ユビキタス環境で動的にサービス実現するためのサービス合成技術

山登庸次[†] 中辻 真[†] 須永 宏[†]

ユーザ状況に応じてサービスを実現するユビキタスサービス合成技術を提案し，検討，実装によりその有効性を示す．ユビキタス環境では，多種の機器・機能が連携した，コンテキストアウェアサービス実現が期待されている．そのためには，状況に応じてサービスコンポーネントを組み合わせるアプローチが有力である．著者らが提案するサービス合成技術は，サービスフローを意味的メタデータを用いて抽象記述し，ユーザの利用時に，適切なコンポーネントを動的に発見し，バインドして実行することで，個人個人に適切なサービスを提供することが特徴である．本稿では，サービス合成技術を提案後，サービス合成の検証必要課題として，メタデータ管理容易化，サービスフロー記述効率性，サービス実行時性能をあげ，それらを検証することで，提案技術が実用可能であることを示す．

Ubiquitous Service Composition Technology for Ubiquitous Network Environments

Yoji Yamato,[†] Makoto Nakatsuji[†] and Hiroshi Sunaga[†]

We proposed ubiquitous service composition technology which enables end-users to compose context aware services easily. In ubiquitous network environments in the future, context-aware services are expected because the context-awareness is one of the most important factors in ubiquitous network. But it is insufficient in conventional service based on rigid interface design. Therefore, the flexible service composition based on user context is required. Our proposed approach is that a service composition engine discovers suitable service components from the network based on the user context and binds them dynamically in accordance with a semantic-level service flow. Through the study, implementation and perfomance measurement, we showed the effectiveness and practicality of porposed technology.

1. はじめに

IT 技術の進歩にともない，PC だけでなく，携帯電話，家電，センサなど様々な機器がネットワークにつながりユーザ生活を支援する，ユビキタス環境[1] が実現しつつある．ユビキタス環境では，これらの機器・機能が有機的に連携することで，従来は実現が困難だった，コンテキストアウェアサービス[2] が期待される．

ユーザニーズは，位置など実世界の状況に応じて動的に変化するため，従来の様にサービス提供者があらかじめサービスを作り込んでおく形でなく，その場の状況に応じてサービスコンポーネントを組み合わせてサービスを実現するサービス合成の考え方が重要である．

本研究は，サービス合成技術の研究であり，その特徴としてサービスフロー（本稿ではサービステンプレート：ST と呼ぶ）を意味的メタデータを用いて抽象記述し，ユーザの利用時に状況・嗜好に応じて適切なサービスコンポーネント（本稿ではサービスエレメント：SE と呼ぶ）を動的に発見し，バインドして実行することで，個人個人に適切なサービス提供することを目指している[3)~7)]．なお，意味的メタデータとは，サービスコンポーネントがどのように動作するか，どのような性質を持つかなどを示すメタデータである．本稿では，サービス合成技術を提案し，検証必要課題として，メタデータ管理容易化，サービスフロー記述効率性，サービス実行性能をあげ，それらを検討・実装により検証することで，提案サービス合成技術が実用可能であることを示す．

本稿は以下の構成をとる．2 章で ST を用いたサービス合成技術を提案し，検証が必要な 3 つの課題をあげる．3 章では，オントロジマッピング技術により，メタデータ管理が容易にできることを述べ，4 章で，ST による記述効率の良さを従来記述と比較で示し，5 章でサービス実行性能が問題ないことを実装したサービス合成システムの性能評価で示す．6 章で関連研究に

† 日本電信電話株式会社 NTT ネットワークサービスシステム研究所
NTT Network Service Systems Laboratories, NTT Corporation

ついてふれ，7 章でまとめを行う．

2. ユビキタスサービス合成技術提案

2.1 サービス合成前提と既存技術課題

本研究の目的は，ユーザが，個人端末やサーバに搭載されたサービス合成エンジンを用いて，ユーザ状況に応じて SE を合成し，適切なサービスを利用可能にする技術の確立である．

状況に応じた合成とは，サービスフローである ST に基づいて，ネットワークから利用可能な SE 群を発見し，その中からユーザ状況に適切な SE を選択して，バインドし実行することを意味し，適切な SE の選択は，2.4 節記述のユーザコンテキストと SE 情報のマッチングで高得点の SE を選択することで実現する．

本稿では SE に関して以下の 2 つの前提をおく．対象とするコンポーネントは Web Service（以降 WS）と Universal Plug and Play[8]（以降 UPnP）の SOAP を用いるコンポーネントとする．また，SE はどのような機能かのサービス記述（インタフェース，属性など）をネットワークに公開しているとする．WS と UPnP 以外のコンポーネントは，Java2WSDL などのツールを用いて WS に変換し，WS としてバインド・実行するとする．なお，SE のバインド・実行とは，http など SOAP の下のレイヤでコネクションを張り，SOAP レイヤでオペレーション呼び出しなどのメッセージングを行うことである．

既存技術として，個々の WS を連携させる手法である，BPEL（Business Process Execution Language for Web Services）[9] が注目を集めている．しかし，BPEL は元来 B2B が対象であり，あらかじめ利用する WS を決定し，それに合わせたサービスフローである BPEL 文書と接続定義を，BPEL 実行エンジンにデプロイする必要があるため，ユーザの状況に応じて WS を発見し，バインド・実行することは困難である．ユーザ状況に合わせたサービス合成に BPEL を適用するには，具体的には，以下の 3 つの問題がある．

- BPEL 文書には，個々の WS のポートタイプ名とオペレーション名が記述されるため，それらが完全マッチする WS にしか適用できない（業種ごとに WSDL を揃えることは大きな調整稼動が必要）．
- BPEL 実行には，BPEL 文書およびどの WS に接続するかの接続定義情報をあらかじめデプロイしておく必要があり，実行前にすでに WS は決まっている．
- BPEL はルーチンワークを想定しているため，それを解釈する BPEL 実行エンジンも，頻繁なデプロイは想定外であるため，デプロイには長い時間がかかり（数秒〜10 秒），ユーザごとに動的にデプロイするのは困難である．

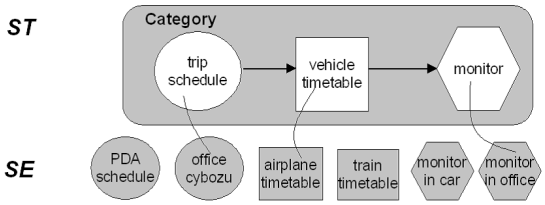

図 1 ST のイメージ図
Fig. 1 An image of Service Template.

2.2 意味的メタデータを用いた ST，SE 記述

BPEL の課題を解決するため，著者らはサービスフローである ST に個々の SE のインタフェース名を記述するのではなく，欲する機能の意味的メタデータを用いて抽象的に記述し，インタフェース名は異なるが機能は同等な SE 群を発見し選択可能にすることで，状況に応じたサービスの実現を目指している（**図 1** 参照）．図 1 の，丸，四角，六角形は意味的に同等な機能，矢印は制御の順番を示す．BPEL では，1 つ 1 つの WS を厳密に指定した制御フローを記述する必要があるが，ST では同等な SE から選択できる．

なお，本稿の主眼からは外れるが，理解補助のためサービス合成を用いたアプリケーション例として出張サポータの説明を付録 1 に添付する．必要に応じて参照されたい．

ここで，メタデータを用いて ST を記述して，サービス合成することを考えた場合，欲する機能にマッチする SE を数多く発見し利用できること（多くの候補により選択肢が増え，カスタマイズ性，耐障害性に優れる）が課題であり，以下の 3 つが必要となる．

- ST 記述者が個々の SE の詳細実装を意識せず記述できるよう，統一的形式で記述が可能であること．
- 既存コンポーネントをそのまま利用できるように，SE の記述は標準または標準候補の記述を採用すること．
- メタデータの語彙定義を統一的に定めるのは困難であるため，語彙の違いを解決し，同等機能ならば利用可能であること（例：車と car は両方利用可能）．

機能に関して補足説明をする．SE の機能とは，WSDL や UPnP doc などでネットワークにインタフェースを公開し，そのインタフェースを介して利用可能なメソッドやイベント等のオペレーションを指す．

また，同等機能の SE とは，URL やインタフェースは異なっていても同様の動作をする SE のことである（例：5F のキャノンプリンタと 6F のエプソンプリンタは URL は異なるが，動作は同様である）．

上記の 3 つの条件から，SE 記述に近年進んできた Semantic Web Services 技術（文献 10)，11) など）の OWL-S（Web Ontology Language for Services）[12] を用いて設計，実装を行った．OWL-S は，3 つの記述からなり，Profile 記述はどのようなサービスを提供するかを記述し，Process 記述はどのように動くかの抽象的 Process の入出力や条件，効果を記述し，Grounding 記述はどのようにアクセスするかの抽象 Process と実際のオペレーションのマッピングを記述する．

OWL-S を利用する理由を説明する．まず，実際のオペレーションにマッピングされる Process は，UPnP，WS に中立であり，ユーザは Process を実際のオペレーションのメタデータとして使うことで，SE の詳細実装を意識せず，ST を記述できる．次に，OWL-S は標準候補としてあがっており，将来的には OWL-S で記述されたサービスがネットワークに遍在することが予想されている．また，語彙違いを解決するために次節で提案する，メタデータ関係を OWL（Web Ontology Language）[13] で記述し該リンクをたどって解決する方式と親和性が高い．

