修士論文 2009 年度 (平成 21 年度)

uMediator: 異種サービス間での ハンドオフ支援機構

慶應義塾大学大学院 政策・メディア研究科

中川 直樹

修士論文 2009 年度 (平成 21 年度)

uMediator: 異種サービス間でのハンドオフ支援機構

論文要旨

近年，サービスの構築支援技術の研究，開発が盛んに行われている．例えば，Google Maps API[12] や Amazon Web API[2] に代表される Web API，また UPnP[29] や Bonjour[22], Jini[17] に代表されるサービス発見プロトコルである．Web API は Web サイトなどで高機能なコンテンツをより短期間・低コストで開発できるよう支援する．サービス発見プロトコルはネットワークに新たに接続した機器に対して，既に接続されている機器やサービスの発見を可能にすることで，環境固有のサービスの開発を支援する．

各技術の発展に伴い，Web サービスや環境固有のサービスの構築が盛んに行われるようになった．特に環境固有のサービスは近年の計算機やセンサの小型化，多様化に伴い，センサや位置情報を利用したより環境に特化したサービスの構築が行われるようになった．また計算機やセンサの低価格化も同時に進んだことで環境固有のサービスは至る所で展開されるようになった．

環境固有のサービスが増加した情報環境では，異なるサービスでも似た内容のサービスが利用可能である場合が多数存在する．そのため，サービスの一貫性を確保しながらもその環境ごとで最もリッチなユーザエクスペリエンスの提供が可能なサービスに切替え，移動しながらサービスを利用したいという要求が考えられる．しかし，現在の技術では異なる環境内に存在するサービス間でのサービス切替えは不可能である．それは環境が異なると管理主体やサービス開発者が異なるため，共通の切替え用インタフェースを備えていないからである．また，各環境内に複数存在するサービスの中から，切替え先となり得る似た内容のサービスの判定もできないからである．

本論文では，これらの問題を解決するサービスハンドオフという新たな研究領域を定義し，サービスハンドオフを実現する環境としてサービスハンドオフフレームワークを提案した．その中でも主となるサービスハンドオフ支援機能とサービス情報管理機能をそれぞれ uMediator（ミドルウェア），Usdl Control Server（サーバ）として実装するとともに，サービス情報記述言語である USDL の拡張を設計することで問題を解決した．また，異なる環境に存在し共通のインタフェースを備えない動画再生サービス間でサービスハンドオフの実証実験を行い uMediator の有効性を実証した．さらに uMediator の異種サービス発見機能とサービスハンドオフデータ変換機能の性能評価を行った．

キーワード：

1 サービスハンドオフ　2 ユビキタスコンピューティング　3 異種サービス　4 データ変換　5 サービスローミング

慶應義塾大学大学院　政策・メディア研究科

中川 直樹

Abstract of Master's Thesis Academic Year 2009

uMediator: A System for Handoff between Heterogeneous Services

Summary

Technology to construct services has being researched and developed actively in recent years. Some of the examples of this technology are service discovery protocols, such as UPnP, Bonjour and Jini, and Web APIs, such as Google Maps API and Amazon Web API. Service discovery protocol allows a device newly connected to a network to discover devices and services that already exist in the network. Web APIs enable users to develop high quality contents with small amount of time and effort. As a result of advancement in these technologies, web services and environment specific services have been constructed briskly. Environmental specific services have been built using the location and environmental information acquired by sensor nodes, which have been improved in performance and size. Moreover, price reduction of computers and sensor nodes lead to spread of environment specific services. There are many kinds of similar services, therefore, user demands to use richest user experience service while moving around and switching between services that have consistency. However, switching between services that exist in different environment is impossible when using present technology. This is because, service manager and developer differs for each environment, and there is no interface provided for switching between services. In addition, there is no way of finding out similar service from multiple services that exist in the environment.

In this thesis, we define a new research area called service handoff, and propose a framework as an environment that can achieve service handoff. We implement service handoff and information management feature as uMediator and Usdl Control Server respectively, by extending USDL, a service information description language. We evaluate uMediator's validity by using it to handing off video streaming service that exists in different environment, which has no common interface. In addition, uMediator's performance is evaluated in terms of service discovery and service handoff data conversion function.

Keyword:

1 Service Handoff 2 Ubiquitous Computing 3 Heterogeneous Servicesf 4 Data Conversion 5 Service Roaming

Keio University Graduate School of Media and Governance

Naoki Nakagawa

目 次

図 目 次

表 目 次

第1章

序論

本章では，本研究の背景である環境依存サービスの増加について述べ，環境依存サービスが増加した情報環境で発生すると考えられる新たな要求と，要求実現への問題点について言及する．その後，本研究の目的を述べ，本論文の構成について解説する．

1.1 研究背景

近年，サービスの構築支援技術の研究，開発が盛んに行われている．例えば，Google Maps API[12] や Amazon Web API[2] に代表される Web API，また，UPnP[29] や Bonjour[22]，Jini[17] に代表されるサービス発見プロトコルである．Web API は，Web サイトなどの開発のためにインターネット経由で利用できる API であり，高機能なコンテンツをより短期間・低コストで開発できるよう支援する．サービス発見プロトコルは，ネットワークに新たに接続した機器に対して，既に接続されている機器やサービスの発見を可能にすることで，環境固有のサービスの開発を支援する．

本研究における “環境” とは，人のまわりを取り巻く周囲の状態や世界のことを指し，“情報環境” とは，人が生活する上で日々必要な情報を入手，蓄積，編集，利用が可能である環境を指す．また，“サービス” とは，環境に存在するシステム，情報機器上や Web 上に存在するアプリケーション，専用情報機器の場合は情報機器自身を指す．部屋の自動消灯を行うシステムであれば自動消灯システム自体を指し，Skype[25] や Google Maps[11] であればアプリケーション自体を指し，ミュージックプレーヤーであればミュージックプレーヤー自体を指す．また環境内でのみ利用可能でありその環境に特化した環境固有のサービスのことを，本研究では “環境依存型サービス” と呼ぶ．特徴としては汎用的なサービス構築が要求されない分，環境固有の情報機器からアクチュエートするよう作り込みが可能であり，Web サービスなどよりもリッチなユーザエクスペリエンスの提供が可能であることが挙げられる．

図 1.1: **現在の情報環境**

特に環境依存型サービスは，近年の計算機やセンサの小型化や多様化に伴い，センサや位置情報を利用した，より環境に特化したサービスの構築が行われるようになってきた．そして計算機やセンサの低価格化も同時に進んだことにより，環境依存型サービスは至る所で展開されるようになってきた [10]．環境依存型サービスが増加した情報環境では，異なるサービスでも似た内容のサービスが利用可能である場合が多数存在する．図 1.1 にこのような現在の情報環境の概観を示す．

その一例として，ナビゲーションサービスがある．環境依存型のナビゲーションサー

ビスに関しては，あしナビ [33] や上野まちナビ実験 [34]，ロボットや RFID[7] タグを利用したナビゲーションシステム [30] など多くの研究，開発が行われている．環境依存型のナビゲーションサービスが存在しない環境においては，ユーザ端末内のアプリケーションや Web 上のナビゲーションサービス [18] の利用が可能である．

これらのサービスのように，異なるサービスでも同系統のサービスのことを，本研究では異種サービスと呼ぶ．異種サービスは Java[16] の継承でいう，同一クラスの継承をしたサブクラス同士の関係であるサービスのことを指す．図 1.2 に異種サービス間の関係について示す．

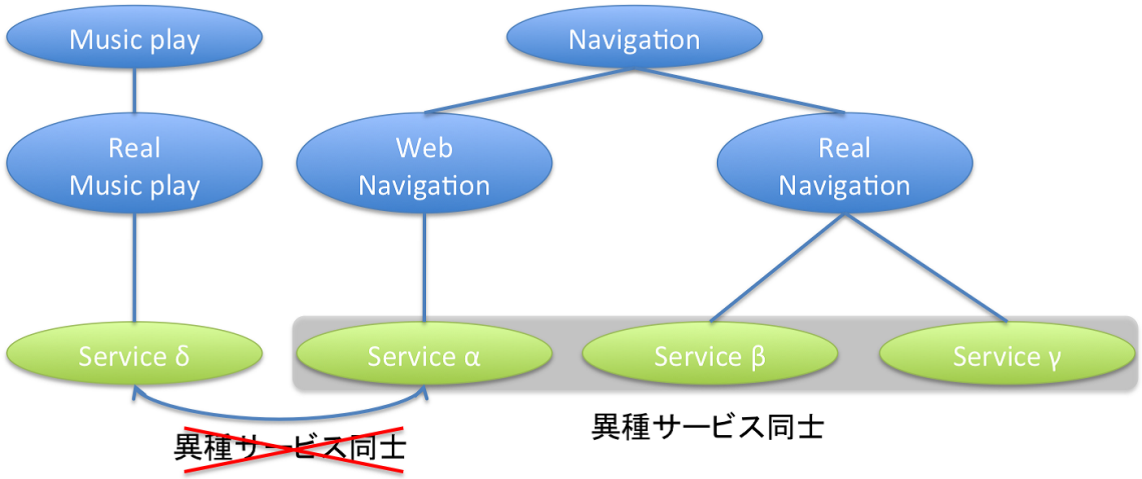

図 1.2: **異種サービス間の関係**

これらの状況を考慮すると，ユーザはサービスの一貫性を保ちつつ，その環境ごとで最もリッチなユーザエクスペリエンスの提供が可能なサービスに切替え，移動しながらサービスを利用したいという要求が考えられる．しかし，現在の技術では異なる環境内に存在するサービス間でのサービス切替えは不可能である．それは環境の管理主体やサービス開発者が異なるため，サービスが共通のインタフェースを備えていないからである．また，各環境内に複数存在するサービスの中から，切替え先となり得る異種サービスの発見ができないからである．

1.2 研究目的

本研究の目的は，異種サービス間での互換性を確保しサービスの切替えを支援する機構の構築である．本機構はサービス利用中ユーザの情報環境を跨ぐ移動の際，サービスの一貫性は保ちつつ各環境に存在する異種サービスへ切り替えられるようにする．また，切替え先の異種サービスの発見とサービス間の差異吸収は本機構が行うようにする．

1.3 本論文の構成

本論文は，全7章で構成される．次章では，本研究における想定環境とサービスローミングとの差異について論じ，問題を定義する．第3章では，本研究の研究領域であるサービスハンドオフについて定義し，機能要件とアプローチ方法について論考する．第4章では，サービスハンドオフフレームワークについて述べた後に，USDLA.4.5拡張記述とuMediator，Usdl Control Serverの設計について解説する．第5章では実装の詳細を述べ，第6章でシステムの基本性能評価の結果について述べる．第7章にて，本論文のまとめと今後の課題について言及する．

第2章

問題定義

本章では，まず本研究の想定環境とサービス切替えシナリオについて述べる．次に本研究と既存サービス移行技術であるサービスローミングとの差異について解説し，本研究が扱う領域を明確にする．最後に，異種サービス間での切替えの際に発生する問題点について論考し問題を定義する．

2.1 想定環境

本研究では，環境依存型サービスの利用が可能である情報環境と，非環境依存型サービスのみ利用が可能である情報環境が混在した環境を想定している．本節では，まず各情報環境の特徴について解説し，その後，本研究の想定環境についてまとめる．

環境依存型サービス利用可能環境

環境依存型サービス利用可能環境とは，環境依存型サービスの利用が可能である環境を指す．環境依存型サービスとは 1.1 節で述べたように環境固有のサービスのことを指す．特徴としては，汎用的なサービス構築が要求されない分，環境内の情報機器からのアクチュエートするよう作り込みが可能であり，他のサービス形態よりもリッチなユーザエクスペリエンスの提供が可能であることが挙げられる．つまり，環境依存型サービス利用可能環境は非環境依存型サービスのみ利用可能な環境より，リッチなサービスの利用が可能な環境であるといえる．

現状，このような環境は公衆無線 LAN 環境とユビキタスコンピューティング環境として存在している．

公衆無線 LAN 環境とは，一般的に無線 LAN を利用したインターネットへの接続サービスを利用出来る場所を指す．しかし，インターネットへの接続のみでなく環境内に存在するプリンタの利用や，環境内からのみアクセス可能なサービスなど，環境依存型サービスの提供を行う公衆無線 LAN 環境も多数存在する．

ユビキタスコンピューティング環境とは，Mark Weiser 氏 [31] が提唱した「環境中に多くのコンピュータを組み込むことで，いつでも，どこでも，だれでもが，意識しないで，状況に応じた最適な情報の利用ができる情報システム」であるユビキタスコンピューティングを実現した環境のことを指す．ユビキタスコンピューティング環境は多くの研究所や大学で研究・開発されており，マサチューセッツ工科大学の Oxygen[19]，カーネギーメロン大学の AURA[13]，ジョージア工科大学の AwareHome[6]，マイクロソフト社の Easy Living[4] などが挙げられる．これらの研究では，日常生活空間としてユビキタスコンピューティング環境を構築し，計算機やセンサを空間内に配置している．そして，空間内に設置されるセンサとして，温度や照度，湿度や人の位置情報を取得できるセンサが開発され，それらを利用した環境依存型サービスの研究や開発が行われている [15].

非環境依存型サービス利用可能環境

非環境依存型サービス利用可能環境とは，非環境依存型サービスのみ利用が可能である環境を指す．非環境依存型サービスとは，1.1 節で述べた Web サービスやユーザモバイル端末上に構築されたアプリケーションのことを指す．特徴としては，環境依存型サービスのように環境に特化したリッチなサービスの提供はできないが，環境に依存せずどこでも利用できることが挙げられる．ただし，NAVITIME[18] などの Web サービスのようにインターネットへの接続を要求するものも多数存在するため，本研究においてはインターネットへの接続が可能で

あり，環境依存型サービスの利用が出来ない環境を非環境依存型サービス利用可能環境と呼ぶ．

現状，このような環境は携帯電話通信方式として採用された第3世代移動通信システムにより提供され，広域で実現されている．

各情報環境共通の特徴としては，管理主体が異なり，サービスの開発者も異なることが挙げられる．そして，全てのサービスを共通に構築可能なガイドやフレームワークが存在しないため，各情報環境で提供されるサービスは互いに共通のプロトコルで通信可能なインタフェースを備えないことが挙げられる．また，1.1節で述べたように各情報環境では，異なるサービスでも似たような内容のサービス（異種サービス）の構築が行われている場合が多数存在することも挙げられる．

以上より，本研究では環境依存型サービス利用可能環境と非環境依存型サービス利用可能環境が混在し，各環境において互いに共通のプロトコルで通信可能なインタフェースを備えない異種サービスが構築されている環境を想定する．ゆえに互いに共通のプロトコルで通信可能なインタフェースを備えないサービス間でのサービス切替えを想定する．図2.1に本研究の想定環境の概観を示す．

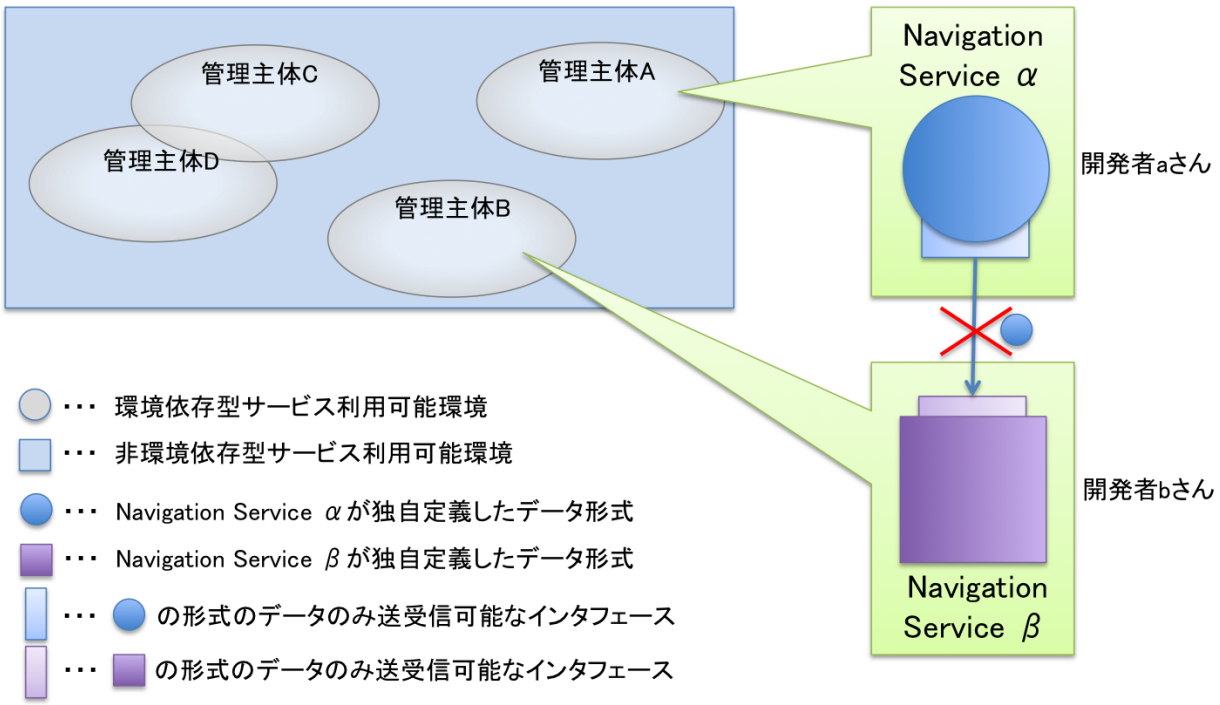

図 2.1: **想定環境の概観**

2.2 シナリオ

本節では，互いに共通のプロトコルで通信可能なインタフェースを備えないサービス間でサービス切替えが可能な場合のシナリオについて，二つ例を挙げ述べる．

ナビゲーションサービスの例

本シナリオは，A さんがショッピングモール X 内に店を構える店 Y に買い物に行くまでのシナリオである．

ある日，A さんはショッピングモール X 内に店を構える店 Y に買い物に行くことになった．しかし，A さんはショッピングモール X にも店 Y にも行ったことが無かったため，インターネットで店 Y の住所を調べ，持っていたモバイル端末から Web サービスであるナビゲーションサービス α を利用し店 Y へ向かうことにした．ショッピングモール X までは Web サービスを利用し無事にたどり着くことができた．しかし，非環境依存型サービスである Web サービスでは，ショッピングモール内のナビゲーションまでは行えない．すると A さんのモバイル端末上に実装された “異種サービス間でのサービス切替え支援機構” がショッピングモール内に構築された環境依存型のナビゲーションサービス β を発見し，目的地のお店を環境依存型サービスに通知することでサービスを切替えた．サービスを切替えたことで，A さんは店 Y までナビゲーションサービスを利用することができ，無事にたどり着くことができた．

