]);
      ArrayList<Term> hypotheses = prover1.getHypotheses();
      try {
         BufferedWriter buf = new BufferedWriter(new FileWriter(
           args[3]));
         for(Term hyp: hypotheses){
            System.out.println(hyp);
            buf.write(hyp.toString());
         }
      } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }

}
```

ソースコード 2: Reputation.java

```
package abduction;


import java.io.*;
import java.util.ArrayList;



public class Reputation {
    private int repSize;
    private ArrayList<String> str;
    public Reputation(String filename){
        try {
            BufferedReader rep = new BufferedReader(new FileReader(new
                File("../conf/"+filename)));
            String line = null;
            int lineNum =0;
            this.str = new ArrayList<String>();
            while((line = rep.readLine())!= null){
                lineNum++;
                this.str.add(line);
            }
            this.repSize = lineNum;



        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public ArrayList<String> getAllRep() {
        return this.str;
    }
```

```
    public int getRepSize() {
        return this.repSize;
    }


}
```

ソースコード 3: ReputationProver.java

```
package abduction;

import java.util.ArrayList;
import java.util.Hashtable;

import jpl.*;

public class ReputationProver {
    private ArrayList<Term> hypotheses;
    @SuppressWarnings("unchecked")
    public ReputationProver(Reputation repSet, String reputation,
      String abd){
        consult("../conf/alp.swi");
        this.hypotheses = new ArrayList<Term>();
        //決めうちでもいいのかも.仮説推論本体
        consult("../conf/"+abd);
        //abducible_predicatesの導入
        consult("../conf/hypotheses.swi");

        consult("../conf/knowledge.swi");



        ArrayList<String> reps = repSet.getAllRep();
//      Compound facts = new Compound("facts",repSet.getRepSize()-1);
        Compound[] facts = new Compound[repSet.getRepSize()-1];
        int i =0;
        for(String rep :reps){
            if(!rep.equals(reputation.replace("[],", "")) && rep !=
              null){
                facts[i] = (Compound) jpl.Util.textToTerm(rep.replace
                  (".", ""));
```

```
                i++;
            }
        }
//      Atom[] temp = {new Atom(reputation.replace(".", ""))};
        Compound[] temp = new Compound[] {(Compound)jpl.Util.
          textToTerm(reputation.replace(".", ""))};

//      Compound obs = new Compound("obs",temp);
        jpl.Variable D = new jpl.Variable("D");
        Term[] arg = {jpl.Util.termArrayToList(facts),jpl.Util.
          termArrayToList(temp),D};
        jpl.Query query = new jpl.Query("abduction",arg);
        System.out.println(query);
//      if(!query.hasMoreElements()){
        if(!query.hasSolution()){
            System.out.println("no answer");
            return;
        }else{
            while(query.hasMoreElements()){
                Term ans = ((Hashtable<String,Term>)query.nextElement
                  ()).get("D");
                this.hypotheses.add(ans);
            }
        }
//      Hashtable<String,Term>[] solution = query.allSolutions();
//      for(Hashtable<String,Term> hyp : solution){
//          System.out.println(hyp.get("D").toString());
//          this.hypotheses.add((Compound)hyp.get("D"));
//      }
//      while(query.hasMoreElements()){
//          System.out.println("test");
//          Term ans = ((Hashtable<String,Term>)query.nextElement()).
  get("D");
//          System.out.println(ans);
//          this.hypotheses.add(ans);
//      }


    }
    public ArrayList<Term> getHypotheses() {
        return this.hypotheses;
    }

    private static boolean consult(String dcgPath) {
        if (dcgPath == null) {
```

```
            System.out.println("file not found");
            return false;
        }
        jpl.Query query = new jpl.Query("consult('" + dcgPath +
          "')");
        boolean isSucess = query.hasSolution();
        return isSucess;
    }


}
```

ソースコード 4: ConsistentChecker.java

```
package abduction;

import java.util.ArrayList;
import java.util.Arrays;

import jpl.Term;
import jpl.Variable;

public class ConsistentChecker {
    private ArrayList<String> consistent;
    private int[] hypNum;
    private int[] hypSize;
    private int repNum;
    public ConsistentChecker(ArrayList<ReputationProver> prover) {