このような理由から，SE は OWL-S をネットワークに公開することとし，それに対応して ST は以下のように記述する．ST は利用する SE の機能を，カテゴリ名（同等 SE 機能をカテゴライズした単位）とオペレーションメタデータで指定し，指定した各カテゴリ間の情報引継ぎをあわせて記述する．ここで，オペレーションメタデータは，OWL-S の Process とそのパラメータにあたる．カテゴリ間の情報引継ぎには，前のカテゴリのパラメータを，別カテゴリのどのパラメータに入れるかのマッピングが記述される．

ユーザがサービスを利用する際は，サービス合成する合成エンジンに ST を入力すると，合成エンジンが，ST に記述されたメタデータを用いて同等な SE 群を検索し発見する．次に発見された SE 群の Profile とユーザコンテキストを用いて点数付けをしてユーザ状況に適切な SE を選択する．利用する SE が決定された段階で，合成エンジンは，選択した SE の Grounding 記述を用いて SE の実際のインタフェース（WSDL や UPnP）に変換して，SE をバインド・実行する．**図 2** は，メタデータから実際のオペレーションに変換する過程を示した図である．なお，同等機能でも異なる語彙で定義されたメタデータが振られている場合（例：車と car など）は，次節で提案するカテゴリツリーのリンクをたどり語彙変換することで同等機能として発見可能である．語彙変換により，代替候補 SE 数を増やし，発見の柔軟性を確保することができる．

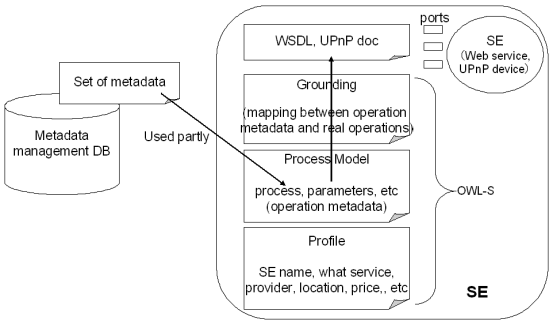

図 2 SE 記述への OWL-S の利用
Fig. 2 Using OWL-S for Service Element description.

2.3 語彙解決するためのメタデータ管理方法

メタデータの管理は，ST 記述者，SE 作成者，ユーザのすべてが利用しやすい形でなければならない．そのため，以下の 3 つが必要となる．

- ST 記述者が，記述したい ST に必要なオペレーションメタデータをすぐに発見，選択が可能．
- SE 作成者が，新機能を作成したときに，新たなオペレーションのメタデータを追加可能．また，異なる語彙のメタデータの関係を登録することも可能．
- ユーザが，ST に基づいて SE を検索する際，欲する機能と同等機能を持つ SE はすべて検索可能．

これらを解決するため，以下の 3 つを特徴としたメタデータ管理方式を検討し実装した．

- それぞれの目的ごとにオペレーションメタデータ群をカテゴライズして管理．
- 新しい機能を加えた新カテゴリを登録可能．異なる語彙で定義したカテゴリ間の関係を記述可能．
- カテゴリ間で，メタデータの関係（同値，継承関係）を双方向にリンク．

管理方式を**図 3** を用いて説明する．ここで，オペレーションメタデータ群を目的ごとにカテゴライズしたものをカテゴリと呼ぶ（例：ビデオ，旅行予約カテゴリなど）．図 3 の長方形はカテゴリ，小文字は該カテゴリに属するオペレーションメタデータ，矢印はメタデータの関係，を示すものである．

たとえば，カテゴリ A にいくつかのオペレーションメタデータが定義されているが，新しい機能を作った SE 作成者が C というカテゴリを作り，新たなオペレーションメタデータを追加し，既存カテゴリとの

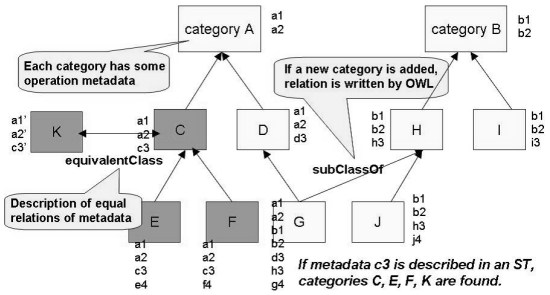

図 3 メタデータ管理方法
Fig. 3 Management method of metadata.

継承関係を記述する．これが繰り返されることで，カテゴリツリーが形成される．カテゴリの継承関係だけでなく，異なる語彙のメタデータの同値関係も記述できる（カテゴリ C と K）．ST 作成者は，メタデータ管理 DB に登録されたメタデータを参照して ST を記述する．また，合成エンジンが，SE を検索する際は，図 3 で，ST に必要な機能としてメタデータ c3 が記述されている場合は，リンクをたどり，c3 か c3' を持つ C，E，F，K というカテゴリにマッピングしている SE が検索される．

このように，SE 作成者がメタデータを追加可能にすることで，業界や目的ごとにインタフェースを完全共通化する作業を行う必要がない．なお，ルートとなるカテゴリは，誰かが最初に作成する必要があるが（travelXML [14] などの共通化された最小限のメタデータセット利用を想定），SE 作成者が新たな機能を追加してくれるため，自然選択によってユーザに使いやすいカテゴリが普及していくことが期待される．

提案方式実装においては，オペレーションメタデータに OWL-S の Process を用い，メタデータの同値関係や継承関係を OWL の equivalentClass，subClassOf プロパティを用いて記述し，これらが記述された OWL ファイルを XML DB で保持する形とした．運用方法としては，同値や継承や新規登録を，SE 作成者などから登録依頼があった際に，メタデータ管理 DB 管理者が確認後メタデータ管理 DB に登録する運用を想定している．

このように，OWL リンクをたどることで，利用できる SE 数を増やすことができるが，逆にいうと，実際は同等な機能なのに OWL リンクが引かれていなければ，発見することはできない．また，別々の人が定義するメタデータの同値関係を管理するのは大きな負担がかかる．これを解決するため，メタデータを管理するメタデータ管理 DB には，著者らが以前開発したオントロジマッピング技術[15),16)] を改良して，カテゴリ間の OWL リンクを自動で推測しリンクを引く方式を実装しており，次章で管理稼動の検証を行う．

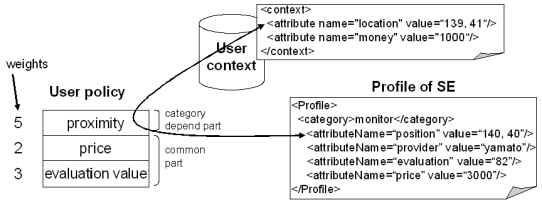

図 4 SE 点数付けのイメージ図
Fig. 4 Method of scoreing Service Element.

2.4 コンテキストに応じた SE 選択と SE 切換え

メタデータ管理 DB と Grounding により，同等機能の SE を柔軟に発見し，実際のインタフェースに変換してバインド・実行することが可能となる．本節では，発見された SE 群から，コンテキストに適した SE を選択すること，さらに，コンテキスト変更に応じて SE を切り換えることについて説明する．

2.4.1 コンテキストに応じた SE 選択

状況に応じたサービスを実現するため，利用可能な SE 群から，ユーザ状況に適切な SE を選択する必要がある．適切な SE を選択するため，どのような SE を優先するかのユーザポリシ，ユーザがどのような状況かのユーザコンテキスト，SE がどのようなプロパティを持つかの SE Profile の 3 つの組のマッチングで，SE に点数を付け，高得点の SE を自動選択またはその点数を参考に手動選択する方式の検討，実装を行った．

ユーザポリシ，ユーザコンテキスト，SE Profile は，それぞれ，いくつかのデータの重ね合わせで表される．位置を例にあげると，ユーザの位置がユーザコンテキストの一部で，SE の位置が SE Profile の一部であり，近い SE を優先したいという要望がユーザポリシの一部である．

点数付け方法を図で表すと，**図 4** のようになる．以下，図 4 を参照しながら説明を行う．

まず，ユーザポリシの説明をする．ユーザポリシは，いくつかの評価基準の重ね合わせであり，評価基準として，値段の安さや SE 評価値（ユーザ利用履歴による評価値）の高さなどの全カテゴリ共通の評価基準と，距離の近さなどカテゴリに応じた評価基準がある．ユーザは各評価基準に重み付けを設定することで，近さ重視や安さ重視など自分用のポリシを設定できる．

ユーザコンテキストは，位置などのユーザ状況を示すデータの集合であり，ユーザコンテキストファイルとしてコンテキスト保持部に保持される．なお，コンテキスト収集機能として，GPS や温度センサなどい

ろいろあるが，これらセンサ類の実装は本研究対象外とする．本研究では，それらセンサ類から情報を集めたコンテキストファイルがコンテキスト保持機能に保持されていることを前提にする．

SE Profile は OWL-S の Profile のパラメータとして記述されており，SE の位置や値段など SE のプロパティが記述されている．

各評価基準には，点数付け関数が設定されており，評価基準ごとに，SE の Profile とユーザコンテキストをマッチングすることで点数をつける．たとえば，位置の近さである「proximity」評価基準の場合は，ユーザ位置と SE 位置を用いた，exp(− (UserLatitude − SeLatitude)^2 − (UserLongitude − SeLongitude)^2) のような評価式が用いられる．

各評価基準の点数は重みに応じて重ね合わせられ，SE の総合点が出される．このように，ユーザはポリシを詳細に設定することで，状況に応じて SE の点数付けをし，SE の自動選択または，その点数を参考に手動選択ができる．なお，詳細なポリシ設定を行いたくないユーザにはデフォルトポリシを準備している．