動画再生サービスの例

本シナリオは，B さんが会社から帰宅した後にコンビニに出かける際のシナリオである．

B さんは会社からの帰宅時，持っていたモバイル端末でネット上にアップロードされたドラマをストリーミング再生し，鑑賞ながら帰宅した．B さんが自宅に到着しリビングに入ると，リビングのテレビに再生途中の動画が再生された．これは B さんのモバイル端末上に実装された “異種サービス間でのサービス切替え支援機構”が，自宅内に存在する環境依存型のサービスで切替えが可能な異種サービスを発見し，切替えたからである．その後，B さんはコンビニに買い物に行こうと家を出ようとすると，再生途中の動画がモバイル端末での再生に再度切り替わった．これは，B さんが環境依存型サービスの存在しない環境に移動したことをモバイル端末が検知し，非環境依存型サービスでサービスの切替えが可能な異種サービスを発見し，切替えたからである．このような異種サービス間でのサービス切替えが行われることで，B さんは移動しながらも，各環境においてサービスの一貫性を保ちながらサービスを切替え利用することができた．

2.3　サービスローミングとの比較

本節では，本研究の関連研究であるサービスローミングについて解説し，本研究との差異について述べる．

2.3.1　サービスローミング概要

サービスローミングとは，複数の計算機を協調させることによって，ユーザが利用しているサービスを別の計算機へ移送することである．ここでのサービスとは，本研究におけるサービスとは異なり，情報機器に存在するアプリケーションからユーザへ提供される機能を指す．例えば，メールソフトであればメール作成機能やメール送受信機能を指し，電話であれば通話機能を指す．

サービスローミングの実現手法には，アプリケーションが動作しているリモート計算機のデスクトップ画面をローカル計算機へ移送する方法（視覚移送）と，アプリケーションそのものを移送する方法（全移送），アプリケーションの状態のみ移送する方法（状態移送）がある．視覚移送は，リモートのデスクトップ画面をキャプチャして移送するため，既存アプリケーションをそのまま利用でき，VNC[28] や X Window System[32] で用いられている手法である．全移動は，アプリケーション全てを移送するため，アプリケーションを移送元の計算機と同じ状態で移送先の計算機に移送可能である．この手法は主にプロセスマイグレーション [8][27][14] として実現される．状態移送は，移送元の計算機で動作しているアプリケーションの状態を移送し，移送先の計算機で同じ状態を持つアプリケーションを生成する．例として，モバイルオブジェクト [1] が挙げられる．この手法では，送られてきた状態に応じてアプリケーションを再生成するため，移送先の計算機に同じアプリケーションの実行コードが設置されている必要がある．

上記の全ての移送手法において移送先アプリケーションは，移送元アプリケーションの移送手法に対応するよう開発されている．つまり，移送元と移送先のアプリケーションが共通のフレームワークまたは共通の API で開発されている．または専用のソフトウェアがインストールされているのである．これは，移送元と移送先のアプリケーションの管理主体が同一であるか，開発者が同一である場合に可能である．または，標準となるような移送手法が確立されている場合である．しかし現状そのような手法は存在しない．

2.3.2　サービスローミングとの差異

本研究が想定するような情報環境を跨ぐ移動の場合，移送元と移送先のアプリケーションの管理主体や開発者が同一であるような状況は考えにくい．ゆえに，これらの移送手法を適用しサービスローミングを行うことは不可能である．また，ユーザの移動に応じサービスを切替える点は同じであるが，同系統の異なるサービスへの切替え

を目的とする本研究と，同一のサービスを移送するサービスローミングとでは目的が若干異なる．図 2.2 に本研究とサービスローミングの対象領域の差異を示す．

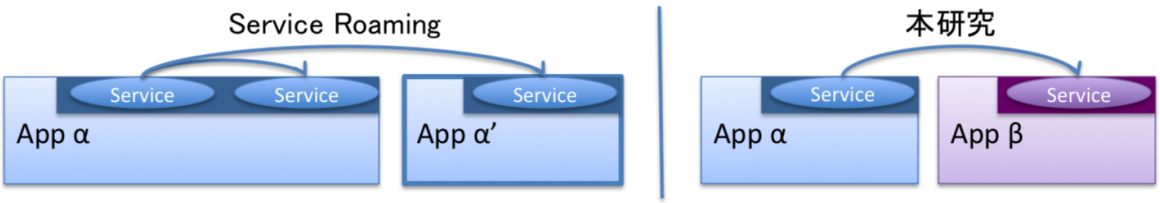

図 2.2: **本研究とサービスローミングの対象領域の差異**

2.4 ハンドオフの定義

本節では，本研究におけるハンドオフを定義する．

> 本研究における**ハンドオフ**とは，アプリケーションレイヤでサービスを切替えることである．

一般的に “ハンドオフ” とは，移動端末が接続するモバイル・アクセス・ゲートウェイを切替えて移動することを指すが，本研究ではアプリケーションレイヤでサービスを切替えることを指す．

アプリケーションレイヤ以外でのハンドオフとしては，ネットワークレイヤでのソフトハンドオフやハードウェアレイヤでのハードハンドオフ等がある [3][9]．ソフトハンドオフとは，現在通信中の基地局（ハンドオフ元）と新しく通信したい基地局（ハンドオフ先）を一時的に同時通信状態にした後に切替えることであり，理論上ハンドオフによる瞬断は無い．ゆえに，電波状態の良い基地局に瞬断無く切替える事が可能である．ハードハンドオフとは周波集切替えることであり，ソフトハンドオフと異なりハンドオフによる瞬断が生じる．

ハンドオフは上記のレイヤ以外のレイヤでも多くの研究 [26][24] が行われており，どのレイヤの研究においてもよりより良い状態を保つことを目的に切替えを行っている．本研究におけるハンドオフも，切替え対象やレイヤは異なるもののユーザにとってより良い状態を保つため，他のモノへ切替えるという目的は同じである．

2.5 異種サービス間でのハンドオフの問題点

本節では，想定環境である各情報環境の混在環境において，共通のインタフェースを備えない異種サービス間でのハンドオフを行う際に発生する問題点について論考する．

2.5.1 異種サービス発見問題

本研究では情報環境を跨ぐ移動をした際のサービスのハンドオフを想定しているため，まずハンドオフ先となるサービスを発見する必要がある．ハンドオフ先となるサービスは，ユーザが利用中のサービスからハンドオフしてもサービスの一貫性を保つことが可能なサービスである必要がある．しかし現在，UPnP[29] や Bonjour[22]，Jini[17] 等のサービス発見プロトコルは存在するものの，ハンドオフ先になりうる異種サービスを判定することはできない．

2.5.2 異種サービス間の互換性確保問題

サービス間でのハンドオフを行うには，ハンドオフ元サービスからハンドオフ先サービスへサービスの設定情報等を含んだ状態情報を渡し，ハンドオフ先サービスがそれを解釈し，サービスの一貫性が保てるように設定を行う必要がある．しかし本研究が想定する異種サービスは，共通のインタフェースを備えないため，独自定義したデータ形式でデータを作成し，独自定義したハンドオフプロトコルでデータを送受信する（図 2.3 参照）．

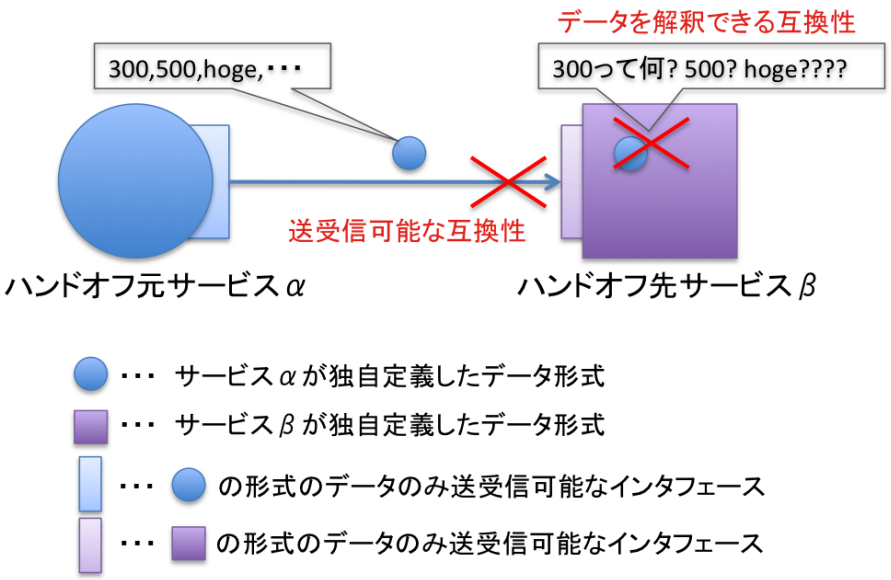

図 2.3: **異種サービス間の互換性問題**

ゆえに，異種サービス間でのハンドオフを実現するには，以下の二点の互換性を確保する必要がある．

- 異種サービス間でデータを送受信しあえる互換性
- 異種サービス間でデータを解釈できる互換性

現状では上記二点の互換性を確保する技術は存在しない．

2.6 本章のまとめ

本章では，まず本研究の想定環境とサービス切替えシナリオについて述べた．次に本研究と既存サービス移行技術であるサービスローミングとの差異について解説し，本研究が扱う領域を明確にした．最後に，異種サービス間での切替えの際に発生する問題点について論考し問題を定義した．

次章では，本章で明らかにした問題を解決する研究分野について定義し，対象サービスを明らかにする．その後，機能要件を明らかにし，アプローチ方法について論考する．

第3章

サービスハンドオフ

本章では，本論文の研究分野であるサービスハンドオフの定義をした後に，対象とする研究領域を明確にする．その後，機能要件について考察し，アプローチ方法について論考する．

3.1　サービスハンドオフの定義

本節では，本研究の研究領域をサービスハンドオフとして定義する．

> **サービスハンドオフ**とは，利用中サービスの一貫性を確保しながら“異種サービスへ”ハンドオフすることである．

この際，利用中サービスのことをサービスハンドオフ元サービス，切替え先サービスのことをサービスハンドオフ先サービスと呼ぶ．また，サービスハンドオフ元サービスとサービスハンドオフ先サービスは，共通のプラットフォームや API を使用せず構築されるものとする．つまり，共通のサービスハンドオフインタフェースを備えないサービス間で，サービスを切替えることがサービスハンドオフである．また，図 3.1 で示すように，2.4 節で述べたハンドオフとはサービスの一貫性を確保しながら異種サービスへ切替える点で異なる．

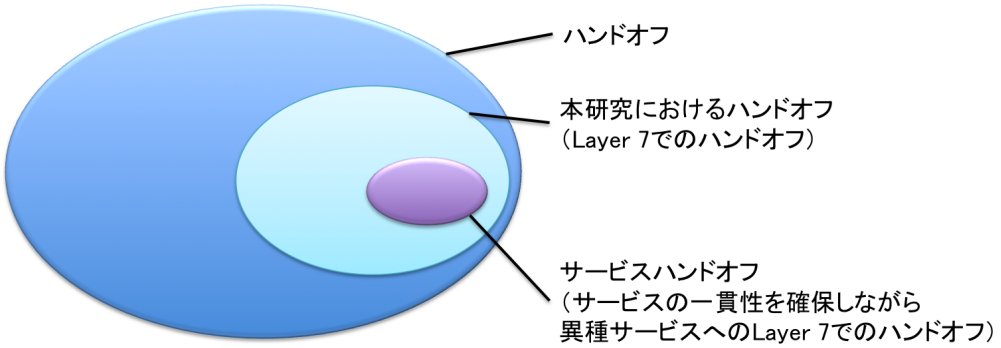

図 3.1: **サービスハンドオフの位置付け**

また，サービスハンドオフを可能にする環境を本論文ではをサービスハンドオフフレームワークと呼び，以降，サービスハンドオフを SH と略して記す．

3.2　本研究の対象とするサービス

本節では，サービスを分類し，本研究の対象とするサービス群を明確にする．まず，分類基準を定め，次にサービスの分類例を示す．その後，本研究の対象とするサービスについて述べる．

3.2.1 分類基準

サービスの分類基準として，そのサービスが自己完結型であるか，それとも連携型であるかという点に注目する．また連携型の場合は，連携先はサービス専用のものなのか，それとも汎用的に連携可能なものなのかという点に注目する．図 3.2 に分類基準の概観を示す．

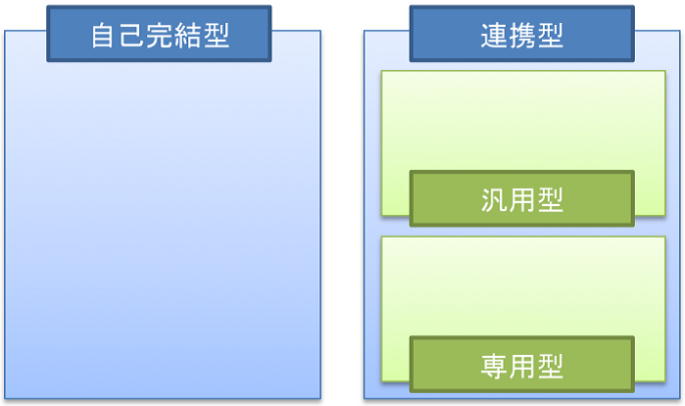

図 3.2: **サービス分類基準の概観**

自己完結型/連携型

サービスが他のサービスと連携せずに動作しているかどうかを示す．自己完結型サービスとは，他のサービスと連携せずに動作しているサービスを示す型である．自己完結型サービス例としては，エディタや電卓のような単独動作のサービスが挙げられる．

逆に，連携型サービスとは，他のサービスと連携しながら動作するサービスである．連携型サービス例としては，ストリーミング動画再生サービスやテレビ電話サービスが挙げられる．

自己完結型サービスは，サービスの状態情報や設定情報が移送されることで正常に SH の実行が可能である．しかし，連携型サービスは次で述べる汎用型サービスであるか連携型サービスであるかによって，SH の実現方法が異なる．

汎用型/専用型

連携型サービスの連携先が汎用的に連携可能かどうかを示す．汎用型サービスとは，連携先が汎用的に連携可能な場合のサービスを示す型である．サービス例としては，ストリーミング動画再生サービスが挙げられる．

逆に，専用型サービスとは，連携先がサービス固有のものであり汎用的に連携ができない場合のサービスを示す型である．サービス例としては，テレビ電話サービスが挙げられる．

汎用型サービスは，一度サービスの切替えが実現すれば，SH は成功する．しかし，連携型サービスはサービスを一度切替えても，その後の通信も変換し続けなくてはサービスの一貫性を確保しサービスを切替えられたことにはならず，SH は成功したことにならない．

3.2.2 サービスの分類

前項で示した分類基準を元に既存サービスの分類例を図 3.3 に示す．

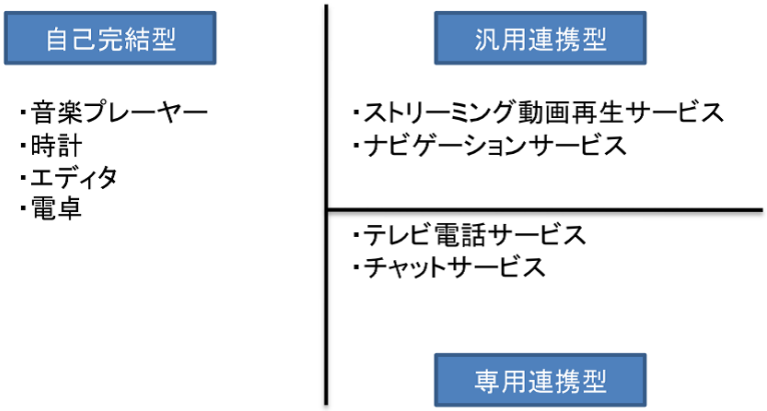

図 3.3: **サービスの分類例**

分類によって得られた 3 つのカテゴリをサービス群と呼ぶ．以下で，各サービス群の具体例について述べる．

自己完結型

他のサービスと連携せずに動作しているサービス群である．具体例としては，エディタや電卓のような単独動作のサービスが挙げられる．

汎用連携型

他のサービスと連携し，かつ連携先が汎用的に連携可能な場合のサービス群である．つまり，ユーザ利用中サービスを切替えれば，サービスの一貫性を保つことができるサービス群である．具体例としては，ストリーミング動画再生サービスのような連携動作のサービスが挙げられる．これは，リソース URI と再生経過時間を最初に取得すればサービスの一貫性を保ち切替えが可能である．