//      //評判
        consult("../conf/reputations.swi");

        this.consistent = new ArrayList<String>();
        hypNum = new int[prover.size()];
        repNum = prover.size();
        int i;
        for(i=0;i<prover.size();i++){
            hypNum[i]=0;
        }
        hypSize = new int[prover.size()];
        for(i=0;i<prover.size();i++){
            hypSize[i] = prover.get(i).getHypotheses().size()-1;
        }
        ArrayList<Term> hyp = new ArrayList<Term>();
```

```
    //直積をとって追加
    do{
        Term[] temp = new Term[0];
        for(int j=0;j<prover.size();j++){
            Term cur = prover.get(j).getHypotheses().get(this.
              hypNum[j]);
            System.out.println("hypothesis:"+cur);
            temp = this.addhyp(temp, cur);
            }
        hyp.add(jpl.Util.termArrayToList(temp));
    }while(updateNum());
    for(Term check : hyp){
        System.out.println(check);
        if(check_consistency(check)){
            this.consistent.add(check.toString());
        }
    }
}



//一貫性制約に反していないかを調べる
private boolean check_consistency(Term check) {
    Variable D = new Variable("D");
    Term[] term = {check,D};
    jpl.Query query = new jpl.Query("demo",term);
    return query.hasSolution();
}




private boolean updateNum() {
    return updateNum(0);
}




private boolean updateNum(int i) {
    if(hypNum[i]<hypSize[i]){
        this.hypNum[i]++;
        return true;
    }
    else{
```

```
        if(i<repNum-1){
            this.hypNum[i]=0;
            return this.updateNum(i+1);
        }
        else return false;
    }
}



private Term[] addhyp(Term[] hyp, Term cur){
    if(cur.toString().equals("[]")){
        System.out.println("test1");
        return hyp;
    }else if(hyp.length==0){
        System.out.println("test2");
        return jpl.Util.listToTermArray(cur);
    }else{

        ArrayList<Term> temp = new ArrayList<Term>(Arrays.asList(
          hyp));
        Term[] newTerm = cur.toTermArray();
        for(Term term :newTerm){
            if(!temp.contains(term)){
                System.out.println("test3");
            temp.add(term);
            }
        }
        Term[] ret = new Term[temp.size()];
        temp.toArray(ret);
        return ret;
    }
}



public ArrayList<String> getConsistent() {
    return this.consistent;
}

private static boolean consult(String dcgPath) {
    if (dcgPath == null) {
        System.out.println("file not found");
```

```
            return false;
        }
        jpl.Query query = new jpl.Query("consult('" + dcgPath +
          "')");
        boolean isSucess = query.hasSolution();
        return isSucess;
    }


}
```

A.2 サービス推薦システムのソースコード

ソースコード 5: selection.swi

```
package service;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.regex.*;
import jpl.*;

public class Main {

    /**
     * @param args
     *             username,task
     */
    @SuppressWarnings("unchecked")
    public static void main(String[] args) {
        // Prologファイルを読み込む
        consult("./conf/service.swi");
        consult("./conf/knowledge.swi");
        consult("./conf/reputations.swi");
        consult("./conf/consistent_hyp.swi");

        // サービスの最大数の指定
        final int MAX_SERVICE_SIZE = 100;
        Variable S = new Variable("S");
        Variable F = new Variable("F");
        Term term[] = { F, new Atom(args[0]), S, new Atom(args[1]) };
        Query q = new Query("query", term);
        // 有用なサービスがなければ最も
          QoSの高いと推定されるサービスだけによる出力
```

```
if (!q.hasSolution()) {
   Variable S1 = new Variable("S1");
   Variable S2 = new Variable("S2");
   Term param[] = { S1, S2 };
   Query qos = new Query("high_QoS", param);
   int[] good_count = new int[MAX_SERVICE_SIZE];
   int count = 0;
   ArrayList<String> serviceName = new ArrayList<String>();
   while (qos.hasMoreElements()) {
      String betterService = ((Hashtable<String, Compound>)
        qos
            .nextElement()).get("S1").toString();
      if (!serviceName.contains(betterService)) {
         count++;
         serviceName.add(betterService);
         good_count[count] = 1;
      } else {
         good_count[serviceName.indexOf(betterService)]++;
      }
   }
   // 数が最大のものを取得及び標準出力
   int max = argMax(good_count, serviceName.size());
   System.out.println("service:" + serviceName.get(max));
   System.out.println("reputation:We have no appropriate
     service."
         + " So the service of the best QoS is selected.");
}
// 有用なサービスがあれば
else {
   ArrayList<Term> slist = new ArrayList<Term>();
   ArrayList<Term> flist = new ArrayList<Term>();
   while (q.hasMoreElements()) {
      Hashtable<String, Compound> hash = ((Hashtable<String,
        Compound>) q
            .nextElement());
      // サービスおよび推論理由を取得
      Term bound_to_s = (Term) hash.get("S");
      Compound bound_to_f = (Compound) hash.get("F");
      // 有用なサービスをリストに追加
      if (!slist.contains(bound_to_s)) {
         slist.add(bound_to_s);
         flist.add(bound_to_f);
      }
   }
   // サービス間の仮説出力
```