2.4.2 コンテキスト変化に応じたサービス切換え

サービス切換えとは，一度サービス合成して実行中に，ユーザ状況変化にともない，より適切な SE が出てきた場合に，ST のサービスフローの状態（どの SE まで処理が進んだかやパラメータ値）を保ったまま，一部の SE を別 SE に自動または手動で切り換えることをいう．適切な SE かどうかは，前述の SE 点数によって判定できるので，サービス切換えは以下の形で実現できる．

①コンテキスト収集機能は，定期的にユーザコンテキスト情報を集め，コンテキスト保持機能に保持されたユーザコンテキストファイルを更新する．

②SE を点数付けする，サービス選択機能は，一定周期ごとに，最新のコンテキスト情報を用いて利用できる SE 群を点数付けし，現在利用の SE よりも良い点数の SE がある場合に，SE の点数リストを合成エンジンの制御機能に上げる．

③制御機能は，自動切換えならば自動で新しい SE に切り換え，手動切換えならばユーザに SE 切換えを提案してユーザの確認後，新しい SE に切り換える．SE 切換えが，最も複雑な場合は，新 SE の Grounding, WSDL を解析後，古い SE の状態を取得し，新しい SE にオペレーションの引数として状態やパラメータを渡して実行し，古い SE を停止し，ST フロー内の不要になったパラメータを解放する形である．これらの停止や実行など，SE とエンジン間の制御はすべて SOAP で行う．制御メッセージは SE とエンジン間のみでやりとりされ，SE どうしは，データ通信を直接行うことはあっても，制御メッセージを直接やりとりすることはない．

ただし，電話サービスでなく，WS や UPnP が対象なため，複雑な引継ぎはまれである．たとえば，ユーザ近くのモニタに定点カメラ映像を配信するサービスでは，ユーザの移動に応じて近くのモニタ SE を新 SE として選択すると，そのモニタ SE の Grounding や WSDL を解析後，ST のフロー内のパラメータとして保持しているカメラ映像配信 SE の URL を新しいモニタ SE に渡して実行し，古いモニタ SE を停止する形である．

なお，サービス切換えは，頻繁に行われると煩わしいものであるため，切り換えたくないユーザは，点数付け周期を長くしたり，周期を無限長にして行わないように設定したりすることもできる．

2.5 検証が必要な課題

以上，著者らが提案するサービス合成技術を説明した．サービス合成技術により，ST を 1 度記述すれば，ユーザの環境に応じて適切な SE が選択されるため，個々の SE に応じて毎回 ST を記述する必要がないため効率的である．また，合成サービス実行時に，ある SE より状況に適切な SE が見つかった際に，他の SE は保持したまま，該 SE だけ切り換えるサービス切換えもできるようになる．しかし，ST を抽象的に記述できる分，SE 側にメタデータを記述する必要がある．また，実行時に OWL などを解釈して実行するため性能に不安がある．そこで，サービス合成の実用性を示すためには，以下の 3 つの課題の検証が必要と考える．

- メタデータ管理容易化：ST 記述が容易になっても，異なる定義の語彙間の同値関係記述など，メタデータ管理コストが多くては仕方ないため，メタデータ管理稼動が低いことを示す必要がある．
- ST 記述効率性：ST により柔軟な合成ができることを，BPEL など従来記述と比較して，効率が良いことを示す必要がある．
- サービス実行時性能：メタデータ解釈を通した SE 発見，選択，実行は性能劣化が懸念されるため，性能が実用上問題ないことを示す必要がある．

3 章以降では，これらの課題を検証することで，サービス合成技術の実用性を示す．

3. オントロジマッピング技術によるメタデータ管理容易化

本章では，メタデータ管理容易化について検証を行

う．サービス合成技術は，OWL リンクをたどることで柔軟な合成を可能にする．しかし，同等機能でも，OWL リンクがなく，別々に定義されたメタデータは互換性がなく 1 つの ST から同じように利用はできない．また，ドメイン A，B で別々に定義されたメタデータを，双方から利用できるようにするには，メタデータ間の OWL リンクを手動記述する必要があり手間がかかる．これらの解決のため，メタデータを管理するメタデータ管理 DB に，オントロジマッピング技術[17] を利用し，カテゴリ間 OWL リンクを自動生成する方式を検討し，実装している．

3.1 オントロジマッピング用いたメタデータ関係抽出の評価

オントロジとは対象世界の概念化の明示的記述であり，オントロジマッピングとは異なるオントロジ空間間の同値関係などのマッピング情報を導出することである．著者らは，以前に NMS（Network Management System）を対象にしたマッピングエンジンを開発している[15),16)]．これは，状態パラメータのみを扱ったものであったが，サービス合成の SE（カテゴリ，Atomic Process，入出力パラメータを持つ）に利用できるよう，改良を行って実装し，メタデータ管理を容易化できることの検証を行った．開発したマッピングエンジンは，語彙のシソーラス類似度およびメタデータ間の接続形態である継承・同値関係トポロジの近似度の重ね合わせで評価点を付け，別々に定義されたメタデータ間の同値・継承関係を推測する．

もし，メタデータが乱数などでランダムに付けられていたら，語彙近似度による推測はできない．しかし，サービス合成システムにおいては，メタデータで欲する機能を指定するため，新たなメタデータを作る SE 提供者は利用されやすいようなメタデータを付けると想定するし，また，Semantic Web でもメタデータ付与に何らかのルールを作る動きも出てきている．メタデータは OWL クラスで記述されているため，同値や継承などの OWL プロパティとシソーラスにより，高精度で推測できる．

次に，メタデータ管理 DB のマッピング機能の利用方法を説明する．サービス合成は，実行時に動作が保証できる SE のみ合成することが望まれるため，ユーザの検索時にマッピングを行うのではなく，ユーザ検索がないバックグラウンドでマッピングを行う**図 5** の形とした．メタデータ管理 DB 管理者は，独立な（OWL リンクがない）2 つのカテゴリツリーの Process Model ファイル群と継承・同値関係などの OWL 記述ファイルをマッピング機能に渡す（図 5 の①）．マッピング機能

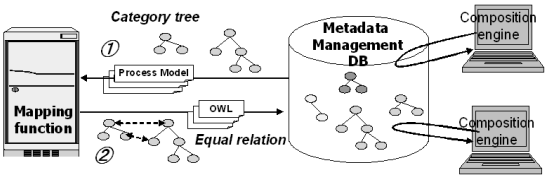

図 5 マッピング機能の利用方法
Fig. 5 An image of using mapping function.

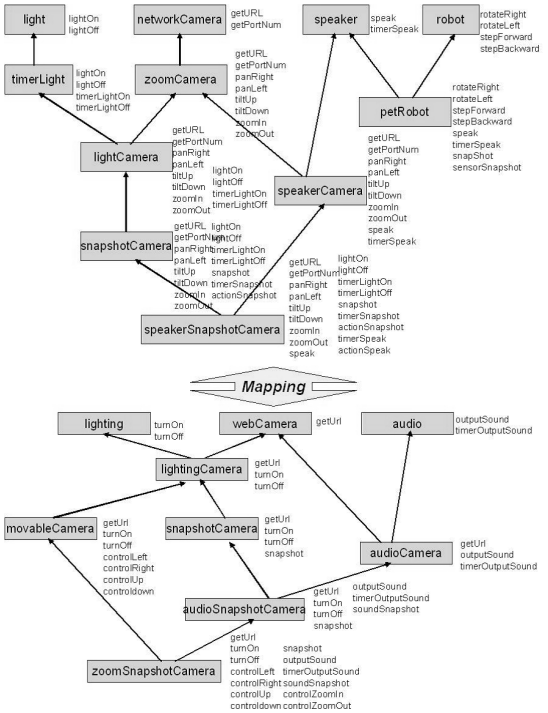

図 6 マッピングサンプルデータ
Fig. 6 Sample data of ontology mapping.

は，まず，OWL-S Process Model の Atomic Process（オペレーションのメタデータ），Atomic Process の Input，Output（入出力パラメータのメタデータ）を用いて，Atomic Process どうしの同値関係を判定し，さらに Atomic Process 群の関係からカテゴリの同値，継承関係を判定する．判定したメタデータどうしの関係はメタデータ管理 DB に返される（図 5 の②）．推定された関係は管理者が確認して，その確認後，メタデータ管理 DB に新たな OWL リンクが登録される．

実装したマッピング機能の検証のため，サンプルデータとして，サービス合成のアプリケーションに利用したネットワークカメラ関連のカテゴリツリー（**図 6**）を用いてマッピングを行った．このカテゴリツリーは，Panasonic と Sony のネットワークカメラの機能にメタデータを付与したもので異種ドメイン

のメタデータと考えてよい．検索精度の一般的評価指標である，再現率（提示正解数/全正解数）と適合率（提示正解数/全提示数）を用いて，Atomic Process のマッピング精度を調べる．機種が異なるので，正解をどう決めるかは考慮が必要であるが，カメラを右回転と右旋回など同等の意味と考えられる機能を正解として，あらかじめ正解を定めておき，マッピング機能の判定を確かめた．再現率は 184/184=100%，適合率は 184/267=68.9%であり，メタデータ同値関係推測に有効といえる．

適合率が 7 割であるため，提示される中には誤りもあるが，別々の人が定義したメタデータどうしの関係を 100%推測するのは困難なため，以下の 3 点も考慮して十分と考えている．第 1 に，提示結果に誤りがあっても，メタデータ DB 管理者が確認することで，誤りを落とすことができる．第 2 に，候補 SE を増やすためには，同等性の OWL リンクが数多く引かれることが必要だが，それに重要な再現率は 100%を示している．第 3 に，3.2 節記述のとおり，今後マッピング判定に必要な情報を増やすことでより高い適合率を得られることが強く期待できる．