専用連携型

他のサービスと連携し，かつ連携先がサービス固有の場合のサービス群である．つまり，単純にユーザ利用中サービスを切替えても，サービス全体としての一貫性を保つことができないサービス群である．専用連携型サービス群の例として

は，テレビ電話サービスやチャットサービスのような連携動作のサービスが挙げられる．これらはユーザのクライアントサービスを切替えても通信相手のサービスは切り替らないため，継続的にデータを変換し続けなくてはサービスの一貫性を保つことはできない．

3.2.3 サービスハンドオフの対象とするサービス群

本研究では，3 種類のうち自己完結型と汎用連携型サービス群への SH 対応化を行う．専用連携型を除く理由は，専用連携型はサービス切替え後も通信データを継続的に変換し続ける必要があり，サービス固有の切替え機構についての研究になってしまうためである．つまり，特定のサービス間でのみ SH を実現する研究になってしまうためである．本研究ではまだ確立されていない SH の基礎技術を確立すべく，まずサービスを一度切替えれば，サービスの一貫性を保つことが可能であり，SH を実現可能な自己完結型と汎用連携型に対象を絞って研究を行う．汎用的により多くのサービス間での SH を実現できるようにすることを目指す．

3.3 機能要件

本節では，前述した本研究の対象領域を考慮し，SH 実現のための機能要件について述べる．

異種サービス発見機能

SH は，サービスを切替えてもサービスとしての一貫性は保たれる必要があり，同系統のサービス間でなくては実現不可能である．つまり全てのサービス間で実現可能なものではない．ゆえに，複数存在するサービスの中から SH 先サービスとなる異種サービスの発見を行う必要がある．これを実現するのが異種サービス発見機能である．本機能は SH を行う上で不可欠である．

SH データ変換機能

本研究が想定する各サービスは，共通の SH インタフェースや SH プロトコルを備えないため，独自定義した SH データを送受信し，お互いのデータを解釈し合うことはできない．ゆえに，各サービスが備える SH インタフェースや SH プロトコルを理解し，お互いにデータを解釈し合えるよう変換する必要がある．これを実現するのが SH データ変換機能である．本機能を実現することで，異種サービス間での互換性確保が可能となる．

3.4　アプローチ方法の比較検討

本節では，3.3 節で述べた機能要件を満たし，本研究の目的である異種サービス間でのサービス切替えを実現する方法を，サーバ側で行う場合とユーザ端末側で行う場合に分けて論考する．これは，サーバとユーザ端末の性質の違いを利用する事で異なるタイプの SH が可能なためである．以下で各アプローチ方法について解説した後に，各方法のメリット，デメッリトを論じ，本研究で適用するアプローチ方法について論考する．

3.4.1　サービス固有の変換サーバによる状態データの変換移送

ユーザモバイル端末上の SH 支援機構が環境内に存在するサービス情報を取得し，SH 元サービスの情報と比較検討し，SH 先サービスを決定する．そして，SH 元サービスへ SH 先サービスの情報を送信することで，SH 元サービスは，互いのサービスの SH プロトコルを解釈し変換できる変換サーバへ状態データ（SH データ）を送信する．変換サーバは，受信した状態データを SH 先が解釈可能な形に変換し，SH 先サービスへ状態データを送信する．この変換サーバは，同系統のサービスへの変換を一通り行えるように実装する．SH 元サービスと SH 先サービスの SH プロトコル情報を解釈し，変換するサーバの情報は，各サービスのサービス情報に自 SH プロトコルを解釈可能なサーバのリストとして記載しておくことで発見することができる．また，本サーバは切替え時の状態情報のみでなく，切替え後にも通信が必要であるサービスにも対応可能である．例えば，テレビ電話サービスである．テレビ電話サービスは継続的に通信が行われるため，データを継続的に変換し送信し続ける必要があるが，本アプローチ方法では SH 先サービスからのデータを変換サーバが継続的に変換を行い，通信相手に送信することで対応が可能である．図 3.4 にサービス固有の変換サーバによる変換移送のイメージを示す．

3.4.2　モバイル端末上の支援機構による状態データの変換移送

ユーザモバイル端末上の SH 支援機構が環境内に存在するサービス情報を取得し，SH 元サービスの情報と比較検討し，SH 先サービスを決定する．そして，SH 元サービスの状態データ（SH データ）を取得し，SH 先サービスが解釈可能な形に状態データを変換する．その後，変換した状態データを SH 先サービスへ送信する．SH 元と先のサービスが対応する SH プロトコル情報は，サービス情報として取得しておくため，データ変換前に各サービスが解釈可能なデータ形式等を知ることができる．図 3.5 に状態データ（SH データ）の変換移送のイメージを示す．

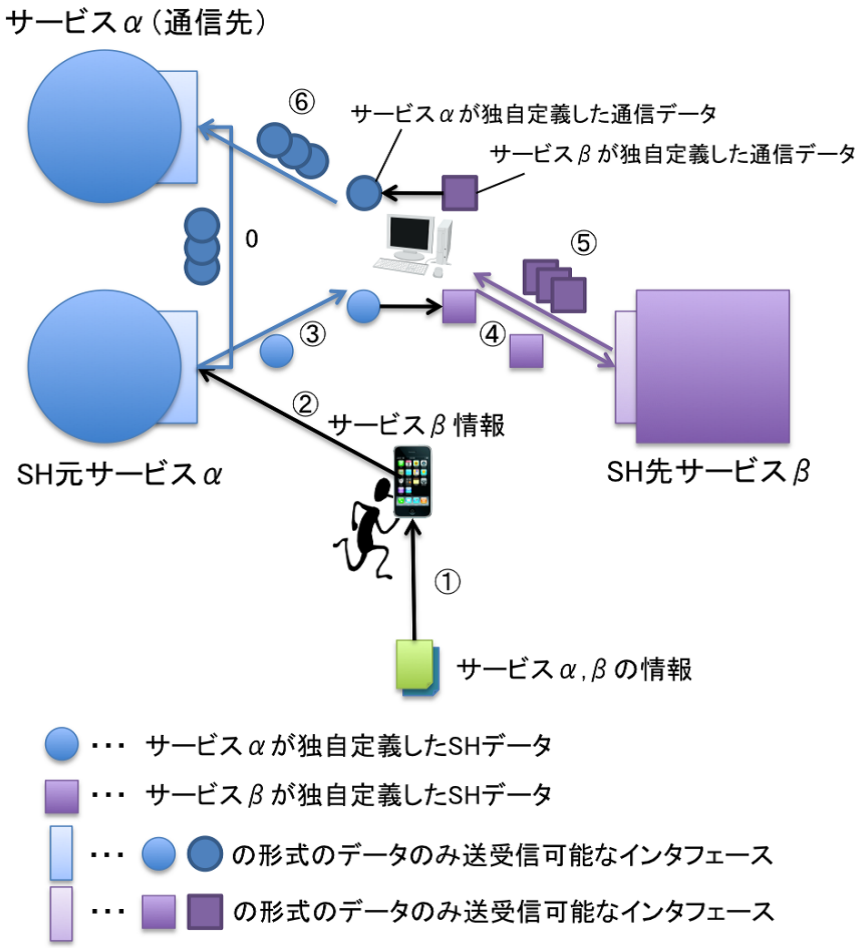

図 3.4: **サービス固有の変換サーバによる変換移送のイメージ**

3.4.3 本研究のアプローチ方法

3.3節で述べた機能要件を満たし，本研究の目的である異種サービス間でのサービス切替え（サービスハンドオフ）を実現する方法を二つ挙げた．

サービス固有の変換サーバによる状態データの変換移送については，サービス切替え後もデータ変換を行うことが可能であるため，モバイル端末上の支援機構による状態データの変換移送が対応できないサービスへの対応が可能である．しかし，同系統のサービスへの変換を一通り行えるように実装するには，大変な開発コストがかかる．また，どこかの環境において新たに同系統のサービスが開発されたら，それに対応するために追加実装する必要があり，非現実的な上に汎用的にサービスを切替えられる手法とはいえない．

モバイル端末上の支援機構による状態データの変換移送の場合は，サービス固有の変換サーバによる状態データの変換移送とは異なり，サービス切替え後のデータ変換を行うことは不可能であるが，汎用的にサービスを切替えられる手法である．また，

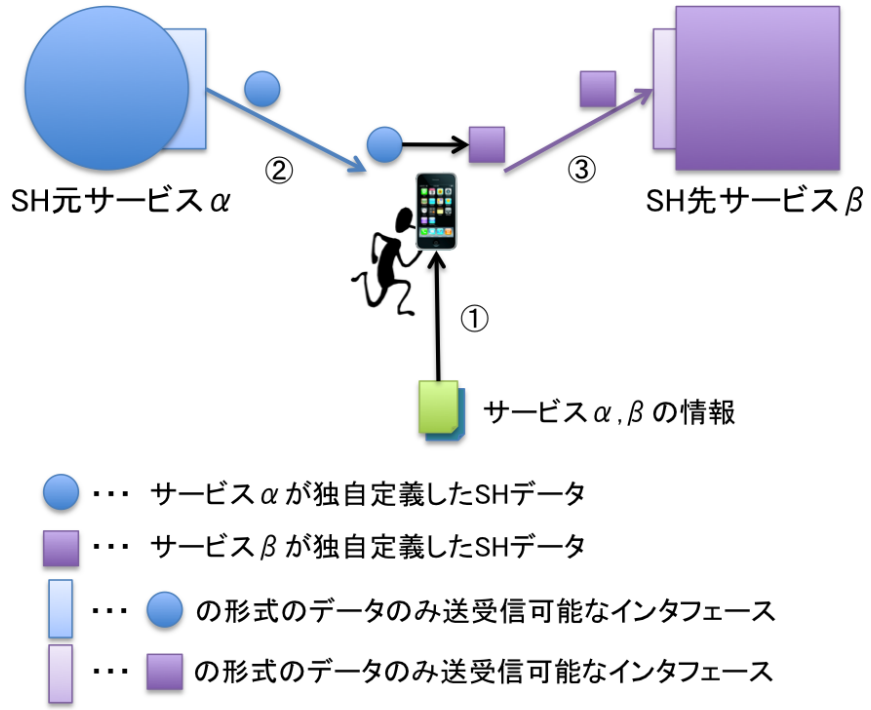

図 3.5: **状態データの変換移送のイメージ**

サービスの数だけ変換機構を実装する必要は無く，基本的にはモバイル端末上の SH 支援機構の構築を行うのみで SH 可能であり，現実的な手法であると言える．ゆえに，本研究ではモバイル端末上の支援機構による状態データ（SH データ）の変換移送を行うことで SH を実現する．

3.5 本章のまとめ

本章では，本論文の研究分野である SH を定義した後に，対象とする研究領域を明確にした．その後，機能要件について考察し，アプローチ方法について論考した．

次章では，アプローチ方法を元に SH フレームワークの構成について論考し，本研究の対象機能を明らかにする．その後，対象機能の設計について解説する．

第4章

設計

本章では，まずサービスハンドオフを可能にするサービスハンドオフフレームワークの構成について述べ，本研究の対象機能を明らかにする．その後，対象機能を実現するミドルウェアと，連携サーバ，サービス記述言語拡張の設計について述べる．

4.1　サービスハンドオフフレームワーク

本節では，3.4 節で述べたアプローチ方法に即し，SH を実現するサービスハンドオフフレームワーク（以降，SH フレームワーク）について述べる．異種サービス発見機能は SH 管理機能と SH 発見機能，SH 支援機能，サービス情報管理機能により実現される．SH データ変換機能は SH 支援機能とサービス情報管理機能により実現される．

本節では，まず，SH フレームワークの全体像について解説する．その後，各処理機能とサービス情報記述言語について述べる．

4.1.1　サービスハンドオフフレームワークの全体像

本フレームワークは，SH 管理機能と SH 支援機能，サービス情報配信機能，サービス情報管理機能で構成される．図 4.1 に本フレームワークの全体像を示す．

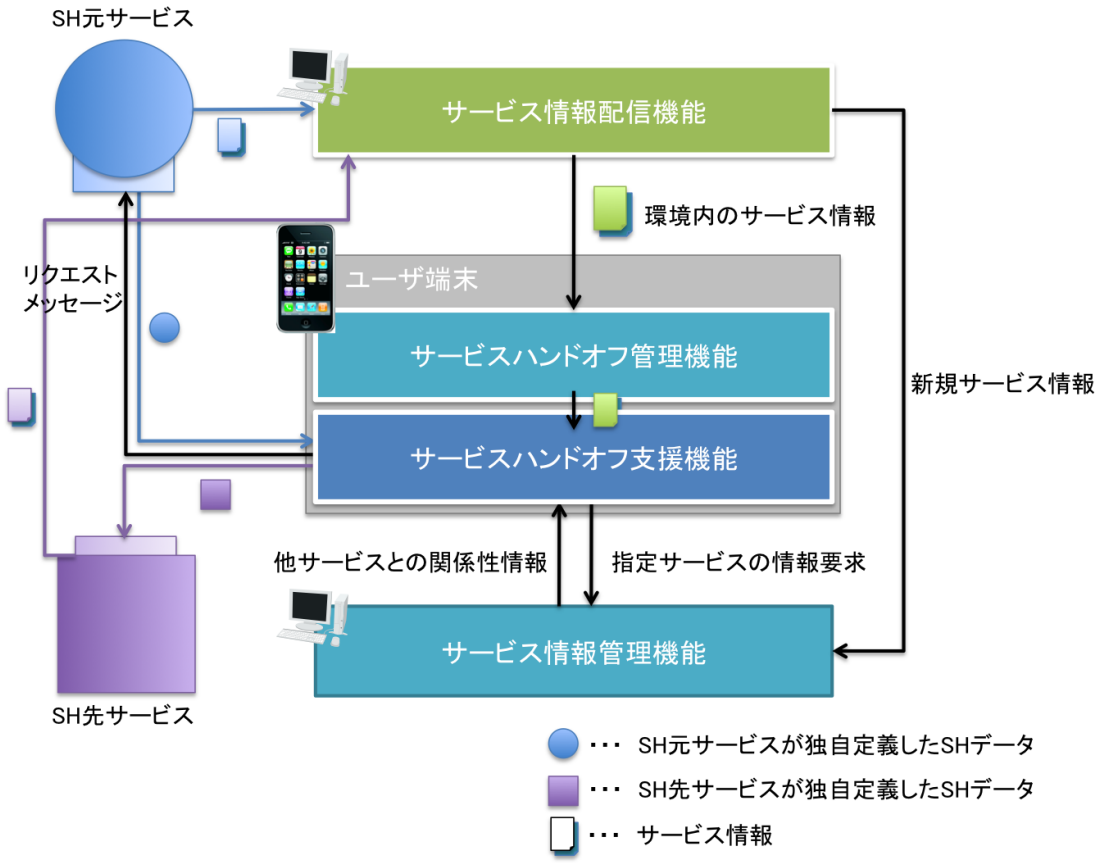

図 4.1: **サービスハンドオフフレームワーク**

本フレームワークは複数存在する SH 先候補サービスの中から，異種サービスを判定することで SH 先サービスを発見する．そして SH 元サービスからの SH データを SH 先

サービスが解釈できる形に変換し仲介することでSHを実現する．この際，SH先サービスを決定したり，SH先が解釈可能なデータ形式を本フレームワークが知るために，サービス情報記述言語が必要となる．次項からは各処理機能について解説する．

4.1.2 サービスハンドオフ管理機能

SH管理機能はユーザモバイル端末上に実装され，SH自体を管理する．具体的にはユーザへSHインタフェースを提供し，ユーザのSH設定を可能にする．ゆえに，ユーザにとってのリッチなサービス等も本機能が定義する．また，ユーザ利用中のサービスや情報環境の移動を監視し，移動を確認した際はSH先候補となるサービスの情報を収集する．そして，ユーザからの設定に従いSH先候補のサービス情報ファイルをフィルタリングしたうえで，SH支援機能に渡すことでSH自体を管理する．

4.1.3 サービスハンドオフ支援機能

SH支援機能はSH管理機能と同じくユーザモバイル端末上に実装され，SH管理機能からSH先候補となるサービス情報を受け取り，SH先を決定する．そして，SH元サービスからSHデータを取得し，SH先サービスが解釈できるよう変換しSHデータを仲介することでSHを支援する．図4.2にSH支援機能のイメージを示す．

4.1.4 サービス情報配信機能

サービス情報配信機能は情報環境ごとに必要とされ，環境内のサービス情報を保持する．そして環境内に新たに加わった端末に対してサービス情報を配信することで，SH支援機能の処理を支援する．また新規にサービスが環境内に構築され，サービス情報が追加された場合はサービス情報管理機能に新規サービス情報を通知する．

4.1.5 サービス情報管理機能

サービス情報管理機能は任意の範囲に一つ必要とされ，サービス情報配信機能に新たなサービス情報が追加されると，その情報を取得，解析し他のサービス間との関係性や継承を整理し，保存する．そして，SH支援機能からの情報要求に応え，処理を支援する．