```
        Variable S1 = new Variable("S1");
        Variable S2 = new Variable("S2");
        Term param[] = { S1, S2 };
        Query qos = new Query("high_QoS", param);
        String betterService;
        int[] order = new int[slist.size()];
        while (qos.hasMoreElements()) {
            betterService = ((Hashtable<String, Compound>) qos
                    .nextElement()).get("S1").toString();
            for (int i = 0; i < slist.size(); i++) {
                if (betterService.equals(slist.get(i).toString())) {
                    order[i]++;
                }
            }
        }
        // high_QoSの第一引数となっている回数が多い順に出力
        for (int i = 0; i < slist.size(); i++) {
            int max = argMax(order, slist.size());
            // サービス及び評価理由の表示
            System.out.println("service:" + slist.get(max));
            System.out.println(getRepMessage(flist.get(max).
              toString()));
            System.out.println(getHypMessage(flist.get(max).
              toString()));
            System.out.println(getRuleMassage(flist.get(max).
              toString()));
            order[max] = -1;
        }

    }

}

// int型配列から, 最大の値を持つ添字を返す
private static int argMax(int[] array, int size) {
    int temp = -100;
    int argTemp = -1;
    for (int i = 0; i < size; i++) {
        if (temp < array[i]) {
            temp = array[i];
            argTemp = i;
        }
    }
    return argTemp;
}
```

```
// 推論規則のメッセージを表示させる
private static String getRuleMassage(String reason) {
   Pattern hypPattern = Pattern
         .compile("(high_QoS|easy_task|high_ability)\\((.+?),
           (.+?)\\)");
   Matcher hypM = hypPattern.matcher(reason);
   if (hypM.find()) {
      if (hypM.group(1).equals("high_QoS")) {
         return "rule:Using inference Rule \"The higher is the
           QoS, the more useful the service\"";
      } else if (hypM.group(1).equals("easy_task")) {
         return "rule:Using inference Rule \"The easier is the
           task, the more useful the service\"";
      } else if (hypM.group(1).equals("high_ability")) {
         return "rule:Using inference Rule \"The lower is the
           user\'s ability, the more useful the service\"";
      }

   }
   return null;
}

// 用いた仮説のメッセージを表示させる
private static String getHypMessage(String reason) {
   Pattern hypPattern = Pattern
         .compile("(high_QoS|easy_task|high_ability)\\((.+?),
           (.+?)\\)");
   Matcher hypM = hypPattern.matcher(reason);
   if (hypM.find()) {
      if (hypM.group(1).equals("high_QoS")) {
         return "hypothesis:There is a hypotheses \"" + hypM.
           group(2)
               + " has higher QoS than " + hypM.group(3) + "\"";
      } else if (hypM.group(1).equals("easy_task")) {
         return "hypothesis:There is a hypotheses \"" + hypM.
           group(2)
               + " is easier task than " + hypM.group(3) + "\"";
      } else if (hypM.group(1).equals("high_ability")) {
         return "hypothesis:There is a hypotheses \"" + hypM.
           group(3)
               + " has lower English ability than " + hypM.group
                 (2)
               + "\"";
      }
```

```
        }
        return null;
    }

    // 用いた評判のメッセージを表示させる
    private static String getRepMessage(String reason) {
        Pattern repPattern = Pattern.compile("(useful|useless
          )(\\(.+?\\))");
        Matcher repM = repPattern.matcher(reason);
        if (repM.find()) {
            return "reputation:Reputation of " + repM.group(2) + " was
               "
                    + repM.group(1);
        }
        return null;
    }