サンプルデータは，Atomic Process がそれぞれ 81，42 種あるので，手動で行う場合は，81 × 42=3332 パターンに関して，Atomic Process のパラメータやコメントを見て同値かどうかを判断する必要があり，管理者に多大な稼動がかかる．しかし，マッピングエンジンにより，無関係の組合せは除かれ，適切な関係候補だけが提示され，管理者は OK/NG をつけるだけなので，メタデータ関係を記述する手間は，最低でも 267/3332=8%以下に減らすことができる．これらの OK/NG 判定を定期的に行うだけで，メタデータ管理が容易にできる．メタデータ管理 DB アップデートの頻度だが，メタデータ管理はつねに完璧である必要はなく，OWL リンクがない場合は，候補 SE 数が減少するが，実行上の大問題にはつながらないため，メタデータ管理 DB のアップデート作業は，数週間に 1 度程度を想定している．

3.2 メタデータ管理の今後の課題

今後はさらに，適合率上昇のチューニング，マッピングする 2 つのカテゴリツリーの選択方法，に取り組む．適合率に関しては，現在 Atomic Process とその入出力を用いて，マッピングを行っているが，OWL-S の事前条件，効果を利用することで，マッピング判定に用いる情報量が増えるため改善することが可能である．さらに，過去のマッピング結果の統計処理を基にマッピング誤りを除去する機構を加える予定である．カテゴリツリー選択に関しては，現在 DB 管理者が近そうなカテゴリツリーを 2 つ選んでマッピングをとる形だが，カテゴリ名近似度などを使って，自動で 2 つのツリーを選択できるようにする．

4. ST 記述効率性の比較検討

本章では，ST 記述により，従来の方式に比べて効率的に多彩なサービスを実現できることを検討する．

まず，比較対象を選定する．本研究は疎結合の連携サービスが対象であり，比較対象としては，Service Oriented Architecture 実現のための技術である，BPEL がある．実際，ST の記述は BPEL の制御タグを参考にしており，記述例を**図 7** に示すが，BPEL との差分は，大きく以下の 2 つである．第 1 に，ポートタイプをカテゴリで，オペレーションを Atomic Process で抽象指定している点で，SE の具体化は合成エンジンが実行時に行う．第 2 に，BPEL は実行中の WS 検索は想定外であるが，ST には実行中に SE を検索する独自タグの search を拡張定義している．

なお，ST 記述には，SE を指定する際に WSDL を直指定するオプション記述も可能である．たとえば，クレジット決済 SE など信頼性が高く要求される SE に関しては，WSDL 直指定で厳密に記述し，モニタやレンタカー予約などはカテゴリとして抽象指定することでユーザごとにカスタマイズすることが可能である．ただし，すべての SE を WSDL 直指定の場合は，BPEL と同じ動作になるため，状況適応性はなくなる．

また，OWL-S を記述に用いたサービス合成として，Task [19] があげられる．Task は，あるエリアに入るとそのエリアで利用可能な OWL-S コンポーネント群から，入出力がマッチする 2 つの組合せ候補がすべて提示され，ユーザは提示された組合せから手動で選択しサービス合成をする．すなわち，サービスフローは実

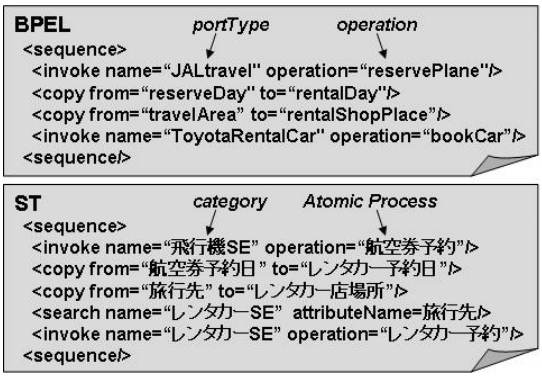

図 7 BPEL と ST 記述例（ただし，比較部分以外は省略）
Fig. 7 Example of description of BPEL and ST.

行時まで不在で，実行時に一から合成する形である．

本章では，BPEL および Task で ST と同じことをやる場合の記述性比較を行う．

BPEL で本研究のように，状況変化に応じて，適切な WS を連携させる場合には 3 つの案がある．

- 同等な WS のインタフェースをすべて統一して交換可能なようにする．
- すべての状況ごとに BPEL ファイルを準備しておき，当該状況になったら当該 BPEL ファイルを選択，実行する．
- BPEL ファイルは 1 つで，様々な状況パターンごとに，switch 文で起動する WS を記述しておく．

ここで，一番最初のインタフェースをすべて統一する方式は非常に困難で現実的でないため除外することとし，ST，BPEL，BPEL（switch 文），Task の 4 つの方式をサービスフロー記述性に関する以下の評価観点で比較検討する．評価観点は，記述文量，対応サービス数，サービス合成時のユーザ負荷，記述できるアプリケーションの幅，メタデータ管理負荷である．

記述文量に関して検討する．ST と BPEL は図 7 のように，記述文量自体はほとんど変わらず，既知の WSDL を基に記述するか，メタデータを基に記述するかの違いである．BPEL（switch 文）の場合は，状況パターンごとに switch-case で起動する WS のポートタイプとオペレーションを指定するとともに，condition 文でどの条件のときにどの case を選択するかを BPEL ファイルに記述する必要があるため，記述量が多い（ST の case 数倍以上）．Task は，サービスフローは実行前には存在しないため，記述文量はない．

対応サービス数に関して検討する．ST は，たとえば記述されたカテゴリ数が 5 で，それぞれのカテゴリで指定された機能と同等機能を提供できる SE が各 a，b，c，d，e 個ある場合は，最大で abcde パターンのサービスを合成できる．BPEL は 1 パターンのサービスにしか対応できない．BPEL（switch 文）は，BPEL ファイル内に記述した case 数分だけ対応できるが，すべて記述するのは困難である．また，BPEL や BPEL（switch 文）は仕様上，BPEL 作成時に DB に登録してあった WS しか利用できないが，ST は実行直前に新たに登録された SE も，OWL リンクをたどって同等であることを解決できれば利用できる．Task は毎回合成するため対応サービス数は 1 である．

サービス合成時のユーザ負荷について検討する．ST の場合は，ユーザは SE 自動選択に使うユーザポリシを指定しておく必要があり（安さ，近さなどの評価基準の重ね合わせで点数付け），また手動選択の場合は点数が高い SE から選択する必要がある．ただし，SE 選択のためのユーザポリシは 1 度合成エンジンに設定しておけば変える必要もないので，それほどの負荷にはならない．BPEL の場合は，状況に応じて利用する BPEL ファイルを選択する必要があり，適切な BPEL ファイル選択する稼動が必要である．BPEL（switch 文）は，condition 文に条件が書いてあるので，自動で起動できるが，あらかじめ様々なパターンの条件を書いておくのは困難である．Task は毎回，利用できる WS の組合せでできる 2 つの WS の組を指定して合成するため，毎回手動で 1 つ 1 つの WS を指定する必要がありユーザ負荷は高い．

記述できるアプリケーション幅について検討する．ST と BPEL は制御タグはほぼ同じものを使っているので，記述可能アプリケーション幅は変わらない．記述ではなく実装の問題だが，BPEL を解釈する BPEL エンジンはあらかじめデプロイした WS を起動する形であり，サービス実行中に状況変化があり新たな WS に切り換えるということはできないが，ST を解釈する合成エンジンは，ユーザの状況変化に応じてより適切な SE が出てきた際に，サービス実行中に SE を切り換えることができる．Task は，可能な制御はメソッド呼び出しとパラメータコピーだけであり，BPEL や ST でできる，ループや分岐などの複雑な処理はできない．

メタデータ管理負荷について検討する．ST を用いる合成システムでは，あらかじめメタデータ付与とメタデータ関係記述が必要だが，別々の人が定義しても，ある程度はマッピング機能がメタデータ関係記述を抽出してくれるため，管理負荷はそれほど大きくない．BPEL の場合は，インタフェースを統一的に定義・決定する必要があるため，決定するまでの調整稼動は大きいが，決定後は新しいインタフェースを追加するまでは管理稼動は必要ない．Task は，WS に OWL-S でメタデータを付けているので，異なるドメインで定義されたメタデータをマッピングすることも原理的に可能だが，現在のスコープおよび実装はメタデータ体系が一意であることを前提としている．

以上の検討を整理すると，**図 8** のようになり，ST の効率性が分かる．特に，メタデータマッピングと ST の抽象性により，1ST でいくつものパターンのサービスに対応できるメリットは大きく，付録のアプリケーションなどの開発においても，BPEL に比べて開発効率は高い．

なお，ST によるサービス合成は，実行時まで SE が決まらないため以下の 3 つの不安要素がある．これ

	ST	BPEL	BPEL(switch)	Task
Amount of description	fair	fair	bad	good
Number of services which can be applied	good	bad	fair	bad
User effort of service composition	good	fair	good	fair
Type of applications which can be described	good	fair	fair	bad
Management cost of metadata	fair	good	good	fair
Total evaluation	good	fair	fair	bad

図 8 記述効率性の比較表
Fig. 8 Evaluation table of service flow description.