4.1.6 サービス情報記述言語

本フレームワークでは，各サービスが自サービス情報をサービス情報配信機能へ登録する．サービス情報は，サービス提供サーバのIPアドレスやPort番号だけでなく，

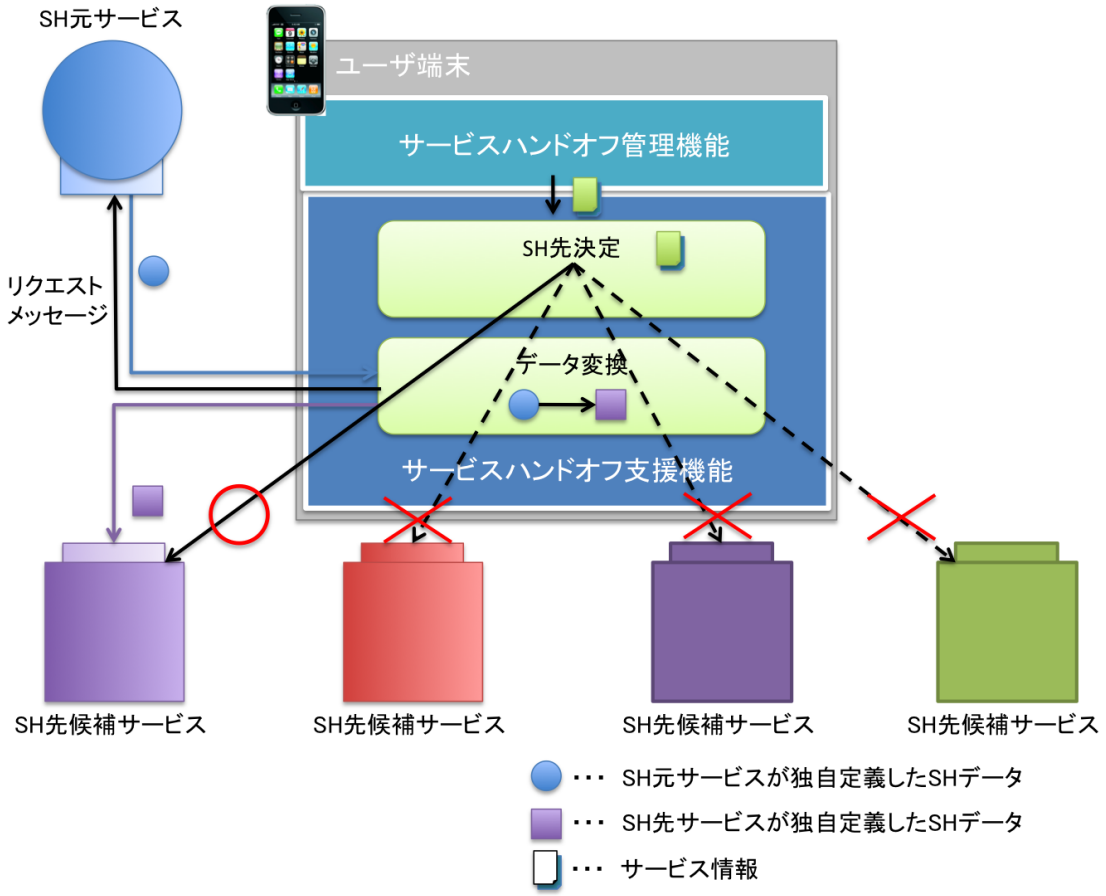

図 4.2: **SH 支援機能イメージ**

各サービスが独自定義した SH データや SH プロトコル情報についても記述する必要がある．SH データについてはサービス内で独自定義したものなので，他のサービスが解釈できるようにデータ内の情報の一つ一つの意味情報を記述する必要がある．また，後述するが異種サービスの判定をするためにサービスの継承についても記述する必要がある．

上記の記述要件を満たすことが可能なサービス情報記述言語は，現状存在しない．そこで本研究では，サービス提供サーバの IP アドレスや Port 番号，サービスの継承情報を標準で記述可能であり，サービス情報を型と実体に分けて記述し型を継承することで体系的に継承情報を整理可能な言語である USDL（Universal Service Description Language）（A.4.5 参照）を採用する．そして拡張記述法を新たに設計，定義することで記述要件を満たせるようにする．また，以降サービス情報は USDL によって usdl ファイルに記述されるものとする．

4.1.7 本論文の対象機能

本研究では，SH フレームワークのコアである，SH 支援機能に焦点を絞って研究を進めた．ゆえに，以降では SH 支援機能を中心に述べる．しかし，サービス情報管理機能は SH 支援機能と通信し処理に大きく関わるため，SH 支援機能への応答に必要な処理部を中心に解説する．他の処理機能に関しては SH 支援機能のテスト環境としての簡易実装のみを行ったので，本論文で詳細については述べない．

4.2 設計方針

本節では，サービス情報記述言語である USDL の拡張記述と SH 支援機能を実現するミドルウェア uMediator，サービス情報管理機能を実現するサーバ Usdl Control Server の設計方針について述べる．

4.2.1 USDL 拡張記述

USDL の拡張記述を設計・定義することで，サービス提供サーバの IP アドレスや Port 番号，サービスの継承情報だけでなく，各サービスの SH プロトコル情報をサービス情報の一部として USDL 記述可能にする．それにより異種サービスの判定等が可能となるだけでなく，各サービスから uMediator への SH プロトコル情報の通知が可能となり，uMediator では SH プロトコル情報を解釈することで SH データの変換が可能となる．

4.2.2 uMediator

ユーザ端末上で SH 先サービスの発見を行い，SH データを SH 先サービスが解釈可能な形に変換することで仲介を行うミドルウェアである．SH 先サービスの発見は，SH 管理機能が収集した usdl ファイルをもとに，SH 先候補サービス群の中からユーザ利用中サービスの異種サービスを判定することで，SH 先サービスを発見する．また，SH データの変換は SH 元サービスと SH 先サービスの SH プロトコル情報を usdl ファイルから抽出し，SH 元サービスからの SH データを解釈・分解し，SH 先サービスが解釈可能な形に再構築することで SH データの変換を行う．

4.2.3 Usdl Control Server

任意の範囲で一意に設置され，uMediator からのサービス情報要求に応答するサーバである．サービス開発者により作成された usdl ファイルはサービス情報配信サーバである Usdl Delivery Server にアップロードされ，Usdl Delivery Server が新規 usdl ファイルを Usdl Control Server に送信する．Usdl Control Server では，受信 usdl ファイル

から必要情報を抽出しデータベースとして保持することで uMediator からの要求に備える．

4.3 USDL 拡張記述の設計

本節では USDL の拡張記述について述べる．拡張箇所は SH プロトコル情報を記述する handoff 要素と SH プロトコル情報の記述で利用される変数定義，変数間の関係性を記述する relationships 要素である．変数については 4.3.5 項で詳細を述べる．handoff 要素は SH の受信側サービスの情報を記述する handoff_receive_protocol 要素と，SH データの送信側サービスの情報を記述する handoff_send_protocol 要素を子要素として持つ．

また本研究における SH プロトコル情報とは，USDL 記述における handoff 要素内で記す，各サービスが独自定義した SH データの構造情報，意味情報，データタイプ（xml や csv などのフェイル形式を指す），リクエストメッセージを指す．

4.3.1 handoff 要素

handoff 要素はサービスの実態情報を記述する entity 要素の子要素として，SH プロトコル情報を記述する．サービスの型情報を記述する type 要素の子要素記述は許可しない．理由は，本研究が定義する SH は各サービスごとに独自の SH プロトコル利用が可能であるため，type 要素への handoff 要素記述を許可する事で，本来の USDL の使用用途において継承の制限を引き起こす可能性があるためである．

また，サービスは SH 元サービスとなる場合と SH 先サービスとなる場合の二つの状況が考えられる．ゆえに本要素では，SH プロトコル情報をサービスが SH 元サービスである場合と SH 先サービスである場合に分け記述する．handoff 要素の記述構成の全体像を図 4.3 に示す．

SH 元サービスの SH プロトコル情報記述

SH 元サービスは図 4.4 で示すように，uMediator から SH データのリクエストメッセージを受信すると，独自の SH プロトコルに従い SH データを uMediator に送信する．

この際，正常に処理を完了するためには，サービスから uMediator へ事前に SH プロトコル情報を通知する必要がある．具体的には SH データの構造情報，意味情報，データタイプ，リクエストメッセージである．ゆえに，これらの SH プロトコル情報を記述する handoff_send_protocol 要素の設計を行う．handoff_send_protocol 要素は data_type 属性と requestMes 属性を保持する．data_type 属性では SH データのデータタイプを指定する．SH データのデータタイプについては 4.3.2 項で詳細を述べる．requestMes 属性にはサービスに SH データのリクエストを送信する際に必要となるメッセージ内容の指定を行う．

-要素名 handoff_send_protocol

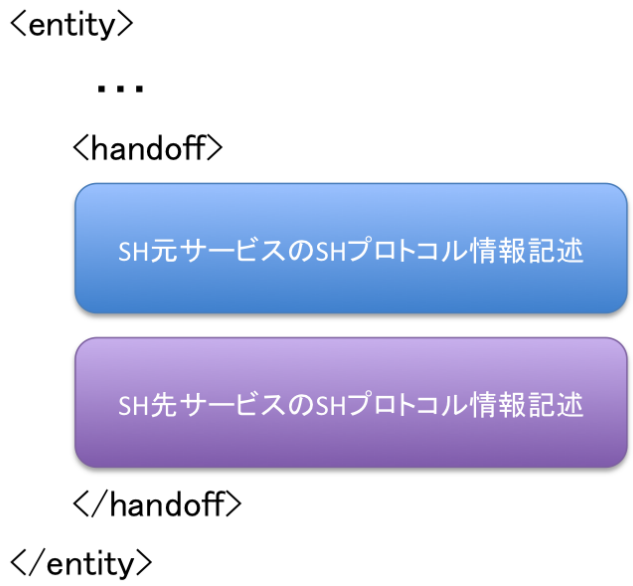

図 4.3: **handoff 要素の記述構成の全体像**

-属性 data_type, requestMes

-出現数 必ず 1 つ

例：

<handoff_send_protocol data_type="*xml*" requestMes="*Handoff_request_xml*">
データタイプ別 SH データの構造&意味情報記述
</handoff_send_protocol>

SH 先サービスの SH プロトコル情報記述

SH 先サービスは図 4.5 で示すように，uMediator から SH データを独自の SH プロトコルに従い受信する．

この際，正常に処理を完了するためには，サービスから uMediator へ事前に SH プロトコル情報を通知する必要がある．具体的には，SH データの構造情報，意味情報，データタイプである．ゆえに，これらの SH プロトコル情報を記述する handoff_receive_protocol 要素の設計を行う．handoff_receive_protocol 要素は data_type 属性を保持する．data_type 属性では SH データのデータタイプを指定する．SH データのデータタイプについては 4.3.2 項で詳細を述べる．

-要素名 handoff_receive_protocol

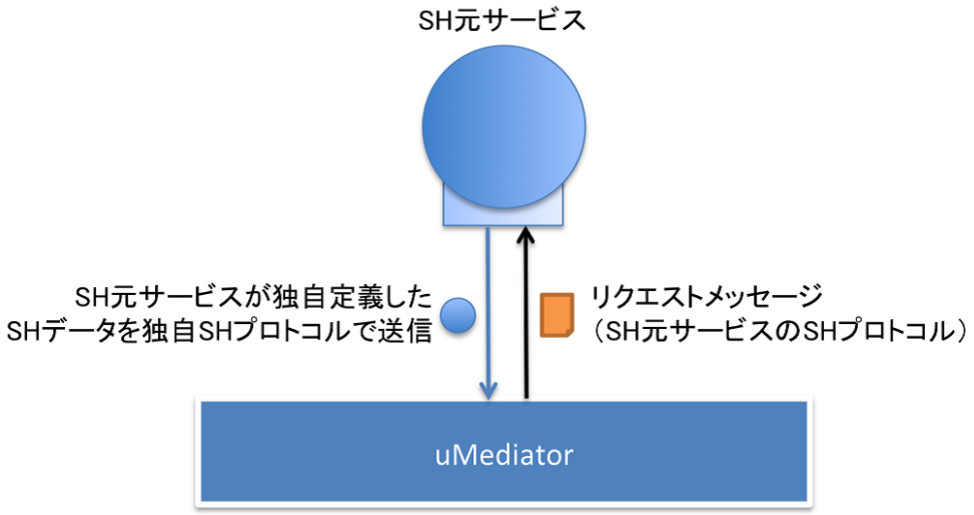

図 4.4: **SH 時の SH 元サービスと uMediator**

-属性 data_type

-出現数 必ず 1 つ

例：

<handoff_receive_protocol data_type="*xml*">
　　データタイプ別 SH データの構造&意味情報記述
</handoff_receive_protocol>

4.3.2　データタイプ別 SH データの構造&意味情報記述

本項では SH プロトコルとして利用可能なデータタイプ（xml や csv などのファイル形式を指す）を分類し，各データタイプごとの SH データの構造情報と意味情報の記述について解説する．

SH データは様々なプログラミング言語で構築された異種サービス間での状態情報交換に利用される．そのため，SH データは汎用的にメッセージ交換可能なデータタイプである必要がある．ゆえに本研究で想定する SH データは，プログラム間で汎用的にメッセージ交換可能な XML と CSV に限るものとし設計，定義する．以下ではデータタイプ別に SH データの構造情報と意味情報の記述について述べる．

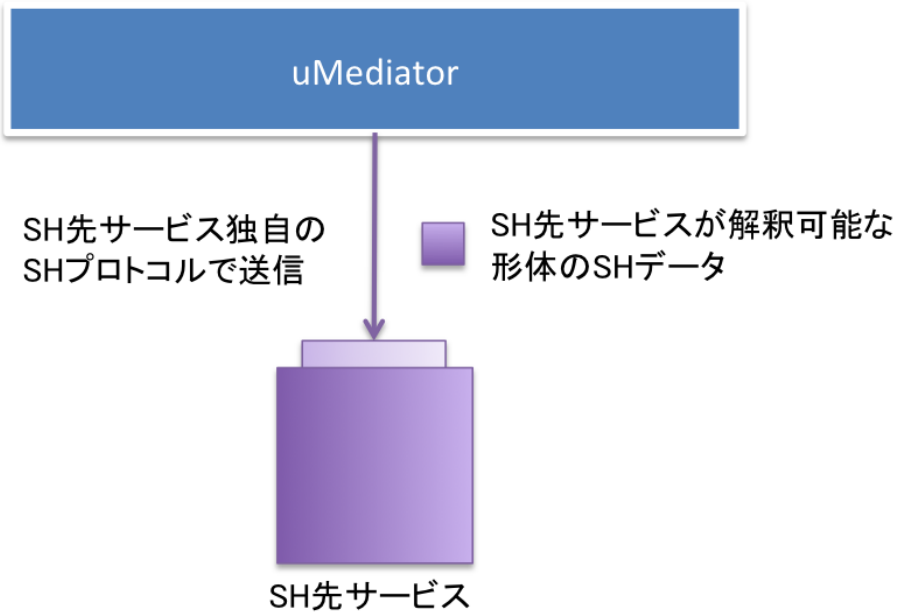

図 4.5: **SH 時の SH 先サービスと uMediator**

データタイプが XML の場合

SH データの送信側（handoff_send_protocol 要素），受信側（handoff_receive_protocol 要素）のどちらの場合においても，自サービスが対応する SH データのデータタイプが XML の場合，data_type 属性で “xml” を指定することで SH データのデータタイプが XML であることを示すことが可能である．また，SH データの送信側（handoff_send_protocol 要素），受信側（handoff_receive_protocol 要素）のどちらの場合においても，子要素に実際の XML データの実データ値部分を変数に置き換えた XML データを記述することで，SH データの構造情報と意味情報を記述することが可能である．変数については 4.3.5 項で詳細を述べる．図 4.6 に usdl 記述例（送信側）と送信 SH データ例，図 4.7 に usdl 記述例（受信側）と受信 SH データ例を示す．

図 4.6 と図 4.7 の usdl 記述例と実際に送信される SH データを比較して見ると分かるが，USDL の handoff_send_protocol 要素と handoff_receive_protocol 要素の子要素は，実際の XML データの実データ値部分が変数に置き換えた XML データがそのまま記述されている．これにより，サービスが送受信可能な SH データの構造情報と意味情報を共に uMediator に通知する事が可能となる．[···] で記述されている部分はデータの意味情報を記述した usdl ファイルの id を示しており，SH 時には [···] の部分に実データが代入されることから本研究においては [···] を変数と呼ぶ．

usdl 記述例（送信側）：

```
<handoff_send_protocol data_type="xml" requestMes="Handoff_request_xml">
  <serviceA>
    <length unit="msec">
      [jp.ac.keio.sfc.ht.audio.variable.time_msec]
    </length>
    <data>
      <play_time unit="msec">
        [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_msec]
      </play_time>
      <extension>
        [jp.ac.keio.sfc.ht.audio.variable.file_extension]
      </extension>
      <path>
        [jp.ac.keio.sfc.ht.audio.variable.filepath]
      </path>
      <protocol>
        [jp.ac.keio.sfc.ht.variable.protocol.transfer]
      </protocol>
    </data>
  <serviceA>
</handoff_send_protocol type>
```