    // prologのファイルを読み込む
    private static boolean consult(String dcgPath) {
        if (dcgPath == null) {
            System.out.println("file not found");
            return false;
        }
        jpl.Query query = new jpl.Query("consult('" + dcgPath +
          "')");
        boolean isSucess = query.hasSolution();
        return isSucess;
    }

}
```

A.3 Prologによる仮説推論のソースコード

ソースコード 6: reasoning.swi

```
% Declarations %
:- dynamic  abducible_predicate/1, ic/0,
            useful/3,useful/4, useless/3,useless/4,
            high_ability/2, low_ability/2,
            high_QoS/2, low_QoS/2,
            easy_task/2, difficult_task/2,
            task/1, service/1, user/1,
            consistent/0,dummy/1,
            ability_consistent/1, qos_consistent/1,task_consistent
              /1.
```

```
dummy(a).
dummy(b).
dummy(c).
dummy(d).
dummy(e).
dummy(f).

% Theory %
useful(U1,S,T) :-
   dummy(a),retract(dummy(a)),high_ability(U2,U1),useful(U2,S,T),
     assert(dummy(a)).

useful(U,S1,T) :-
   dummy(b),retract(dummy(b)),high_QoS(S1,S2),useful(U,S2,T),assert
     (dummy(b)).

useful(U,S,T1) :-
   dummy(c),retract(dummy(c)),easy_task(T1,T2),useful(U,S,T2),
     assert(dummy(c)).

useless(U1,S,T) :-
   dummy(d),retract(dummy(d)),high_ability(U1,U2),useless(U2,S,T),
     assert(dummy(d)).

useless(U,S1,T) :-
   dummy(e),retract(dummy(e)),high_QoS(S2,S1),useless(U,S2,T),
     assert(dummy(e)).

useless(U,S,T1) :-
   dummy(f),retract(dummy(f)),easy_task(T2,T1),useless(U,S,T2),
     assert(dummy(f)).

% Integrity Constraints %
ic :- high_ability(U,U).
ic :- high_QoS(S,S).
ic :- easy_task(T,T).

%一貫性
ic :- high_QoS(S1,S2),high_QoS(S2,S1).
ic :- easy_task(T1,T2),easy_task(T2,T1).
ic :- high_ability(U1,U2),high_ability(U2,U1).

%推移律
ic :- high_QoS(S1,S2),high_QoS(S2,S3),high_QoS(S3,S1).
```

```
ic :- easy_task(T1,T2),easy_task(T2,T3),easy_task(T3,T1).
ic :- high_ability(U1,U2),high_ability(U2,U3),high_ability(U3,U1).


%facts and observations must be list!
abduction(Facts,Observations,D) :-
    assert_dummy,
    assert_all(Facts),
    print_all(Facts),nl,
    demo(Observations,D),
    print_consistent(D),
    assert_dummy;
    retract_all(Facts),fail.

print_all([X|Y]) :-
    write(X),nl,print_all(Y).
print_all([]).

assert_dummy :-
    retractall(dummy(_)),
    assert(dummy(a)),assert(dummy(b)),assert(dummy(c)),assert(dummy(
      d)),assert(dummy(e)),assert(dummy(f)).

assert_all([X|Y]) :-
    asserta(X),assert_all(Y).
assert_all([]).

retract_all([X|Y]) :-
    retract(X),retract_all(Y).
retract_all([]).

print_hypotheses :-
    write('Hypotheses are:'),nl,
    abducible_predicate(X),write(X),nl,fail;true.

print_consistent(Delta) :-
    write('Consistent hypotheses set:    '), write(Delta),nl.
```

ソースコード 7: hypotheses.swi

```
abducible_predicate(high_QoS).
abducible_predicate(easy_task).
abducible_predicate(high_ability).
```

ソースコード 8: knowledge.swi

```
task(news).
task(chat).

service('NICTJServer').
service('GoogleTranslate').
service('Translution').

user(userA).
user(userB).
user(userC).
```

ソースコード 9: reputation.swi

```
useful(userA,'Translution',chat).
useful(userB,'GoogleTranslate',chat).
useful(userC,'NICTJServer',news).
useless(userC,'GoogleTranslate',chat).
useless(userC,'Translution',news).
useless(userB,'Translution',chat).
```