らについて補足説明する．

①選択した SE が故障などで利用できない不安．

②選択した SE が必要な機能を持っていない不安．

③選択した SE がユーザにとって適切でない不安．

まず，①に関しては，WS などの通常の分散システムでも同様の問題があり，故障の際は例外処理を行うが，基本的にはそのコンポーネントの利用は諦めるしかない．サービス合成も同様の考えで，エラーが返ってきたら，新しい SE を選択する形を想定している．

次に，②に関しては，ST に記述した Atomic Process と同等機能を実装している SE を，メタデータ管理 DB と SE-DB を用いて検索するため，OWL リンクおよび OWL-S 記述が正確であれば，この問題は防げる．問題は不整合がある場合だが，カテゴリどうしの OWL 記述の不整合がある際は，マッピング機能などを用いて定期的にアップデートすることで整合をとる．また，SE の OWL-S 記述が不正の場合は，SE-DB 管理者に不正登録として通知して削除してもらうか，あるいは，SE-DB に登録する際に SOAP レイヤでの到達性をツールで確認してから登録させる方法が考えられる．

最後に，③に関しては，ユーザの設定したポリシに従って，点数付けするため，ポリシが詳細に設定されていれば，適切なものを選択できるが，詳細なポリシ設定は大きな稼動が必要である．そのため，完全自動選択が不安なユーザには，ユーザポリシに基づいて点数付けした後，高得点の上位 SE 群を提示して手で選択してもらう手動選択モードの利用を想定している．

5. 合成システムの実装と性能評価

本章では，サービス合成実行時の性能について検証する．サービス合成技術は，ST に基づいて，欲する機能と同等の SE 群を発見し，コンテキストに合った SE を選択し，サービスを合成する．しかし，メタデータ解決や適切な SE の選択は計算量を必要とするため，通常の BPEL エンジンに比べて，実行性能に懸念がある．そこで，提案のサービス合成システム（合成エンジン，メタデータ管理 DB，SE 管理 DB，ST 管理 DB，SE）を実装し，性能測定を通して，サービス合成システムの性能に問題がないかの検証を行う．

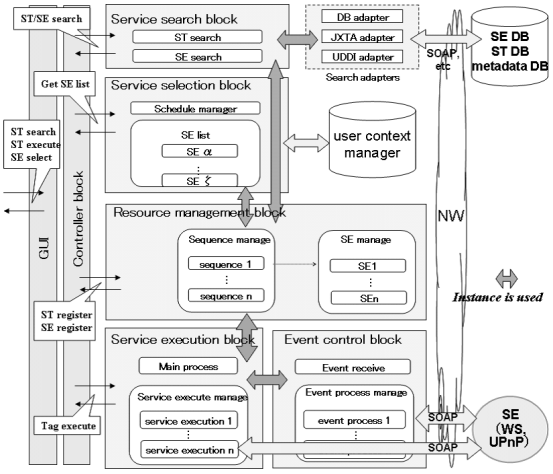

図 9 合成エンジンの機能ブロック図
Fig. 9 Service compositon engine function blocks.

5.1 合成システム実装

図 9 に，実装したサービス合成エンジンの機能ブロック図を示す．合成エンジン，各 DB，SE は OS が WindowsXP の PC に Java 言語（JDK1.4.2）で実装した．利用したミドルウェアは以下のとおりである．合成エンジン：Axis 1.1，Jena 2.1，Jetty 5.1.3．メタデータ管理 DB：EsTerra（XML DB）．SE，ST 管理 DB：PostgreSQL 8.0．SE：Tomcat 4.1.31，Axis 1.1．

以下，各機能部の概要を説明する．

サービス検索部は，ST，SE 管理 DB から，ST や SE の OWL-S 記述を取得する．ST を検索する際は，ST の名前や提供者名，キーワードなどをキーに検索を行う．同様に，SE を検索する際は，ST に記述されたカテゴリ名や Atomic Process などをキーに検索が行われ，SE の OWL-S 記述が発見される．ここで，メタデータ管理 DB を利用することで，語彙定義は異なるが同等機能の SE 記述も取得することが可能である．なお，検索アダプタを変更することで，検索手段を，UDDI や JXTA [20] などに変更することもできる．

リソース管理部は，実行中の ST のタグ情報や SE の OWL-S 情報，SE のパラメータ情報，状態などのリソースを管理する．

サービス実行制御部は，ST のタグ解釈実行，SOAP メッセージング，パラメータ型キャストを行う．ST タグ情報に基づいて SOAP メッセージを作成し，SE の

オペレーション呼び出しを行う際は，OWL-S Grounding ファイルを見て，SE の実際のインタフェースに変換して呼び出しを行う．タグ解釈実行について補足すると，4 章記述のとおり，ST と BPEL の制御タグの違いは，search タグだけであり，サービス実行制御部は，BPEL 制御タグと search タグの解釈実行機能を実装している（BPEL には switch や while といった分岐やループなどの制御タグがあるが，ST でも同様に利用可能である）．

イベント制御部は，イベントを上げる SE から同期または非同期のイベントメッセージを受ける．(例：交通事故発生イベント）

サービス選択部は，発見された候補 SE 群の Profile 記述とユーザコンテキスト情報，ユーザポリシを用いて，候補 SE 群の点数付けを行い，高い点の SE を自動または手動で選択を行う．手動の場合は，制御部を介してユーザに高い点の SE を提示し手動選択してもらう．また，合成サービス実行中も周期的に点数付けを行うことで，より状況に適した SE が出てきた場合は，SE の自動切換え，または手動切換えを提案する（切換え処理自体は，SOAP メッセージで制御するため実行制御部が行う）．

制御部は，サービス合成の包括的な制御を行う．具体的には，ST や SE の取得，選択された ST や SE の登録，ST タグ実行処理要求などを行う．そのほか，SE の手動選択モードの際のユーザへの SE 選択の促しや，手動切換えモード時に実行中の SE 切換え提案があった際に，ユーザへの SE 切換え確認も行う．また，SE を起動する際の必須入力パラメータに関しての，ユーザに問合せも行う（例：購入数など）．

GUI 部は，ユーザからの入力を受けたり，出力を表示したりするユーザインタフェースとなる．

5.2 サービス合成の性能測定

性能測定では，サービス合成の処理時間とメモリ使用量を 5 回測定した．グラフは 5 回の測定の平均値である．性能測定の環境は以下のとおりである．

IBM Netvista（CPU：Pentium4 2.53 GHz，RAM：512 MB，OS：Windows XP SP2）5 台を 100 BASE-TX のイーサネットケーブルで同一ハブに接続．

IBM1：合成エンジン（測定では，SE 選択を行う評価基準は，近さ，安さ，SE 評価値の 3 つとした．評価式は自由に定義できるが，たとえば，位置近さの場合，ユーザコンテキストに記述されたユーザ位置と，SE の Profile に記述された SE 位置から，exp(−(UserLatitude − SeLatitude)^2 − (UserLongitude − SeLongitude)^2) で点を付ける．安さは，値段が低い SE が高得点になる式を設定した．また，SE 評価値は，ユーザの利用履歴による SE の評価値であるが，それが高い SE が高得点になる式を設定した．位置や値段や SE 評価値は，Profile に記述されるが，これらは各 SE で測定前に静的に決めている．これら 3 つの評価基準を同じ重みで重ね合わせ最高点 SE を選択する）．

IBM2：ST，SE 管理 DB（測定では，500 個の ST，30000 個の SE の OWL-S，30000 個の WSDL 管理）．

IBM3：メタデータ管理 DB（測定では，300 個カテゴリを管理し，同値，継承関係をたどるステップ数は 2 とした．たとえば，図 3 では，ST に C 記述の場合，ステップ数が 0 の場合は C だけ，ステップ数が 1 の場合は A，C，E，F，K でマッチする SE が対象となる）．

IBM4，5：SE サーバ（測定では，(引数+L)×N÷M の余りを戻り値で返すメソッドの SE のみを各 15000 個ずつ準備した．L，M，N はランダムに決める自然数である．今回の評価は合成システムの評価であるため，部品である SE は処理時間がかからないシンプルなものとした）．

測定パターンと測定区間は以下のとおりである．

- 測定パターン
  1ST 内のカテゴリ数：5，10，20，40，70，100
  1 カテゴリ中の候補 SE 数：1，3，10，30，100，300

ただし，ST に記述されるタグは，invoke（メソッド呼び出し）と copy（パラメータ引継ぎ）だけとし，1 カテゴリに対して，invoke と copy は 1 度ずつ使うこととする．すなわち，ST 内のカテゴリ数が 5 の場合は，選択された 5 個の SE との SOAP 通信はそれぞれ 1 回ずつで，前の SE の戻り値を次の SE の引数に引き継ぐ形である．また，性能測定する ST はランダムにカテゴリを選択して作られる．

- 測定区間
  ST 検索，SE 検索，SE 選択，合成サービス実行．

図 10 は，1ST 内のカテゴリ数が 10 のときに，ST 検索，SE 検索，SE 選択，合成サービス実行の処理時間を縦軸に，1 カテゴリ中の候補 SE 数を横軸に示したグラフである．また，**図 11** は，1 カテゴリ中の候補 SE 数が 10 のときに，各処理時間を縦軸に，1ST 内のカテゴリ数を横軸に示したグラフである．

図 10，11 から，処理時間で支配的なのは，SE 検索時間であることが分かる．SE 検索時間は，カテゴリ中候補 SE 数，1ST 内カテゴリ数増加にともない増える．これは，DB 処理の遅延もあるが，合成エンジン

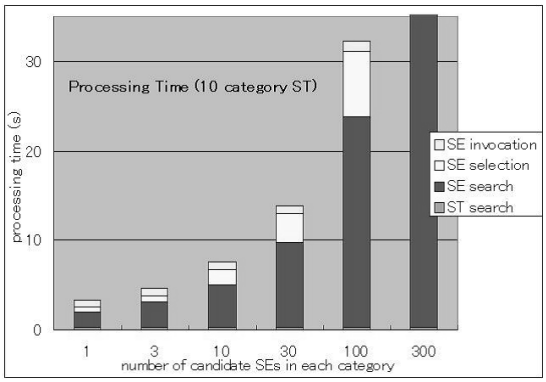

図 10 処理時間（1ST 内カテゴリ数を 10 に固定）
Fig. 10 Processing time (number of category in one ST is fixed to 10).