送信 SH データ例：

```
<serviceA>
  <length unit="msec">1800</length>
  <data>
    <play_time unit="msec">1234</play_time>
    <extension>.mp3</extension>
    <path>http://hoge/fuge.mp3</path>
    <protocol>http</protocol>
  </data>
<serviceA>
```

図 4.6: データタイプが XML の場合の usdl 記述例（送信側）と送信 SH データ例

usdl 記述例（受信側）：

```
<handoff_receive_protocol data_type="xml">
  <serviceB>
    <data>
      <play_time unit="msec">
        [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_msec]
      </play_time>
      <extension>
        [jp.ac.keio.sfc.ht.audio.variable.file_extension]
      </extension>
      <path>
        [jp.ac.keio.sfc.ht.audio.variable.filepath]
      </path>
      <protocol>
        [jp.ac.keio.sfc.ht.variable.protocol.transfer]
      </protocol>
    </data>
    <length unit="msec">
      [jp.ac.keio.sfc.ht.audio.variable.time_msec]
    </length>
  </serviceB>
</handoff_receive_protocol>
```

受信 SH データ例：

```
<serviceB>
  <data>
    <play_time unit="msec">1234</play_time>
    <extension>.mp3</extension>
    <path>http://hoge/fuge.mp3</path>
    <protocol>http</protocol>
  </data>
  <length unit="msec">1800</length>
</serviceB>
```

図 4.7: データタイプが XML の場合の usdl 記述例（受信側）と受信 SH データ例

データタイプが CSV の場合

SH データの送信側（handoff_send_protocol 要素），受信側（handoff_receive_protocol 要素）のどちらの場合においても，自サービスが対応する SH データのデータタイプが CSV の場合，data_type 属性で “csv” を指定することでハンドオフデータのデータタイプが CSV であることを示すことが可能である．また SH データの送信側（handoff_send_protocol 要素）の場合，子要素に csv_data 要素を持ちその子要素に実際の CSV データの実データ値部分を変数に置き換えたようにカンマ区切りで記述することで，SH データの構造情報と意味情報を記述することが可能である．SH データの受信側（handoff_receive_protocol 要素）の場合，子要素に usdl:csv 要素を持ち，その子要素に実際の CSV データの実データ値部分を変数に置き換え usdl:csv_data 要素で変数をはさんだように記述することで，SH データの構造情報と意味情報を記述することが可能である．図 4.8 に usdl 記述例（送信側）と送信 SH データ例，図 4.9 に usdl 記述例（受信側）と受信 SH データ例を示す．

usdl 記述例（送信側）：

```
<handoff_send_protocol data_type="csv" requestMes="HandOff_request_csv">
  <csv_data>
    [jp.ac.keio.sfc.ht.audio.variable.time_msec]
    [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_msec],
    [jp.ac.keio.sfc.ht.audio.variable.file_extension],
    [jp.ac.keio.sfc.ht.audio.variable.filepath],
    [jp.ac.keio.sfc.ht.variable.protocol.transfer]
  </csv_data>
</handoff_send_protocol>
```

送信 SH データ例：

```
    1800,1234,.mp3,http://hoge.fuge.mp3,http
```

図 4.8: データタイプが CSV の場合の usdl 記述例（送信側）と送信 SH データ例

図 4.8 の usdl 記述例と実際に送信される SH データを比較して見ると分かるが，USDL の handoff_send_protocol 要素は実際の CSV データの実データ値部分を変数に置き換えたようにカンマ区切りで記述する．また，図 4.9 の usdl 記述例と実際に送信される SH データを比較して見ると分かるが，handoff_receive_protocol 要素は実データ値を変数に置き換え，usdl:csv 要素の子要素として usdl:csv_data 要素ではさんで記述する．これらの記述により，サービスは送受信可能な SH データの構造情報と意味情報を共に uMediator に通知する事が可能となる．

usdl 記述例（受信側）：

```
<handoff_receive_protocol data_type="csv">
  <usdl:csv>
    <usdl:csv_data>
      [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_sec]
    </usdl:csv_data>
    <usdl:csv_data>
      [jp.ac.keio.sfc.ht.audio.variable.file_extension]
    </usdl:csv_data>
    <usdl:csv_data>
      [jp.ac.keio.sfc.ht.audio.variable.filepath]
    </usdl:csv_data>
    <usdl:csv_data>
      [jp.ac.keio.sfc.ht.variable.protocol.transfer]
    </usdl:csv_data>
    <usdl:csv_data>
      [jp.ac.keio.sfc.ht.audio.variable.time_msec]
    </usdl:csv_data>
  </usdl:csv>
</handoff_receive_protocol>
```

受信 SH データ例：

```
    1234,.mp3,http://hoge.fuge.mp3,http,1800
```

図 4.9: データタイプが CSV の場合の usdl 記述例（受信側）と受信 SH データ例

4.3.3 usdl:repeat 要素

usdl:repeat 要素は子要素として持つデータの繰り返し制御を可能にする要素であり，handoff_send_protocol 要素または handoff_receive_protocol 要素の子要素以下で記述可能である．num 属性で指定した回数の繰り返しを許可する．回数の指定は数値のみでなく不等号を利用し指定する事が可能である．但し，符号はプログラムで解釈する際に正常に解釈されるよう特殊文字で記述する必要がある．つまり “>” は “>”，“<” は “<” と記述する．不等号を用いた場合の例を以下に示す．

-要素名 usdl_repeat

-属性 num

-出現数 0 以上

例：

```
<usdl:repeat num="&gt;=0" >
    <usdl:csv_data>[jp.ac.keio.sfc.ht.audio.variable.time_msec]</usdl:csv_data>
</usdl:repeat>
```

4.3.4 usdl:limit 要素

usdl:limit 要素は子要素として持つデータ値の制限を可能にする要素であり，handoff_send_protocol 要素または handoff_receive_protocol 要素の子要素以下で記述可能である．data 属性で指定した条件をデータ値が満たすか判断し，満たさない場合はそのデータ値の親要素の追加を許可しない．但し，条件はプログラムで解釈する際に正常に解釈されるように正規表現で記述する必要がある．正規表現で条件を設けデータ値の制限を行う場合の例を以下に示す．本例における変数はオーディオファイルのパスを表しており，usdl:limit の制限は拡張子が “mp3”，“wma”，“m4p” のいずれかの場合のみデータの追加を許可することを示している．

-要素名 usdl_limit

-属性 data

-出現数 0 以上

例：

```
<usdl:limit data=".*mp3|.*wma|.*m4p">
    <path>[jp.ac.keio.sfc.ht.audio.variable.filepath]</path>
</usdl:limit>
```

4.3.5 変数定義

変数は型として type 要素に記述することで定義する．変数名は通常の型記述における id 要素の値を使用する．通常の型の記述方法と異なる点は，relationships 要素を拡張記述する点である．型の定義はユーザが自由に継承，拡張定義可能であるため，変数も同様にユーザが自由に定義する事が可能である．relationships 要素については 4.3.6 項で詳細を述べると共に変数定義の実例を示し解説する．

4.3.6 relationships 要素と relationship 要素

relationships 要素は子要素として relationship 要素を複数持つ事が可能で，relationship 要素の子要素に他の変数との関係性を記述する．関係性の記述には四則演算子を用いる．以下で relationships 要素と relationship 要素の設計を行うと共に，変数定義の実例を示す．

-要素名 relationships

-属性 なし

-出現数 0 か 1

-要素名 relationship

-属性 なし

-出現数 1 以上

例：

```
<type>
    <properties>
        <id>jp.ac.keio.sfc.ht.variable.audio_variable.play_time_sec</id>
        <inherit>jp.ac.keio.sfc.ht.variable.sec</inherit>
        <version>1.0.0</version>
    </properties>
         ...
    <relationships>
        <relationship>
            [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_min]*60
        </relationship>
        <relationship>
            [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_msec]/1000
        </relationship>
```

```
    </relationships>
</type>
```

上記例はオーディオ再生時間を秒で示す変数 [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_sec] の定義である．relationships 要素ではオーディオ再生時間を分で示す変数 [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_min] に 60 を掛ける，またはオーディオ再生時間をミリ秒で示す変数 [jp.ac.keio.sfc.ht.variable.audio_variable.play_time_msec] を 1000 で割ることで本定義変数を求める事が可能であるという関係性を示している．

4.4 uMediator の設計

本節では uMediator の設計について解説する．uMediator は uMediator 処理管理部と USDL 受信部，異種サービス判定部，USDL 解析部，Usdl Control Server 問合せ部，SH 先プロトコル情報取得部，SH データ取得&解析部，データマッピング部，欠如データ算出部，SH データ作成部，SH データ送信部から成る．uMediator 処理管理部は各処理部間のデータ受け渡しを行い，uMediator 全体の処理を管理する．USDL 受信部は SH 元サービスと SH 先候補サービスの usdl ファイルのパスを SH 管理機能より受信し，異種サービス判定部へ受信データを渡す．異種サービス判定部は USDL 解析部を利用し SH 元サービスと SH 先候補サービスの usdl ファイルから異種サービスの判定を行い，SH 先サービスを決定する．そして，SH 先サービス USDL 情報を SH 先プロトコル情報取得部へ，SH 元サービス USDL 情報を SH データ取得&解析部へ渡し処理を渡す．SH 先プロトコル情報取得部は SH 先サービス USDL より，必要となる変数名や作成データの構造データ等の SH プロトコル情報と Ip や Port 番号などの情報を取得し，データマッピング部，SH データ作成部，SH データ送信部に取得データを渡す．SH データ取得&解析部は SH 先サービスへ SH データのリクエストを行いデータを取得し，SH データを解析することで変数名とデータ値のリストを取得しデータマッピング部に渡す．データマッピング部は SH 先プロトコル情報取得部と SH データ取得&解析部から受信したデータをもとに，SH データの再構築に必要となる変数名とデータのマッピングを行う．この際，必要データが必ずしも全て揃うとは限らない．そのため，欠如データ算出部が揃わなかったデータを relationships 要素の情報から可能な限り算出する．これらの行程でデータのマッピングが無事完了した場合は，SH データ作成部で SH 先サービスの対応 SH プロトコルに応じた SH データの作成を行う．そして，SH 送信部において完成した SH データを SH 先サービスに送信する．図 4.10）に uMediator の構成を示し，次項より各処理部の詳細を述べる．

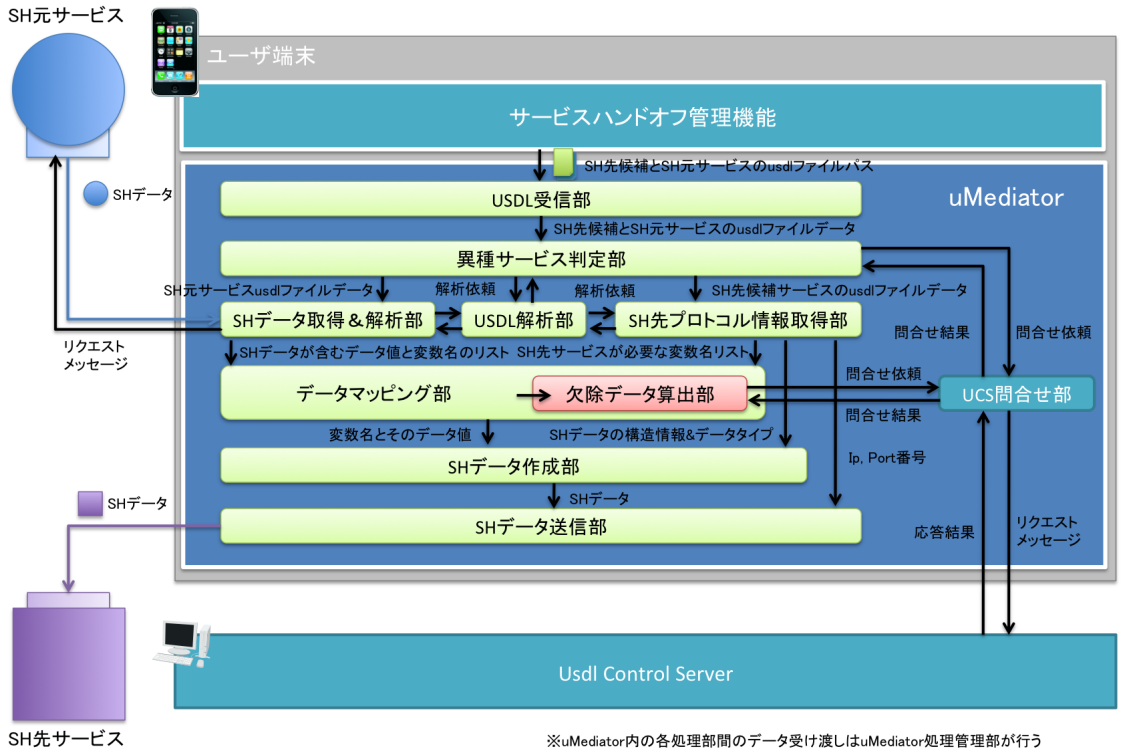

図 4.10: **uMediator の構成**

4.4.1 uMediator 処理管理部

uMediator 処理管理部は各処理部間のデータ受け渡しを行い，uMediator 全体の処理を管理する．

4.4.2 USDL 受信部

USDL 受信部は SH 元サービスと SH 先候補サービスの usdl ファイルのパスを受信し，メモリ上へ usdl ファイルを読み込みプログラムで処理可能な状態にする．そして，メモリ上のデータ参照を異種サービス判定部へ渡し，処理を渡す．

4.4.3 異種サービス判定部

異種サービス判定部は 4.4.4 項で解説する USDL 解析部と 4.5 節で解説する Usdl Control Server 問合せ部と連携し SH 元サービスと SH 先候補サービスの usdl ファイルから異種サービスの判定を行い，SH 先サービスを決定する．

異種サービス判定部は USDL 受信部で読み込まれた SH 元サービスと SH 先候補サービスの usdl ファイルデータを受信し，各々の受信データに対し次の処理を行うことで

各サービスの継承リストを作成する．

- USDL 解析部で受信データからサービスの id を取得
- 取得 id を Usdl Control Server 問合せ部に渡し Usdl Control Server に問い合わせを行うことで対象サービスの継承リストを取得

上記処理により得られた SH 元サービスと SH 先候補サービスの継承リストを利用し，各 SH 先候補サービスが SH 元サービスの継承の何ホップ目の継承と一致するか走査する．一番ホップ数の少なかったサービスを異種サービスと判定する．ホップ数が同一かつ最小のサービスが存在した場合は，先に走査したサービスを異種サービスと判定する．

4.4.4 USDL 解析部

USDL 解析部は各処理部において USDL を解析し情報の抽出が必要となる場合に使用される．ゆえに本処理部は usdl ファイルの情報に対してのアクセサメソッド群により構成される．

4.4.5 Usdl Control Server 問合せ部

Usdl Control Server 問合せ部は 4.5 節で解説される Usdl Control Server への問合せを行う処理部である．Usdl Control Server では各 usdl の継承リストと変数の関係性リストを所持しているため，異種サービス判定部と欠如データ算出部（4.4.9 項参照）からの問合せが行われる．この際に，各処理部から本処理部が呼び出され，Usdl Control Server への直接の問合せは本処理部が行う．

4.4.6 サービスハンドオフ先プロトコル情報取得部

SH 先プロトコル情報取得部は SH 先 usdl ファイルから，SH データ作成の際に必要となる値の変数名リスト，データ構造情報，データタイプ，SH 先サービスの Ip と Port 番号を取得する．SH データ作成の際に必要となる値の変数名リストはデータマッピング部に渡し，データマッピング部では SH データ取得&解析部で取得した SH データとのマッピングに備えることが可能になる．また，データ構造情報，データタイプは SH データ作成部へ，SH 先サービスの Ip と Port 番号は SH データ送信部へ渡す．SH データ作成部ではこれらの情報をもとに SH データの作成を行い，SH データ送信部ではこれらの情報を利用して SH 先サービスとのコネクションを確立する．以下で各 SH プロトコル情報の取得について述べる．

SH データ作成の際に必要となる値の変数名

USDL 解析部を利用し，handoff_receive_protocol 要素の全ての子要素に対し，自要素の子要素に変数が存在するかどうか再帰的に走査する．変数が存在した場合はその変数名を取得しリストとして保存し，全ての走査が終了したら結果をデータマッピング部に渡す．

作成する SH データの構造情報

作成する SH データ構造情報は usdl ファイルの handoff_receive_protocol 要素の子要素部分にあたる．4.4.10 項で詳細について述べるが，SH データ作成部では最終的に作成する SH データのデータタイプが何であろうと，まずは handoff_receive_protocol 要素の子要素部分で示された SH データ構造（XML の形体）で SH データを作成する．ゆえに handoff_receive_protocol 要素の子要素部分を再現可能な形で構造情報として取得する必要がある．

構造情報の取得は handoff_receive_protocol 要素の最初の子要素である SH データのルート要素以下に対して以下の処理を再帰的に行うことで図 4.11 で示すような SH データの構造情報を作成する．但し，handoff_receive_protocol 要素の最初の子要素である SH データのルート要素は初めに自要素の情報を保存するオブジェクトを作成してから以下の処理を行う．

- 自要素に属性が存在すれば属性名と属性値をリストとして親要素の処理で作成された自要素オブジェクトに保存
- 自要素にテキストノードが存在すれば親要素の処理で作成された自要素オブジェクトに保存
- 自要素に制御コマンド要素が存在すれば親要素の処理で作成された自要素オブジェクトに保存
- 自要素に子要素が存在すれば，各子要素に対してオブジェクトを作成しリストとして親要素の処理で作成された自要素オブジェクトに保存