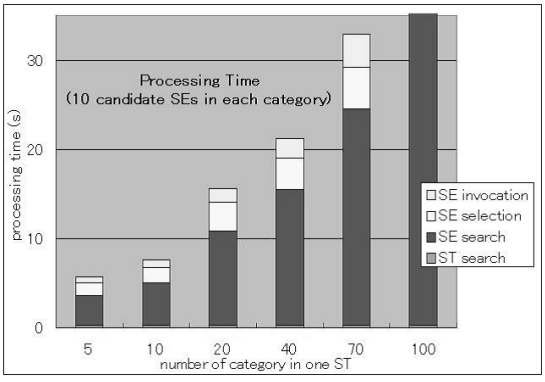

図 11 処理時間（1 カテゴリ中候補 SE 数を 10 に固定）
Fig. 11 Processing time (number of candidate SEs in one category is fixed to 10).

が妥当性チェックのため，検索された SE の Service, Profile, Process, Grounding, WSDL のパースをしているためパースにかかる時間が大きな原因である．

その次の支配項として，SE 選択時間がある．これも，SE 検索同様，カテゴリ中候補 SE 数，1ST 内カテゴリ数の増加にともない増える．SE 選択は各 SE の Profile 記述とコンテキスト情報から，評価式を用いて点数付けする形となっているため，複数の条件で選択しようとすると，評価式が多くなり計算量が多くなるため時間がかかる．今回の測定では，特に位置の近さ計算の計算量が多い．

合成サービス実行時間は，論理的にはカテゴリ中の SE 候補数によらず，1ST 内のカテゴリ数に比例するはずであり，測定結果もほぼそのとおりになっている．処理時間に関しては，ネットワーク遅延も含めて 1invoke あたり 60 ms，1 copy あたり 10 ms と問題ない．また，ST 検索時間は 350 ms でほぼ一定である．

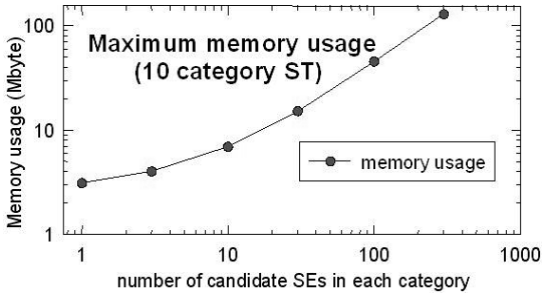

図 12 処理中の最大メモリ使用量
Fig. 12 Maximum memory usage (number of category in one ST is fixed to 10).

トータルの処理時間は，SE 検索時間が支配的であり，カテゴリ中候補 SE 数，1ST 内カテゴリ数の増加にともない増えている（実測の結果，ほぼ 0.4〜0.6 乗に比例している）．なお，サービス合成の処理時間の値は，同スペックのマシン，同等の条件で比較した市中の BPEL エンジンとほとんど遜色ない．

また，**図 12** は，1ST 内のカテゴリ数が 10 のときに，処理区間中最大メモリ使用量を縦軸に，1 カテゴリ中の候補 SE 数を横軸に示したグラフである．メモリ使用量は，ST 検索時に約 1 Mbyte 増加し，SE 検索終了時にほぼ図 12 の最大使用量に達し，その後はあまり変化しない．SE 選択後もメモリが変化しない理由を説明する．合成エンジンには，サービス実行中に状況変化にともないより適切な SE が出てきた際に，その SE に切り換えるサービス切換え機能も実装している．そのため，選択された SE 以外の代替候補 SE の OWL-S も保持するため，実行中もあまり変化しない．なお，メモリの大半は，OWL-S 情報であり，Service, Profile, Process, Grounding, WSDL を合わせると 1SE あたり 20 kbyte 程度で，SE 数に比例してメモリを使っており，SE 数が数万になると仮想メモリを利用するようになるため，性能は劣化する．

ST は一般ユーザが記述することを目指しており，それほど複雑なサービスでなく，1ST 内カテゴリ数は 10 程度，カテゴリ中候補 SE 数は 10 程度が，利用想定領域と考えている．その場合は，5 回測定した，検索，選択，実行の平均が 7.0 秒で処理でき，メモリ使用量は 7 Mbyte と，個人端末で十分実用可能なレベルといえる．一般に，Web システムの許容待時間は 8 秒と，Zona Research 社が 1999 年にまとめた調査報告 “The Need for Speed” [18] にあるが，それは満足している．

5.3 高速処理に向けた今後の課題

カテゴリ数や候補 SE 数が多くなった際でも，合成

エンジンをより高速でストレスなく利用できるようにするためには，さらなる実装工夫が必要であり，今後検討を進めていく．大きな改善可能点としては，以下の 3 つが考えられる．

- パースファイル数に比例して計算量が増えるため，実際の SE 呼び出しまで必要でない，Grounding と WSDL は SE 選択後に取得する．
- SE の検索時間を減らすため，高速に利用したいユーザにはオプション設定として，キャッシュを利用したり，DB から返す SE 発見数を絞ったりすることで検索時間を減らす（候補 SE 数と実行時間のトレードオフ）．
- メモリ使用量を削減するため，SE 選択後は，各カテゴリごとに代替候補 SE 情報は 10 個程度に絞り，他の OWL-S 情報は解放する．

6. 関 連 研 究

ネットワーク上のサービスコンポーネントを連携させ，サービスを提供するという従来手法としてはいろいろあるが，対象，想定が本研究と近いものとして，BPEL[9]，STONE[21]，Ninja[22] などがあげられる．BPEL，STONE，Ninja は，本研究の ST と同様に，それぞれ BPEL 文書，サービスグラフ，Path という，サービスコンポーネントの連携手順記述に従ってサービスを合成する．BPEL は，WSDL に強依存のため，他の WS の代替が困難であることを 2.1 節で述べた．BPEL を相補する仕様として，WS-CDL[23]（Web Services Choreography Description Language）があるが，WS の動作内容や順番を規定する仕様であり，BPEL 同様ユーザに状況に合わせてサービス合成することはできない．STONE は，単機能コンポーネントを入出力インタフェースに独自の名前付けをし，名前一致のコンポーネントのみバインドすることで接続性を保証しているが，その独自性ゆえ拡張性にかける．Ninja も STONE 同様，独自記述の多さから普及が困難である．本研究では，SE に意味的メタデータを振り，検索時は OWL リンクなどを用いてメタデータで検索し，実行時に実際のインタフェースに変換して呼び出すことで，柔軟な合成ができることが優れている．なお，SE 記述には標準候補の OWL-S を，メタデータどうしの関係記述には OWL を，ST 記述は BPEL の制御タグに search タグを拡張した形であり，標準を積極的に利用している．

本研究と同様，Semantic Web 技術を用いてサービス合成する試みに，Task[19] や Ubiquitous Service Finder[24] がある．4 章で説明したように，Task では，デバイスやオブジェクトをすべてサービス化し，OWL-S で記述する．SE にあたるものをサービスと呼ぶが，ユーザがいるエリアで利用可能なサービス群の中で，あるサービスの出力が次のサービスの入力になる 2 つのサービスの組合せ候補が提示され，手動で組合せを選択し合成サービスを実行する．Ubiquitous Service Finder も同様に，入出力がマッチする WS を順次呼び出し数珠つなぎに合成を行う．Task と本研究の違いとしては，Task がエリア内のあるサービスに組み合わさる他のサービスを探す形で，個々のサービスに注目してボトムアップ的に合成するのに対し，本研究はまず欲する合成サービスを ST としてユーザが指定し，それに合う SE を発見し合成する形で，トップダウン的な合成を目指している．これらは相補的といえるが，Task ではエリア内のサービス数が増えた場合に組合せ候補が増えすぎユーザの手動選択が困難，1 つずつつなげていく形であるため分岐やループなどの複雑なサービスの実現が困難，という課題がある．本研究は，欲する機能を ST に記述したカテゴリに絞っているため，カテゴリごとに適切な SE を選択すればよく選択が容易である．また，5.1 節記述のとおり，ST で利用できる制御タグは，BPEL の制御タグと search タグであるため，分岐，ループなどの制御タグを用いることで複雑なサービスの実現も可能である．また，未実装であるが，SE 利用履歴から ST を生成するボトムアップ的仕組みの検討も行っている[5]．

7. ま と め

本稿では，著者らが研究開発を進める，ユビキタス環境においてユーザ主導でサービス合成するためのユビキタスサービス合成技術の提案と，その検証課題として，メタデータ管理容易化，ST 記述効率性，実行時性能の 3 つをあげた．これらの検証課題をそれぞれ，オントロジマッピング技術により少ない稼動でメタデータ管理ができること，従来記述との比較により 1 枚の ST で多くのサービスに対応可能で効率が良いこと，実装したサービス合成システム性能評価により十分な性能であること，を示した．