作成した SH データの構造情報は，SH データ作成部に渡す．

作成データタイプ

最終的に作成する SH データのデータタイプに関する情報を取得する．作成する SH データのデータタイプは SH 元 usdl ファイルの handoff_receive_protocol 要素の data_type 属性として明記されているため USDL 解析部を利用して取得する．取得した作成データタイプは SH データ作成部に渡す．

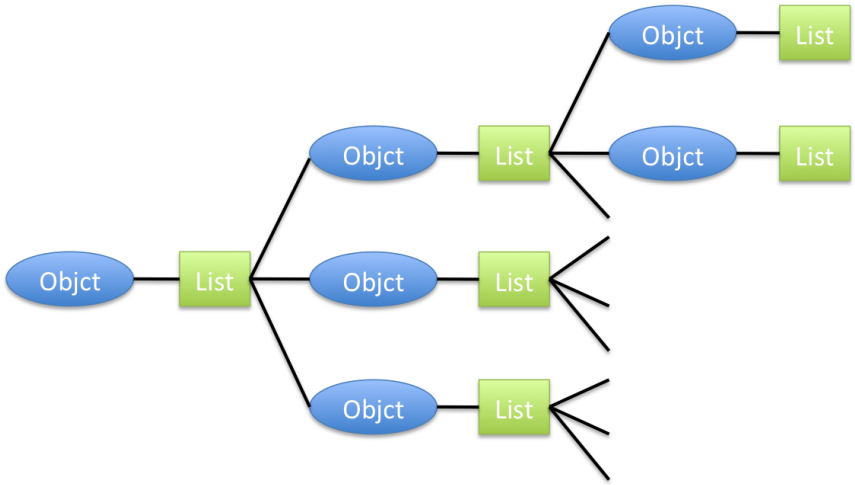

図 4.11: **作成する SH データの構造情報**

4.4.7 サービスハンドオフデータ取得&解析部

SH データ取得&解析部は，まず SH 元サービスの usdl ファイルから，SH 元サービスの Ip と Port，リクエストメッセージ情報，SH データのデータタイプを取得する．次に，それらの取得データを利用し SH 元サービスへ SH データのリクエストを送信し SH データを取得する．この際，データタイプに応じ SH データの受信待ちを行う．取得した SH データは解析し，SH データに含まれる変数名とデータ値のリストにしデータマッピング部に渡す．変数名とデータ値のリストは，SH 元サービスの usdl ファイルと SH データを比較しながら作成する．

4.4.8 データマッピング部

データマッピング部は SH 先プロトコル情報取得部と SH データ取得&解析部より受け取ったデータをもとに，SH で必要となる変数名とそれに対応するデータ値のマッピングを行いリストを作成する（図 4.12）．作成リストデータは SH データ作成部へ渡す．

データマッピングは SH 先プロトコル情報取得部より受け取った変数名リストと対応する値が存在するか，SH データ取得&解析部より受け取った変数名とデータ値のリストを変数名ごとに走査することで行う．変数名が一致した場合は SH データ作成に必要な変数名と対応データ値を新たにリストを作成し保存する．この際，一致した変数名と値は SH データ取得&解析部より受け取ったリストから削除する．これは，同一データが重複して走査され保存されるのを防ぐためである．

また，最終的に SH データ作成に必要なデータを SH 元の SH データから得られない場合も存在する．その場合は，欠如データ算出部へ欠如データの変数名を渡すことで

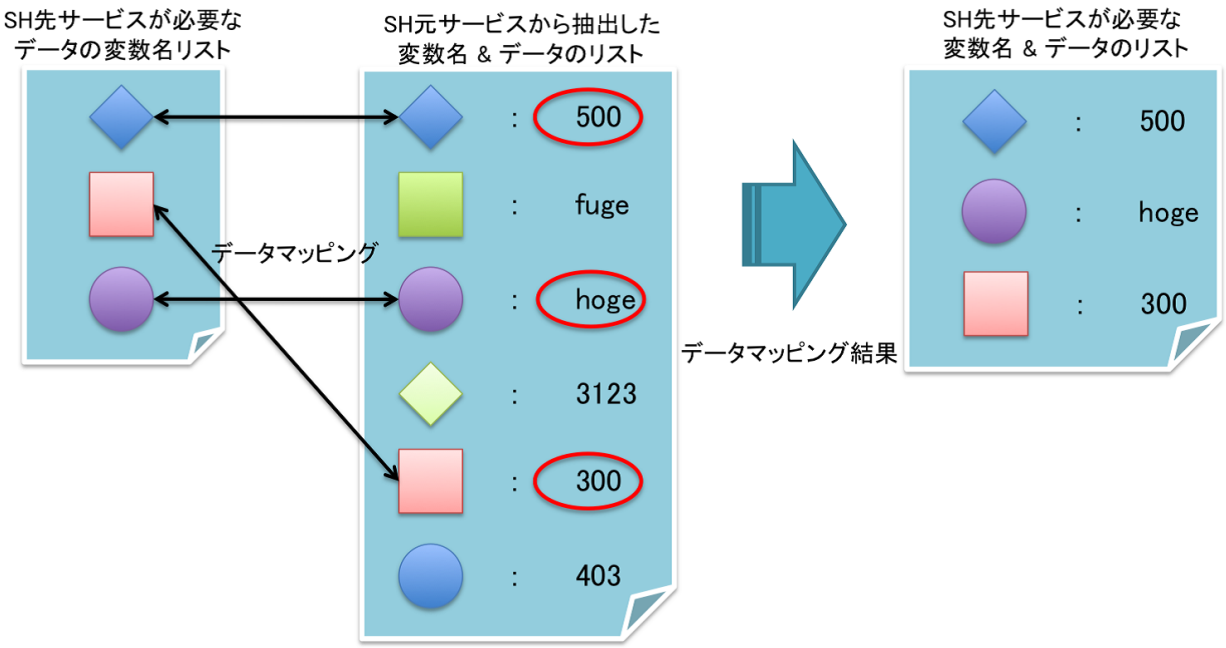

図 4.12: **データマッピングイメージ図**

欠如データの算出を試みる．欠如データ算出部については 4.4.9 項で解説する．

4.4.9　欠如データ算出部

欠如データ算出部はデータマッピング部で SH データ作成に必要となるデータの欠如が確認された場合にデータの算出を試みる．欠如データの変数の関係性を Usdl Control Server 問合せ部から Usdl Control Server に問合せ，他の変数との関係性を取得する．受信 SH データの変数値から算出可能な関係性が存在するか走査し，存在する場合はそれを利用しデータの算出を行う．算出データはデータマッピング部に渡し，データマッピングの続きを行う．

4.4.10　サービスハンドオフデータ作成部

SH データ作成部は SH 先プロトコル情報取得部から受け取った SH データの構造情報と SH データ種類情報，データマッピング部より受け取った変数名と対応データ値のリストから SH データの作成を行う．

構造情報が保存されたオブジェクト（図 4.11）に対し，以下の処理を再帰的に行うことで handoff_receive_protocol 要素の子要素として示された XML 構造を再現し XML 形式の SH データを作成する．

- オブジェクトから自要素名を取得し要素を作成
- 属性が存在する場合は属性と属性値を作成要素に付加
- 子要素としてテキストノードが存在する場合は値を要素の子要素として付加
- 子要素として制御コマンド要素が存在する場合は，その対象となる子要素が条件を満たすか判断するためそのオブジェクトへ処理を移す
- テキストノードでも制御コマンド要素でもない子要素が存在する場合はその要素情報が含まれるオブジェクトに処理を移す
- 子要素が制御コマンドの条件を満たす場合，または制御コマンドがない場合は本要素を親要素に付加

SH データ種類が XML の場合

作成した XML 形式の SH データを SH データ送信部に渡す．

SH データ種類が CSV の場合

作成した XML 形式の SH データを解析し再度 CSV へ作り替える．SH データのデータタイプが CSV の場合の usdl ファイル記述は，4.3.2 項で述べた USDL 拡張記述のデータタイプ別 SH データの構造&意味情報記述に従って行われる．そのため，usdl:csv 要素の子要素である usdl:csv_data 要素の子要素であるテキストノードを全て取得し，カンマ区切りで保存することで CSV の作成が可能である．

4.4.11 サービスハンドオフデータ送信部

SH データ送信部は SH 先プロトコル情報取得部より受信した Ip と Port 番号を利用し，SH 先サービスとコネクションを確立する．その後 SH データを送信し SH データの仲介を完了する．

4.5 Usdl Control Server の設計

本節では Usdl Control Server の設計について解説する．Usdl Control Server は uMediator からの問合せに応答する uMediator 処理系と Usdl Delivery Server（UDS）からの新規登録 usdl ファイル情報の処理を行う UDS 処理系，データベース制御部，Usdl Information データベースにより構成される（図 4.13）．uMediator 処理系は uMediator 接続待機部と uMediator リクエスト応答部，UDS 処理系は UDS 接続待機部と USDL 解析部より構成される．次項より各処理部の詳細を述べる．

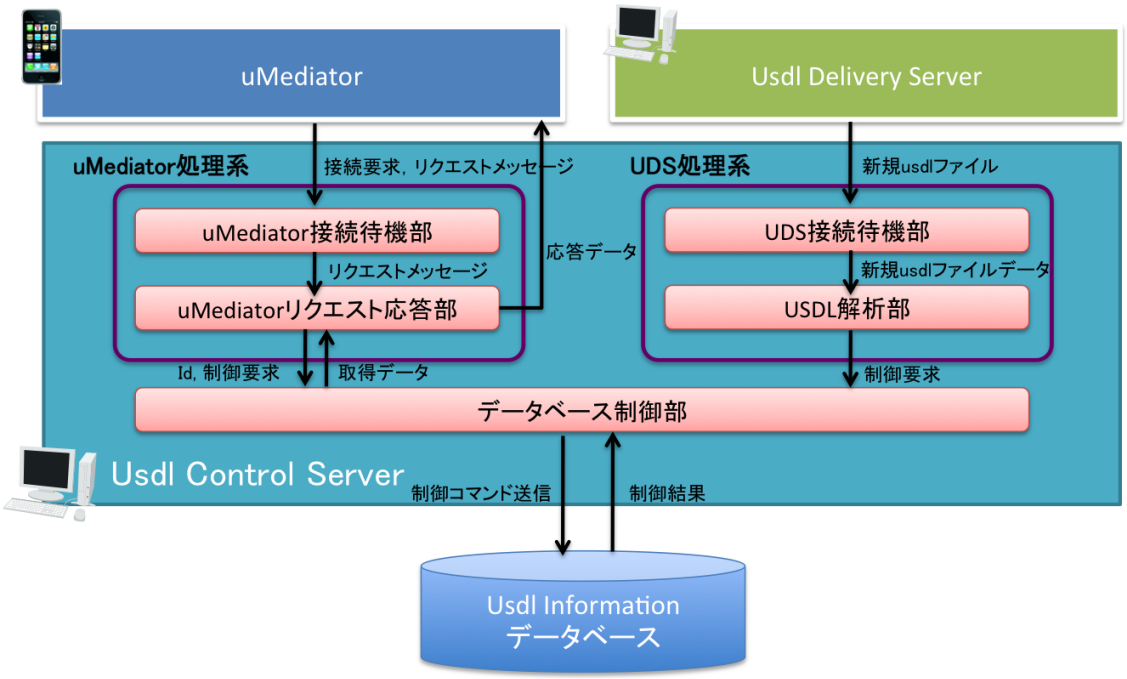

図 4.13: **Usdl Control Server の構成**

4.5.1 uMediator 接続待機部

uMediator 接続待機部は uMediator からの接続待ちを行う．uMediator から接続依頼があると新規チャネルを作成する．それにより複数の接続依頼があった場合にも個別に対応可能になる．

4.5.2 uMediator リクエスト応答部

uMediator リクエスト応答部は uMediator 接続待機部からチャネルとして作成され，uMediator からのリクエストに応答する．まず，uMediator からのリクエストメッセージを受信し解釈する．リクエスト内容は指定 id の継承リスト要求と関係性リスト要求の二種類である．各要求を受けた場合，本処理部ではデータベース制御部（4.5.3 項参照）へ要求情報の取得を依頼し，Usdl Information データベース（4.5.4 項参照）から情報を取得する．取得情報を uMediator に返信することでリクエストの応答を完了する．取得情報の返信はリクエストを受信した際のコネクションを利用して行う．

4.5.3 データベース制御部

データベース制御部は 4.5.4 項で解説する Usdl Information データベースの制御を行う処理部である．制御内容は uMediator リクエスト応答部からの依頼に応じて inherit,

relationship のデータ取得，また USDL 解析部で解析・取得した新規 usdl ファイル情報の登録の二つである．

4.5.4 Usdl Information データベース

本 Usdl Control Server に属する Usdl Delivery Server に登録された usdl ファイルの id と他の usdl ファイルとの関係性を保存するデータベースである．また，本データベースの操作はデータベース制御部からのみ行われる．

4.5.5 Usdl Delivery Server 接続待機部

UDS 接続待機部は Usdl Delivery Server からの接続待ちを行う．Usdl Delivery Server から接続依頼があると新規チャネルを作成する．それにより複数の接続依頼があった場合にも個別に対応可能になる．また，uMediator 接続待機部とは受信データの形体が異なるため，別ポートで待機することで処理系統を分離しその後の処理を行いやすくする．

4.5.6 USDL 解析部

USDL 解析部は UDS 接続待機部からチャネルとして作成され，USDL データを受信し id 要素と inherit 要素, relationships 要素の値を解析・取得し，データベース制御部を利用し Usdl Information データベースに登録する．この際，inherit 値と relationships 値はそのまま登録するのではなく，既にデータベースに登録されているデータから補足可能かどうか走査し，補足したデータを登録する．以下に inherit の走査と relationship の走査について解説する．

inherit の走査

受信 USDL データから取得した inherit 値の継承情報をデータベースから取得する．つまり受信 USDL データの継承元 id の継承情報をデータベースから取得する．取得データがカンマ区切りで複数存在する場合は，それらが受信 USDL データの継承リストである．ゆえに，それらをデータベースに登録する．また，取得値が一つの場合はさらにその取得データに対し，inherit 情報をデータベースから取得し値がカンマ区切りで一つ以上存在するか調べ，複数存在する場合は先ほどと同様に処理する．値が一つの場合は，さらに取得値に対し inherit 情報をデータベースから取得するということを繰り返す．もし，inherit 値が存在しなくなった場合はそこまでに走査した値をカンマ区切りにしたものが継承リストとなる．ゆえにその継承リストを inherit 値として Usdl Information データベースに登録する．

relationship の走査

受信 USDL データから取得した relationships 値をデータベースから取得する．取得データに変数を含む式が存在する場合は，その変数を導く関係性の式も変数に代入し新たな関係性の式になる．ゆえに，データベースから変数名を示す id の relationship 値を取得し，それを代入した式も新たなリストとして追加する．これを再帰的に行いデータベースに登録することで一つの変数に対して多くの関係性を式で表現することが可能になり，欠如データ算出部での算出成功可能性を向上させることが可能になる．

4.6 本章のまとめ

本章では，まず SH を可能にする SH フレームワークの構成について述べ，本研究の対象機能 を明らかにした．その後，サービス記述言語拡張として USDL 拡張記述の設計について述べ，対象機能を実現するミドルウェアとして uMediator， uMediator の連携サーバとして Usdl Control Server の設計について述べた．

USDL 拡張記述によって，各サービスの SH プロトコル情報をサービス情報の一部として USDL 記述可能にした．uMediator は SH 管理機能から受信した usdl ファイルをもとに，異サービス群からユーザ利用中サービスの異種サービスを判定することで SH 先サービスを発見し，SH データの変換を行うことで仲介可能なように設計した．Usdl Control Server は任意の範囲に存在するサービスの関係性をデータベース化し，uMediator からの問合せに対し応答することで SH の支援を可能なように設計した．

次章では，本章の設計をもとにした uMediator と Usdl Control Server の実装について述べる．

第5章

実装

本章では，異種サービス間でのハンドオフ支援機構である uMediator と，連携サーバである Usdl Control Server のプロトタイプ実装について述べる．

5.1 実装環境

本節では実装環境について述べる．本研究では，実装環境として Java[16] と MySQL[20] を用いた．また，データベースのアクセスを行うライブラリとして JDBC（Java DataBase Connectivity）ドライバ [21] を用いた．表 5.1 に詳細を示す．

表 5.1: **実装環境**

実装言語	Java5.1.30
データベース	MySQL（バージョン 5.1.30）
JDBC ドライバ	Connector/J 5.0

5.2 実装概要

本節では，本研究でプロトタイプとして実装した，SH 支援機構 uMediator と，連携サーバ Usdl Control Server の実装概要について述べる．

5.2.1 uMediator

本項では，uMediator の実装概要について述べる．本研究で実装した uMediator は，まず SH 元サービスと SH 先候補群サービスの usdl ファイルから id を取得し，Usdl Control Server に問合せを行うことで継承リストを取得する．次に，各 SH 先候補の継承リストと SH 元サービスの継承リストを比較し，SH 元サービスの継承と一致する継承の有無とホップ数を調べ，SH 元サービスと最も継承が近いものを異種サービスを判定する事で SH 先サービスを発見する．次に SH 元サービスへリクエストを送信し SH データを取得し，SH 元サービスの usdl ファイルと照合することで変数名とデータ値のリストを作成する．SH 先サービスの usdl ファイルより SH に必要となるデータの変数名を取得しておき，必要データの変数名とデータ値のリストをデータマッピングし作成する．作成したデータは SH 先サービスの取得可能なデータ形式に応じて SH データとして再構築する．再構築後の SH データは SH 先プロトコルに従い，SH 先サービスへ送信される．uMediator のクラス構成を表 5.2 に示す．これらのクラスは umediator パッケージに含まれる．

次に，各クラスの概要について述べる．UMediator クラスは uMediator のメインクラスであり SH 管理機能からの接続待ちをする．UMediatorManager クラスは SH 管理機能より UMediator クラスに接続があると生成されるクラスであり，各クラス間のデータや処理の管理を行う．また，SH 管理機能より usdl ファイルを受信するクラスでもあり，設計で述べた uMediator 処理管理部と USDL 受信部にあたる．HomogeneousServiceJudge クラスは受信した usdl ファイルを利用し，Usdl Control Server 問合せを行うことで異

表 5.2: **uMediator のクラス構成**

UMediator	uMediator のメインクラス
UMediatorManager	uMediator の処理管理を行うクラス
HomogeneousServiceJudge	異種サービスの判定を行うクラス
DepartureUsdlParser	SH 元 usdl ファイルのデータを保持し，解析を行うクラス
DestinationUsdlParser	SH 先 usdl ファイルのデータを保持し，解析を行うクラス
UCSRequestSender	Usdl Delivery Server へ問い合わせを行うクラス
HandoffProtocolGetter	SH 先のプロトコル情報取得を行うクラス
HandoffDataGetterAndParser	SH データの取得と解析を行うクラス
HandoffXmlParser	SH データ（XML）を保持し，解析を行うクラス
HandoffCsvParser	SH データ（CSV）を保持し，解析を行うクラス
DataMapper	データのマッピングを行うクラス
LackDataCalculator	欠如データの算出を行うクラス
XmlArchitecture	SH データの XML 構造を保存するクラス
XmlCreater	XML 形式の SH データを作成するクラス
CsvCreater	CSV 形式の SH データを作成するクラス
HandoffDataSender	SH 先へ SH データを送信するクラス

種サービスの判定を行うクラスであり，設計で述べた異種サービス判定部にあたる．DepartureUsdlParser クラスと DestinationUsdlParser クラスは usdl ファイルの解析を行うクラスであり，設計で述べた USDL 解析部にあたる．UCSRequestSender クラスは Usdl Control Server への問合せを行うクラスであり，設計で述べた Usdl Control Server 問合せ部にあたる．HandoffProtocolGetter クラスは SH 先 usdl ファイルより SH プロトコル情報を取得するクラスであり，設計で述べた SH 先プロトコル情報取得部にあたる．HandoffDataGetterAndParser クラスは SH 元サービスへ SH データのリクエストメッセージを送信し，SH データを取得し解析するクラスである．解析の際は，データタイプにより HandoffXmlParser クラスと HandoffCsvParser クラスを呼び分けて解析する．HandoffXmlParser クラスは HandoffDataGetterAndParser クラスが取得した SH データが XML の場合に呼び出され，解析を行うクラスである．HandoffCsvParser クラスは HandoffDataGetterAndParser クラスが取得した SH データが CSV の場合に呼び出され，解析を行うクラスである．解析結果として SH データに含まれる変数名と値をリストとして作成する．これらは設計で述べた SH データ取得&解析部にあたる．DataMapper クラスは HandoffProtocolGetter クラスと HandoffDataGetterAndParser クラスにより収集されたデータをもとに，SH データ作成に必要な変数名と値のリストをマッピングして作成するクラスである．設計で述べたデータマッピング部にあたる．LackDataCalculator クラスは DataMapper クラスで SH データ作成に必要なデータを全てマッピングできず，欠如データが発生した場合に，Usdl Control Server に欠如データの変数が他の変数から算出できるかを問合せ，欠如データの算出を試みるクラスである．設計で述べた欠如データ算出部にあたる．XmlArchitecture クラスは SH 先が対応している SH データの XML 構造を保存するクラスである．XmlCreater クラスは DataMapper クラスにより作成された変数名と値のリスト，HandoffProtocolGetter クラスにより作成されたハンドオフデータの構造データをもとに，XML 形式の SH データを作成するクラスである．CsvCreater クラスは SH データタイプが CSV の場合，XmlCreater クラスにより生成された XML を解析し CSV に変換するクラスである．XmlCreater クラスと CsvCreater クラスはともに SH データ作成部にあたる．HandoffDataSender クラスは XmlCreater クラスまたは CsvCreater クラスにより作成された SH データを SH 先サービスに送信するクラスであり，設計で述べた SH 送信部にあたる．図 5.1 に uMediator のシステム構成図とクラスとの対応関係を示す．

5.2.2 Usdl Control Server

本項では Usdl Control Server の実装概要について述べる．本研究でプロトタイプ実装した Usdl Control Server は，設計で示した UDS 処理系を除く，uMediator 処理系とデータベース制御部，Usdl Information データベースである．本研究では異種サービス間での SH 支援を実現する機構 uMediator の開発を目的としているため，Usdl Control Server の実装は uMediator の処理に直接関係する部分のみ行なった．Usdl Information データベースは設計に従い，UDS 処理系によりデータが追加されたようにテストデー

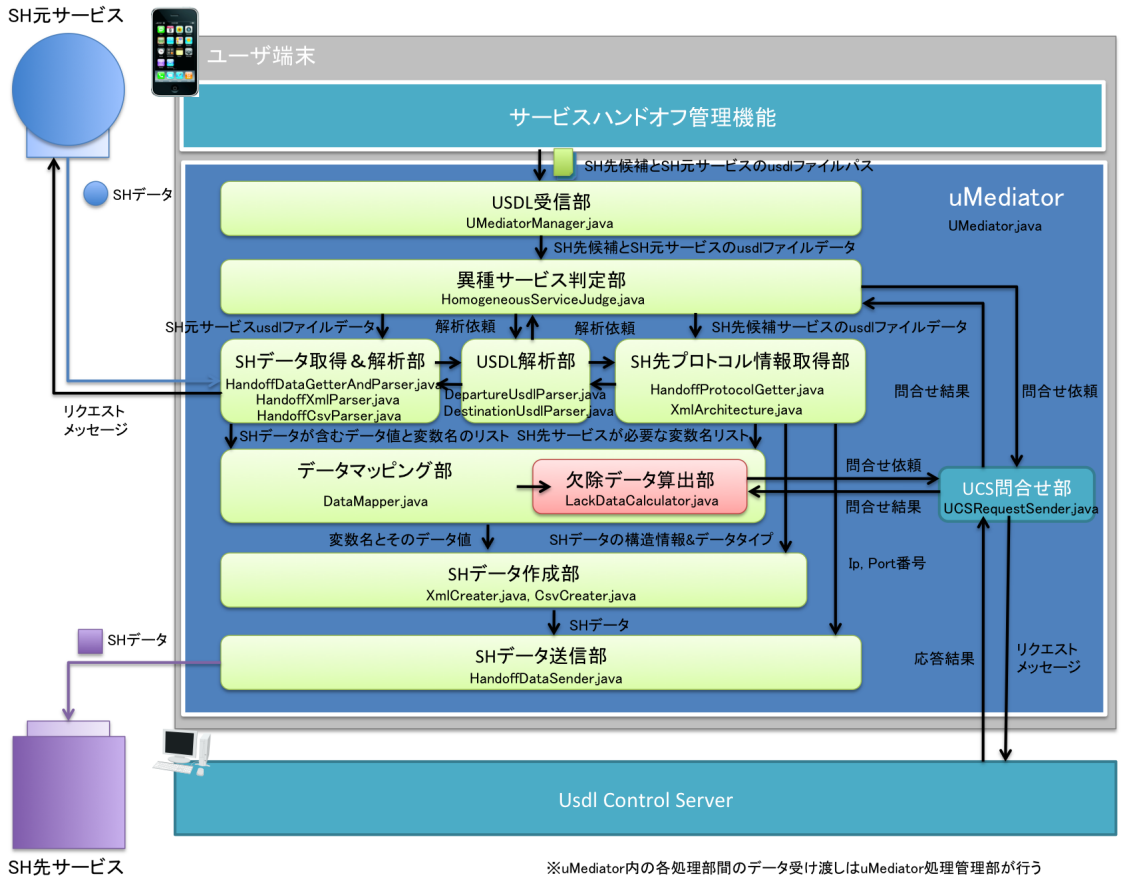

図 5.1: **uMediator システム構成とクラスとの対応関係**

タを登録した．Usdl Control Server のクラス構成を表 5.3 に示す．これらのクラスは ucs パッケージに含まれる．

次に，各クラスの概要について述べる．UCSMain クラスは uMediator 処理系のメインクラスであり，uMediator からの接続を待機する．設計で述べた uMediator 接続待機部にあたる．UCSReplier クラスは uMediator からのリクエストを解釈し，Usdl Information データベースより必要情報を収集 uMediator に送信するクラスである．設計で述べた uMediator リクエスト応答部にあたる．DbController クラスは Usdl Information データベースの制御を行うクラスであり，設計で述べたデータベース制御部にあたる．図 5.2 に Usdl Control Server システム構成図とクラスとの対応関係を示す．

5.3 uMediator の実装

本章では，uMediator の実装について述べる．uMediator の機能要件は，SH 先サービスを発見し，SH 元サービスからの SH データを SH 先サービスが解釈可能な形に変

表 5.3: **Usdl Control Server のクラス構成**

UCSMain	Usdl Control Server の uMediator 処理系のメインクラス
UCSReplier	uMediator のリクエスト応答を行うクラス
DbController	データベースの制御を行うクラス

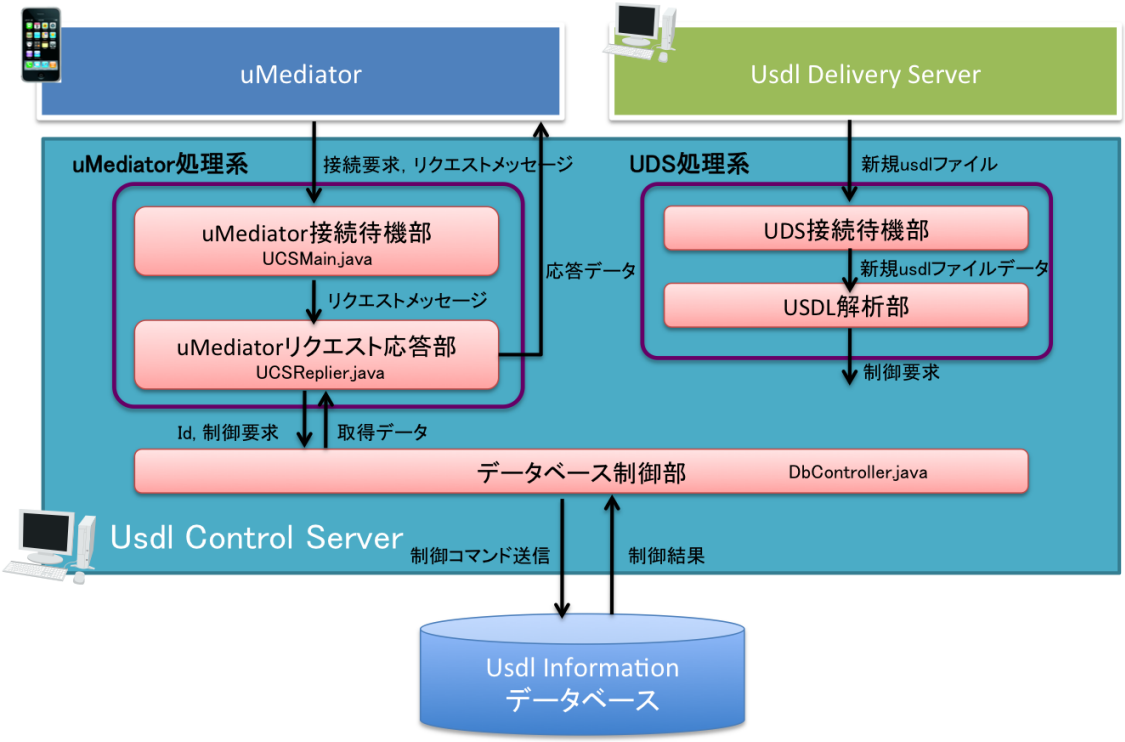

図 5.2: **Usdl Control Server システム構成とクラスとの対応関係**

換し仲介することである．これは uMediator の設計で述べた各処理部によって実現可能である．ゆえに，本研究では uMediator の設計で述べた各処理部に対応するよう実装を行った．本章では，設計で述べた各処理部の順で対応するクラスについて述べる．

5.3.1 uMediator 処理管理部

uMediator 処理管理部は UMediatorManager クラスが担う．UMediatorManager クラスは UMediator クラスによりスレッドとして作成されるため Thread クラスを extends する．また，各処理部のインスタンスを生成しメソッドを呼び出すことで，各処理部に処理とデータを渡し，uMediator 全体の処理を管理する．図 5.3 に UMediatorManager クラスの概要を示す．

```
package umediator;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
 ...

public class UMediatorManager extends Thread {
    UMediator server;
    Socket sok_app = null;
    ObjectInputStream in_app = null;// 入力用ストリーム
    ObjectOutputStream out_app = null;// 出力用ストリーム
        ...
    public UMediatorManager(Socket s, Main_middleware cs) {
        server = cs;
        sok_app = s;
        start();
    }

    public void run() {
        ...
        HomogeneousServiceJudge homogeneousServiceJudge =
                                        new HomogeneousServiceJudge();
        UCSRequestSender UCSRequestSender = new UCSRequestSender();
        HandoffProtocolGetter handoffProtocolGetter =
                                        new HandoffProtocolGetter();
        HandoffDataGetterAndParser handoffDataGetterAndParser =
                                        new HandoffDataGetterAndParser();
        DataMapper dataMapper = new DataMapper();
        LackDataCalculator lackDataCalculator = new LackDataCalculator();
        XmlCreater xmlCreater = new XmlCreater();
        HandoffDataSender handoffDataSender = new HandoffDataSender();
        ...
    }
}
```

図 5.3: UMediatorManager クラスの概観

5.3.2 USDL 受信部

USDL 受信部は uMediator 処理管理部を担う UMediatorManager クラスが兼任して担う．UMediatorManager クラスは図 5.3 で示したように，UMediator クラスにより生成される際に，SH 管理機能とのソケットを生成するので，このソケットを利用し usdl ファイルの保存先パスを受信する．また，受信した保存先パスより usdl ファイルごとにメモリ上にデータの読み込みを行う．読み込み後は何度も usdl 解析を行うため，USDL 解析部のインスタンスとして読み込む．詳細は 5.3.4 項で述べるが，USDL 解析部は DepartureUsdlParser クラスと DestinatioUsdlParser クラスが担っている．DepartureUsdlParser クラスは SH 元サービスの usdl ファイルを，DestinatioUsdlParser クラスは SH 先サービスの usdl ファイルを解析するクラスである．usdl ファイルパスの受信と usdl ファイル読み込みの詳細を図 5.4 に示す．

5.3.3 異種サービス判定部

異種サービス判定部は HomogeneousServiceJudge クラスが担う．HomogeneousServiceJudge クラスが保持するメソッドを表 5.4 に示す．

表 5.4: **HomogeneousServiceJudge クラスのメソッド**

`public ArrayList<String> getInheritList(DepartureUsdlParser departure)`	SH 元サービスの継承リストの取得
`public ArrayList<String> getInheritList(DestinationUsdlParser destination)`	SH 先サービスの継承リストの取得
`public DestinationUsdlParser judgeHomogeneousService( ArrayList<ArrayList<String>> allInheritList, ArrayList<String> departureInherit, ArrayList<String> filePathList)`	異種サービスの判定

また，本クラスのメインである異種サービスの判定を行う judgeHomogeneousService メソッドの詳細を図 5.5 に示す．

5.3.4 USDL 解析部

USDL 解析部は DepartureUsdlParser クラスと DestinatioUsdlParser クラスが担っている．DepartureUsdlParser クラスは SH 元サービスの usdl ファイルを，DestinatioUs-

```
try {
    // SH 元 usdl ファイルパス取得
    usdlpath_dep = (String) (in_app.readObject());
    // SH 先 usdl ファイルパスリスト取得
    usdlpath_des = (ArrayList<String>) in_app.readObject();
} catch (Exception e) {
    e.printStackTrace();
}

// SH 元 usdl ファイルデータの読み込み
DepartureUsdlParser departure = new DepartureUsdlParser(usdlpath_dep);

// SH 先 usdl ファイルデータのインスタンスを保持するリストの生成
ArrayList<DestinationUsdlParser> destinationList =
                                    new ArrayList<DestinationUsdlParser>();

// SH 先 usdl ファイルデータの読み込みを行う繰り返し
for (int i = 0; i < usdlpath_des.size(); i++) {
    // SH 先 usdl ファイルデータの読み込み
    DestinationUsdlParser destination =
                                    new DestinationUsdlParser(usdlpath_des.get(i));
    // SH 先 usdl ファイルデータのインスタンスを ArrayList に追加
    destinationList.add(destination);
}
```