今後は，SE のメタデータ付与を半自動化するツールおよび ST を GUI エディタで容易に記述できるツールの開発を進め，多くのユーザに使ってもらうための検討を進める．また，2006 年 2 月に青森県のエルムの街ショッピングセンター[25] で行った，サービス合成システムを用いた買物支援サービスの実証実験結果をふまえ，実際のユーザの意見を反映した改良を行う．

また，本稿では，性能評価に関しては，PC に実装

した個人端末用合成エンジンについて述べたが，ネットワーク上サーバに合成エンジンを配置し，携帯電話ブラウザを介してサービス合成を行える，プロキシ型合成エンジン[6]，合成エンジン機能を限定してiアプリに実装した，携帯電話実装型合成エンジン[7]も実装している．

謝辞 本研究の一部は，平成17，18年度総務省「ユビキタスネットワーク認証・エージェント技術の研究開発」の研究助成によるものである．

参 考 文 献

1) Weiser, M.: Some computer science issues in ubiquitous computing, *Comm. ACM*, Vol.36, No.7, pp.75–84 (1993).
2) Schilit, B., Adams, N. and Want, R.: Context-Aware Computing Applications, *IEEE Workshop on Mobile Computing Systems and Applications*, pp.85–90 (Dec. 1994).
3) Yamato, Y., Tanaka, Y. and Sunaga, H.: Context-aware Ubiquitous Service Composition Technology, *The IFIP International Conference on Research and Practical Issues of Enterprise Information Systems* (*CONFENIS2006*), pp.51–61 (Apr. 2006).
4) 山登庸次，田中洋平：ユーザコンテキストに適したサービス要素提案機能を備えたサービス合成エンジンの開発，電子情報通信学会技術報告 NS2004-225 (Mar. 2005).
5) Yamato, Y. and Takemoto, M.: Method of Service Template Generation on a Service Coordination Framework, *2nd International Symposium on Ubiquitous Computing Systems* (*UCS2004*) (Nov. 2004).
6) Yokohata, Y., Yamato, Y., Takemoto, M. and Sunaga, H.: Service Composition Architecture for Programmability and Flexibility in Ubiquitous Communication Networks, *SAINT2006 Workshop*, pp.142–145 (Jan. 2006).
7) 山登庸次，田中洋平，徳元誠一，大石哲矢，武本充治：携帯電話実装型サービス合成エンジンの研究開発と評価，FIT2005 LL-001 (Sep. 2005).
8) UPnP web site. http://www.upnp.org/
9) BPEL web site. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
10) Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N. and Sycara, K.: Bringing Semantics to Web Services: The OWL-S Approach, *International Workshop on Semantic Web Services and Web Process Composition* (*SWSWPC2004*), pp.5–21 (July 2004).
11) Paolucci, M. and Sycara, K.: Autonomous Semantic Web Services, *IEEE Internet Computing October*, pp.34–41 (Oct. 2003).
12) OWL-S web site. http://www.daml.org/services
13) Semantic web web site. http://www.w3.org/2001/sw/
14) travelXML web site. http://www.xmlconsortium.org/wg/TravelXML/TravelXMLindex.html
15) Nakatsuji, M., Miyoshi, Y. and Kimura, T.: Proposal and Verification of Flexible Interface Mapping Technique for Automatic System Cooperation based on Semantics, *Web Intelligence 2005*, pp.812–813 (Sep. 2005).
16) 中辻 真，三好 優，木村辰幸：柔軟なシステム連携のための意味情報に基づくメッセージマッピング手法の提案と評価，日本DB学会 Letters, Vol.4, No.1, pp.37–40 (2005).
17) Noy, N.F. and Musen, M.A.: Anchor-PROMPT: Using Non-Local Context for Semantic Matching, *International Joint Conference on Artificial Intelligence* (*IJCAI*) *2001*, pp.63–70 (Aug. 2001).
18) Zona Research, Inc.: The Need for Speed, report (1999). (有償) http://www.aspstreet.com/resources/publications/d.taf/cid,510
19) Masuoka, R., Parsia, B., Labrou, Y. and Sirin, E.: Ontology-Enabled Pervasive Computing Applications, *IEEE Inteligent Systems*, Vol.18, No.5, pp.68–72 (2003).
20) JXTA web site. http://www.jxta.org/
21) 南 正輝，森川博之，青山友紀：動的でアドホックなネットワークサービスフレームワークの検討，マルチメディア，分散，協調とモバイルシンポジウム（DICOMO2000），pp.13–18 (June 2000).
22) Gribble, S., Welsh, M., Behren, R., Brewer, E., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R., Mao, Z., Ross, S. and Zhao, B.: The Ninja Architecture for Robust Internet-Scale Systems and Services, *Special Issue of Computer Networks on Pervasive Computing*, pp.473–497 (Mar. 2001).
23) WS-CDL web site. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/
24) Kawamura, T., Ueno, K., Nagano, S., Hasegawa, T. and Ohsuga, A.: Ubiquitous Service Finder - Discovery of Services semantically derived from metadata in Ubiquitous Computing, *4th International Semantic Web Conference* (*ISWC 2005*), pp.902–915 (Nov. 2005).
25) エルムの街 web site. http://www.elm-nomachi.jp/

26) Interop web site. http://www.interop.jp/call.html

付　　録

A.1 アプリケーション例

サービス合成アプリケーション例として「出張サポータ」（**図 13**）を Interop2005 [26] に展示した．

出張サポータは，ユーザの出張時のルーチンワークを低減するサービスであり，ユーザがどこにいても，出張先に出発する時刻になったら，近くのアラームで出発を促し，近くのプリンタやモニタに地図や電車時刻などの情報を提供するサービスである．特徴はコンテキストアウェア性とカスタマイズ性の 2 つである．1 つ目は，SE 選択機能とメタデータ管理 DB を用いることで，デバイスの違いなどを意識することなく，ユーザの位置に応じて適切なデバイスが実行できる．このデモでは，音を出すという機能に関して，アラームと AIBO の音を出す機能が同等機能とメタデータ管理 DB に OWL で登録されているため，会社ではアラームが起動するが，家ではアラームがないため代替の音を出す機能として最寄りの AIBO を選択して音を出す．ユーザは ST を書きなおさなくても，状況に応じて適切なデバイスを発見し利用することができる．2 つ目は，ST は個々の SE の詳細を意識せずに記述可能であるため，ユーザは GUI エディタで ST を容易に変更可能である（例：地図を天気に変更など）．

移動先において，適切な機器が選択されるのは，作り込みのロケーションサービスと類似に見えるが，作り込み型では，家ならばこの機器，会社ならばこの機器と，あらかじめ switch 文などを記述しているのが前提であり，未知の機器利用はできない．本研究は，その場で SE を発見・点数付けし，適切なものを自動選択して実行することができるため，汎用性，拡張性で優れている．

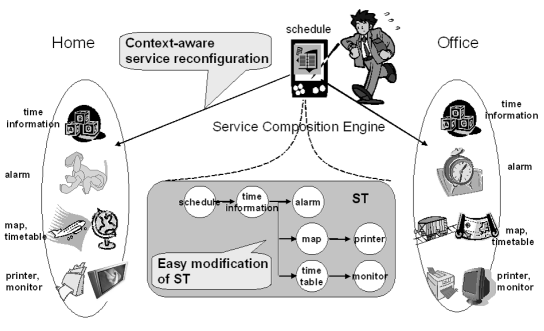

図 13 出張サポータイメージ図
Fig. 13 An image of Business Trip Supporter.

A.2 OWL-S 記述のサンプル

出張サポータの会社モニタ SE の，OWL-S の Service，Profile，Process Model，Grounding，WSDL を**図 14**，**図 15**，**図 16**，**図 17**，**図 18** に示す．テキスト量自体は多いが，OWL-S 作成ツールも実装して

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:service="http://www.daml.org/services/owl-s/1.0/Service.owl#">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>1.0</owl:versionInfo>
    <owl:imports
rdf:resource="file:C:/SWEngine_Oct/data/profile/Profile_Monitor.owl"/>
    <owl:imports
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl"/
>
    <owl:imports
rdf:resource="file:C:/SWEngine_Oct/data/ground/Grounding_Monitor.owl"/>
  </owl:Ontology>

  <service:Service rdf:ID="Service_Monitor">
    <service:presents
rdf:resource="file:C:/SWEngine_Oct/data/profile/Profile_Monitor.owl#Profile_M
onitor"/>
    <service:describedBy
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl#
monitor_PM"/>
    <service:supports
rdf:resource="file:C:/SWEngine_Oct/data/ground/Grounding_Monitor.owl#Grou
nding_Monitor"/>
  </service:Service>
</rdf:RDF>
```

図 14 OWL-S Service 記述サンプル
Fig. 14 Sample description of OWL-S Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:profile="http://www.daml.org/services/owl-s/1.0/Profile.owl#">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>1.0</owl:versionInfo>
    <owl:imports
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl"/
>
  </owl:Ontology>

  <profile:Profile rdf:ID="Profile_Monitor">

    <profile:has_process
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl#
monitor_PM"/>
    <profile:textDescription>Auto generated from
http://192.168.1.34:8080/axis/services/MonitorService?wsdl</profile:textDescri
ption>
    <profile:serviceName>会社モニタ</profile:serviceName>

    <profile:serviceParameter>
      <profile:ServiceParameter>
        <profile:serviceParameterName>エリア</profile:serviceParameterName>
        <profile:sParameter>武蔵野センタ12階東</profile:sParameter>
      </profile:ServiceParameter>
    </profile:serviceParameter>

    <profile:serviceCategory>
      <profile:ServiceCategory>
        <profile:categoryName>モニタ</profile:categoryName>
        <profile:taxonomy></profile:taxonomy>
        <profile:value></profile:value>
        <profile:code></profile:code>
      </profile:ServiceCategory>
    </profile:serviceCategory>

  </profile:Profile>
</rdf:RDF>
```

図 15 OWL-S Profile 記述サンプル（ここではエリアしか書いていないが，提供者や値段などの様々なメタデータも serviceParameter タグを用いて追加できる）
Fig. 15 Sample description of OWL-S Profile.

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:process="http://www.daml.org/services/owl-s/1.0/Process.owl#">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>1.0</owl:versionInfo>
    <rdfs:comment></rdfs:comment>
    <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns"/>
    <owl:imports rdf:resource="http://www.w3.org/1000/01/rdf-schema"/>
  </owl:Ontology>

  <process:ProcessModel rdf:ID="monitor_PM">
    <process:hasProcess rdf:resource="#コンテンツ表示"/>
    <process:hasProcess rdf:resource="#デフォルト表示"/>
  </process:ProcessModel>

  <process:Input rdf:ID="void_in">
    <process:parameterType
rdf:resource="http://www.w3.org/2001/XMLSchema-instance#nil"/>
  </process:Input>

  <process:Output rdf:ID="void_out">
    <process:parameterType
rdf:resource="http://www.w3.org/2001/XMLSchema-instance#nil"/>
  </process:Output>

  <process:Input rdf:ID="コンテンツURL">
    <process:parameterType
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </process:Input>

  <process:AtomicProcess rdf:ID="コンテンツ表示">
    <process:hasInput rdf:resource="#コンテンツURL"/>
  </process:AtomicProcess>

  <process:AtomicProcess rdf:ID="デフォルト表示">
    <process:hasInput rdf:resource="#void_in"/>
  </process:AtomicProcess>

</rdf:RDF>
```

図 16 OWL-S Process Model 記述サンプル（ここでは Atomic Process が 2 つのみで，入出力もシンプルなものであるが，SE の動作に応じて，事前条件などの記述も可能である）
Fig. 16 Sample description of OWL-S Process Model.

いるため作成自体は数分で可能である．合成エンジンの動作としては，SE-DB から Service ファイルを取得し，そこに記述された URI から Profile ファイルを取得し，それに基づいて SE を選択する．実行時には，Process Model と WSDL のマッピングが記述された Grounding ファイルを参照して，実際の WSDL に基づいた SOAP メッセージを作成して実行する．

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:grounding="http://www.daml.org/services/owl-s/1.0/Grounding.owl#">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>1.0</owl:versionInfo>
    <owl:imports
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl"/
>
  </owl:Ontology>

  <grounding:WsdlGrounding rdf:ID="Grounding_Monitor">
    <grounding:hasAtomicProcessGrounding
rdf:resource="#monitor_PM_DISPLAY_CONTENT"/>
    <grounding:hasAtomicProcessGrounding
rdf:resource="#monitor_PM_DISPLAY_DEFAULT_CONTENT"/>
  </grounding:WsdlGrounding>

  <grounding:WsdlAtomicProcessGrounding
rdf:ID="monitor_PM_DISPLAY_CONTENT">
    <grounding:owlsProcess
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl#
コンテンツ表示"/>

<grounding:wsdlDocument>http://192.168.1.34:8080/axis/services/MonitorServi
ce?wsdl</grounding:wsdlDocument>

    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>

<grounding:operation>http://192.168.1.34:8080/axis/services/MonitorService?w
sdl#displayContent</grounding:operation>

<grounding:portType>http://192.168.1.34:8080/axis/services/MonitorService?w
sdl#MonitorSE</grounding:portType>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>


<grounding:wsdlInputMessage>http://192.168.1.34:8080/axis/services/MonitorS
ervice?wsdl#displayContentRequest</grounding:wsdlInputMessage>

    <grounding:wsdlInputMessageParts rdf:parseType="Collection">
      <grounding:wsdlMessageMap>

<grounding:wsdlMessagePart>http://192.168.1.34:8080/axis/services/MonitorSe
rvice?wsdl#url</grounding:wsdlMessagePart>
        <grounding:owlsParameter
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl#
コンテンツURL"/>
      </grounding:wsdlMessageMap>
    </grounding:wsdlInputMessageParts>

  </grounding:WsdlAtomicProcessGrounding>

  <grounding:WsdlAtomicProcessGrounding
rdf:ID="monitor_PM_DISPLAY_DEFAULT_CONTENT">

    <grounding:owlsProcess
rdf:resource="file:C:/SWEngine_Oct/data/process/ProcessModel_Monitor.owl#
デフォルト表示"/>

<grounding:wsdlDocument>http://192.168.1.34:8080/axis/services/MonitorServi
ce?wsdl</grounding:wsdlDocument>

    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>

<grounding:operation>http://192.168.1.34:8080/axis/services/MonitorService?w
sdl#displayDefaultContent</grounding:operation>

<grounding:portType>http://192.168.1.34:8080/axis/services/MonitorService?w
sdl#MonitorSE</grounding:portType>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>


<grounding:wsdlInputMessage>http://192.168.1.34:8080/axis/services/MonitorS
ervice?wsdl#displayDefaultContentRequest</grounding:wsdlInputMessage>

  </grounding:WsdlAtomicProcessGrounding>

</rdf:RDF>
```

図 17 OWL-S Grounding 記述サンプル
Fig. 17 Sample description of OWL-S Grounding.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://192.168.1.34:8080/axis/services/MonitorService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://192.168.1.34:8080/axis/services/MonitorService"
xmlns:intf="http://192.168.1.34:8080/axis/services/MonitorService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://monitorse.smix"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:types><schema
targetNamespace="http://monitorse.smix"
xmlns="http://www.w3.org/2001/XMLSchema"><import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/><complexType
name="PrinterException"><sequence/></complexType></schema></wsdl:typ
es>
  <wsdl:message name="displayContentRequest">
   <wsdl:part name="url" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="displayDefaultContentRequest">
  </wsdl:message>
  <wsdl:message name="outputURLRequest">
   <wsdl:part name="url" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="outputURLResponse">
  </wsdl:message>
  <wsdl:message name="PrinterException">
   <wsdl:part name="fault" type="tns1:PrinterException"/>
  </wsdl:message>
  <wsdl:portType name="MonitorSE">
   <wsdl:operation name="displayContent" parameterOrder="url">
    <wsdl:input message="impl:displayContentRequest"
name="displayContentRequest"/>
   </wsdl:operation>
   <wsdl:operation name="displayDefaultContent">
    <wsdl:input message="impl:displayDefaultContentRequest"
name="displayDefaultContentRequest"/>
   </wsdl:operation>
   <wsdl:operation name="outputURL" parameterOrder="url">
    <wsdl:input message="impl:outputURLRequest"
name="outputURLRequest"/>
    <wsdl:output message="impl:outputURLResponse"
name="outputURLResponse"/>
    <wsdl:fault message="impl:PrinterException" name="PrinterException"/>
   </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="MonitorServiceSoapBinding" type="impl:MonitorSE">
   <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="displayContent">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="displayContentRequest">
     <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://monitorse.smix" use="encoded"/>
    </wsdl:input>
   </wsdl:operation>
   <wsdl:operation name="displayDefaultContent">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="displayDefaultContentRequest">
     <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://monitorse.smix" use="encoded"/>
    </wsdl:input>
   </wsdl:operation>
   <wsdl:operation name="outputURL">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="outputURLRequest">
     <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://monitorse.smix" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="outputURLResponse">
     <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://192.168.1.34:8080/axis/services/MonitorService"
use="encoded"/>
    </wsdl:output>
    <wsdl:fault name="PrinterException">
     <wsdlsoap:fault
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://192.168.1.34:8080/axis/services/MonitorService"
use="encoded"/>
    </wsdl:fault>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="MonitorSEService">
   <wsdl:port binding="impl:MonitorServiceSoapBinding"
name="MonitorService">
    <wsdlsoap:address
location="http://192.168.1.34:8080/axis/services/MonitorService"/>
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

図 18 WSDL 記述サンプル
Fig. 18 Sample description of WSDL.

(平成 18 年 4 月 22 日受付)
(平成 18 年 11 月 2 日採録)

山登 庸次（正会員）

2000 年東京大学理学部卒業．2002 年同大学大学院理学系研究科修了．2002 年日本電信電話（株）入社．P2P，ユビキタスコンピューティング研究に従事．電子情報通信学会会員．

中辻 真（正会員）

2001 年京都大学工学部数理工学科卒業．2003 年同大学大学院情報学研究科システム科学専攻修士課程修了．現在，日本電信電話株式会社ネットワークサービスシステム研究所勤務．セマンティック WEB 技術，データマイニング，情報流通システムの研究に興味を持つ．電子情報通信学会，日本データベース学会各会員．

須永 宏（正会員）

1981 年東京工業大学工学部制御工学科卒業，1983 年同大学大学院制御工学専攻修了，2002 年東北大学博士（情報科学）．NTT 研究所にて，交換システム，CTRON，P2P，Grid，ユビキタス等通信ソフトウェアの研究開発に従事．ITU-T にてラポータ（ネットワーク管理），TTC 専門委員会（IP 電話）委員長等も務め，1999 年に ITU 協会賞受賞．電子情報通信学会，IEEE 各会員．