図 5.4: usdl ファイルの読み込み部分

```
public DestinationUsdlParser judgeHomogeneousService(
                    ArrayList<ArrayList<String>> allInheritList,
                    ArrayList<String> departureInherit, ArrayList<String> filePathList) {

    // 走査結果を格納する ArrayList 生成
    ArrayList<Integer> resultList = new ArrayList<Integer>();

    // allInheritList の各継承が departureInherit の継承と何ホップ目で一致するか走査
    for (int i = 0; i < allInheritList.size(); i++) {
        for (int j = 0; j < allInheritList.get(i).size(); j++) {
            for (int k = 0; k < departureInherit.size(); k++) {
                if (allInheritList.get(i).get(j).equals(departureInherit.get(k))) {
                    resultList.add(k);
                }
            }
        }
    }

    // 最小の継承数を格納
    int min = 0;
    // ArrayList のインデックス
    int index = 0;
    // 捜査結果の中でホップ数が最小の継承を探す
    for (int i = 1; i <= resultList.size(); i++) {
        int min_tmp = resultList.get(i - 1);
        if (min == 0 || min > min_tmp) {
            min = min_tmp;
            index = i - 1;
        }
    }

    // 返り値の DestinationUsdlParser インスタンス生成
    DestinationUsdlParser homogeneousService =
                                        new DestinationUsdlParser(filePathList.get(index));
    return homogeneousService;
}
```

図 5.5: judgeHomogeneousService メソッド

dlParser クラスは SH 先サービスの usdl ファイルを解析するクラスである．これら二つのクラスは同じ様なメソッドやフィールドも含むが，SH データの作成処理に近づくに伴い多くの異なる処理が発生するため別クラスにしている．DepartureUsdlParser クラスが保持するメソッドを表 5.5 に，DestinatioUsdlParser クラスが保持するメソッドを表 5.6 に示す．

表 5.5: **DepartureUsdlParser クラスのメソッド**

`public DepartureUsdlParser(String usdlPath)`	コンストラクタ
`public String getId()`	Id の取得
`public String getIp()`	Ip の取得
`public int getPort()`	Port 番号の取得
`public ArrayList<String> getInherit()`	継承リストの取得
`public String getHandoffDataType()`	データタイプの取得
`public String getHandoffRequestMes()`	リクエストメッセージの取得
`public String getVariableName(String elementName)`	引数の要素名が子要素としてもつ変数名の取得

5.3.5 Usdl Control Server 問合せ部

Usdl Control Server 問合せ部は UCSRequestSender クラスが担う．UCSRequestSender クラスでは HomogeneousServiceJudge クラスと LackDataCalculator クラスからの，Usdl Control Server への問合せ依頼を受けつけ，Usdl Control Server への Tcp[23] を利用した問合せを行う．UCSRequestSender クラスが保持するメソッドを表 5.7 に示す．

5.3.6 サービスハンドオフ先プロトコル情報取得部

SH 先プロトコル情報取得部は HandoffProtocolGetter クラスが担う．HandoffProtocolGetter クラスでは，judgeHomogeneousService メソッドの返り値である SH 先サービスの DestinationUsdlParser インスタンスを利用して，SH に必要となる変数名のリスト，Ip，Port 番号，作成データ構造情報，Type 値を取得する．そして図 4.11 で示したように，作成する SH データの構造情報を XmlArchitecture クラスのインスタンスとして構造化し，データマッピング部に渡す．作成データ構造情報，Type 値は SH データ作成部へ，Ip，Port 番号は SH データ送信部へ渡す．HandoffProtocolGetter クラスが保持するメソッドを表 5.8 に示す．

表 5.6: **DestinationUsdlParser クラスのメソッド**

`public DestinationUsdlParser(String usdlPath)`	コンストラクタ
`public String getId()`	Id の取得
`public String getIp()`	Ip の取得
`public int getPort()`	Port 番号の取得
`public ArrayList<String> getInherit()`	継承リストの取得
`public String getHandoffDataType()`	データタイプの取得
`public XmlArchitecture getXmlArchitectureObj()`	SH データの XML 構造のデータ取得
`public ArrayList<String> getVariableNameList()`	SH データ作成に必要となる変数名リストの取得
`String getVariableName()`	SH データ作成に必要となる変数名の取得 (再帰呼び出し用)

表 5.7: **UCSRequestSender クラスのメソッド**

`public ArrayList<String> sendRequest( String id, String request)`	Usdl Control Server からの情報取得
`void connectUCS(String ip, int port)`	UCS への接続
`void makeReaderAndWriter()`	入出力用ストリームの作成

表 5.8: **HandoffProtocolGetter クラスのメソッド**

`public HandoffProtocolGetter( DestinationUsdlParser destination)`	コンストラクタ
`public String getId()`	Id の取得
`public String getIp()`	Ip の取得
`public int getPort()`	Port 番号の取得
`public ArrayList<String> getInherit()`	継承リストの取得
`public String getHandoffDataType()`	データタイプの取得
`public XmlArchitecture getXmlArchitectureObj()`	SH データの XML 構造のデータ取得
`public ArrayList<String> getVariableNameList()`	SH データ作成に必要となる変数名リストの取得

5.3.7 サービスハンドオフデータ取得&解析部

SH データ取得&解析部は HandoffDataGetterAndParser クラスが担う．HandoffDataGetterAndParser クラスではまず，SH 元サービスの DepartureUsdlParser インスタンスを利用してリクエストメッセージを取得する．次に SH 元データにリクエストメッセージを送信し，SH データ要求を行い SH データを取得する．その後，取得データを HandoffXmlParser クラスまたは HandoffCsvParser クラスのインスタンスとして読み込み，解析を行う．ここでの解析とは取得データと usdl ファイルを照合し，変数名とその値のリストを作成することである．HandoffDataGetterAndParser クラスが保持するメソッドを表 5.9，HandoffXmlParser クラスが保持するメソッドを表 5.10，HandoffCsvParser クラスが保持するメソッドを表 5.11 に示す．

5.3.8 データマッピング部

データマッピング部は DataMapper クラスが担う．DataMapper クラスでは，HandoffProtocolGetter クラスにより作成された SH データで必要となる変数名のリストと，HandoffDataGetterAndParser クラスにより作成された受信 SH データの変数名とそのデータ値のリストから，SH データの作成で必要となる変数名とそのデータ値のリストを再編成し作成する．この際，SH データで必要となる変数名に対応するデータ値が存在しない場合，欠如データ算出部に依頼し欠如したデータを算出することを試みる．データマッピングの詳細を図 5.6 に示す．

表 5.9: **HandoffDataGetterAndParser クラスのメソッド**

public HandoffDataGetterAndParser(DepartureUsdlParser departure)	コンストラクタ
public void getHandoffData(DepartureUsdlParser departure)	SH データ取得
public ArrayList<ArrayList<String[]>> getDataAndElementName()	データ値とその親要素名のリスト作成
public ArrayList<ArrayList<String[]>> getVariableAndElementName()	変数名とその親要素名のリスト作成
public ArrayList<ArrayList<String[]>> mapVariableAndData()	変数名とそのデータ値のリストをマッピングし作成

表 5.10: **HandoffXmlParser のメソッド**

public HandoffDataGetterAndParser(DepartureUsdlParser departure)	コンストラクタ
public ArrayList<ArrayList<String[]>> getDataAndElementName()	データ値とその親要素名のリスト作成
public ArrayList<ArrayList<String[]>> getVariableAndElementName()	変数名とその親要素名のリスト作成

表 5.11: **HandoffCsvParser クラスのメソッド**

public HandoffDataGetterAndParser(DepartureUsdlParser departure)	コンストラクタ
public ArrayList<ArrayList<String[]>> getDataAndElementName()	データ値とその親要素名のリスト作成
public ArrayList<ArrayList<String[]>> getVariableAndElementName()	変数名とその親要素名のリスト作成

```
package umediator;
 ・・・
public class DataMapper {

    public ArrayList<String[]> dataMapping(
                                ArrayList<String> destinationVariableList,
                                ArrayList<String[]> variableAndValueList) {

        //最終的にSHデータ作成に必要となる変数名とデータ値をマッピングしたリスト
        ArrayList<String[]> destinationVariableAndValueList = new ArrayList<String[]>();

        //データマッピングのための走査を行うループ
        for (int i = 0; i < destinationVariableList.size(); i++) {
            //欠如データの有無を判定するフラグ true:無し false:有り
            boolean flag = false;
            label1: for (int j = 0; j < variableAndValueList.size(); j++) {
                if (destinationVariableList.get(i).equals(
                        variableAndValueList.get(j)[0])) {
                    //変数名とデータ値をマッピングし一時的に配列に格納
                    String[] destinationVariableAndValue = {destinationVariableList.get(i),
                                                variableAndValueList.get(j)[1] };
                    //マッピング結果である変数名とデータ値を格納
                    destinationVariableAndValueList.add(destinationVariableAndValue);
                    //一致データの変数名とデータ値を削除
                    variableAndValueList.remove(j);
                    flag = true;
                    break label1;
                }
            }
            //欠如データが有る場合の処理
            if (!flag) {
                LackDataCalculator calculator = new LackDataCalculator();
                //欠如データ算出部の呼び出し，結果をリストに追加
                destinationVariableAndValueList.add(
                                        calculator.calculate(destinationVariableList.get(i));
            }
        }
        return destinationVariableAndValueList;
    }
}
```


図 5.6: DataMapper クラス

5.3.9 欠如データ算出部

欠如データ算出部はLackDataCalculatorクラスが担う．LackDataCalculatorクラスは，DataMapperクラスでのデータマッピングに欠如データが発生した際に呼び出されるクラスである．呼び出されると，欠如データに対応する変数の関係性リスト取得をUCSRequestSenderクラスのsendRequestメソッドを呼び出し取得する．次に，取得した関係性リストのうち，HandoffDataGetterAndParserクラスにより作成された受信SHデータの変数名とそのデータ値のリストで算出可能な関係性を走査する．算出可能な関係性が発見された場合は，欠如データの算出を行いDataMapperクラスに算出結果を返す．LackDataCalculatorクラスが保持するメソッドを表5.12に示す．欠如データの算出を行うcalculateメソッドの概観を図5.7に示す．

表 5.12: **LackDataCalculator クラスのメソッド**

`public LackDataCalculator()`	コンストラクタ
`public String calculate(String variableName)`	欠如データの算出

5.3.10 サービスハンドオフデータ作成部

SHデータ作成部はXmlCreaterクラスとCsvCreaterクラスが担う．SHデータ作成部ではまずXmlCreaterクラスが，DataMapperクラスにより作成された，SHデータの作成で必要となる変数名とそのデータ値のリストをもとにXMLデータを作成する．そして，作成するSHデータのデータタイプがXMLの場合はSHデータ送信部に送信依頼をし，CSVの場合は作成されたXMLからCsvCreaterクラスが再度CSVデータを作成し直してから送信依頼をする．XmlCreaterクラスが保持するメソッドを表5.13に示し，CsvCreaterクラスが保持するメソッドを表5.14に示す．

5.3.11 サービスハンドオフデータ送信部

SHデータ送信部はHandoffDataSenderクラスが担う．HandoffDataSenderクラスでは，まずSH先サービスとのソケットを作成し接続を確立する．次に，SHデータ作成部で作成されたデータを，SH先サービスへプロトコル情報に従い送信する．HandoffDataSenderクラスが保持するメソッドsendHandoffDataメソッドのみである．sendHandoffDataメソッドの詳細を図5.8に示す．

```
public String calculate(String variableName){
    //UCS へ関係性情報問合せ
    UCSRequestSender requestSender = new UCSRequestSender();
    ArrayList<String> list = requestSender.sendRequest(variableName,
                                                    ”relationshipList}request ”);

    //関係性情報の数だけ繰り返し走査する
    label1: for(int i = 0; i < list.size(); i++){
        /*
         * 関係性情報に含まれる変数を抽出し ArrayList relationList に追加
         */
        for(int j = 0; j < relationList.size(); j++){
            //relationList の変数のデータ値が全て揃う関係性情報の有無を判定するフラグ
            boolean flag = false; // true:有り false:無し
            for(int k = 0; k < variableAndValueList.size(); k++){
                /*
                 * relationList の変数のデータ値が全て揃う関係性情報を走査
                 * 発見できたら変数部にデータ値を代入し欠如データを算出し result に代入
                 * flag を true にする
                 */
            }
            //変数のデータ値が全て揃う関係性情報が有った場合
            if(flag){
                break label1;
             }
        }
    }
    return result;
}
```

図 5.7: calculate メソッド

表 5.13: **XmlCreater クラスのメソッド**

public boolean createXml(XmlArchitecture obj, ArrayList<ArrayList<String[]>> list)	SH データである XML ファイルを作成
void createXmlData(XmlArchitecture obj, Element parentElement, ArrayList<String> operatorList_org, ArrayList<String> dataList_org)	XML データの内容作成（再起呼び出し用）
void repeatXml(XmlArchitecture obj, Element parentElement, String num)	リピートコマンドの処理
ArrayList<String> getLimitOperator(String data, ArrayList<String> operatorList)	limit 制御の条件式から&&や—— などの演算子をリストとして取得
ArrayList<String> getLimitData(String data, ArrayList<String> dataList)	limit 制御の条件式から演算子以外の式をリストとして取得

表 5.14: **CsvCreater クラスのメソッド**

public boolean createCsv(String filePath_xml)	SH データである CSV ファイルを作成

```
public void sendHandoffData(String ip, int port, String filePath) {
    try {
        // ソケットを生成
        sok = new Socket(ip, port);
        System.out.println("Connected to server");

        try {
            byte[] data = new byte[512];

            FileInputStream fin = new FileInputStream(filePath);
            out = sok.getOutputStream();
            // ファイルの内容を読み出し，送信する
            System.out.println("Sending file : " + filePath);
            int totalSize = 0;
            int len = 0;
            while ((len = fin.read(data)) != -1) {
                totalSize = totalSize + len;
                System.out.println(new String(data, 0, len));
                out.write(data, 0, len);
            }
            fin.close();
            fin = null;
            System.out.println("size of file : " + totalSize);
            sok.close();
            System.out.println("切断");
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

図 5.8: sendHandoffData メソッド

5.4 Usdl Control Server の実装

本章では Usdl Control Server の実装について述べる．今回は uMediator のテスト環境の一部として Usdl Control Server を実装するため，uMediator からのサービス情報要求に応答する部分のみ実装する．具体的には，設計で述べた uMediator 処理系に属する uMediator 接続待機部と uMediator リクエスト応答部，データベース制御部，Usdl Information データベース部分である．ゆえに，本章では Usdl Control Server の設計で述べた各処理部のうち，今回実装する処理部に対応するクラスについて解説する．

5.4.1 uMediator 接続待機部

uMediator 接続待機部は UCSMain クラスが担う．UCSMain クラスでは，uMediator からの接続を待機し，接続が確立されると UCSReplier インスタンスを作成するクラスである．UCSMain クラスの詳細を図 5.9 に示す．

5.4.2 uMediator リクエスト応答部

uMediator リクエスト応答部は UCSReplier クラスが担う．UCSReplier クラスは UCSMain クラスによりスレッドとして作成されるため Thread クラスを extends する．また，uMediator からのリクエストメッセージを待機し，受信後はリクエストメッセージに応じたデータを DbController クラスを利用し，Usdl Information データベースから取得する．その後，取得データを uMediator へ送信する．UCSReplier クラスの詳細を図 5.10 に示す．

5.4.3 データベース制御部

データベース制御部は DbController クラスが担う．DbController クラスは Usdl Information データベースを制御するクラスであり，UCSReplier クラスからの依頼に応じデータの取得を行う．DbController クラスが保持するメソッドのうち uMediator 処理系で利用するメソッドのみを表 5.15 に示す．

5.4.4 Usdl Information データベース

Usdl Information データベースは id を主キーとして，inherit に継承，relationship に関係性情報を保存する usdl_info テーブルからなる．usdl_info テーブルの構成を図 5.11 に示す．

```
package ucs;

import java.net.ServerSocket;
import java.net.Socket;

public class UCSMain {

    UCSReplier channel[] = new UCSReplier[MAX_CHANNELS];
    ServerSocket serversok; // 接続受け付け用 SereverSocket
    int port = 19210; // ポート番号
    int connectNum = 0; // 接続したクライアント数

    public static void main(String[] args) {
        new UCSMain();
    }

    public UCSMain() {
        waitLoop();
    }

    public void waitLoop() {
        try {
            serversok = new ServerSocket(port); // サーバソケットの準備
            System.out.println("UCSMain 準備完了");
            while (true) {
                Socket sok = serversok.accept(); //接続待ち
                channel[connectNum] = new UCSReplier(sok, this); // 新規チャネル作成
                connectNum++;
            }
        } catch (Exception e) {
            System.out.println("サーバの作成及び接続待ち時の例外");
            serverClose();
        }
    }
     ...
}
```

図 5.9: UCSMain クラス

```
package ucs;
 ...
public class UCSReplier extends Thread {
    ...
   public void run() {
      ArrayList<String> data_get = new ArrayList<String>();

      try {
         in = new ObjectInputStream(sok.getInputStream());
         data_get = (ArrayList<String>) (in.readObject());
      } catch (Exception e) {
         e.printStackTrace();
      }

      // 継承情報の取得要求の場合
      if (data_get.get(1).equals("inheritList_request")) {
         DbController dbController = new DbController();
         // usdl_info から継承情報収集
         inheritList = dbController.getInheritInfo(data_get.get(0));
         try {
            // オブジェクト出力ストリーム
            ObjectOutputStream oos = new ObjectOutputStream(sok
                                          .getOutputStream());
            // 応答データの送信
            oos.writeObject(inheritList);
            sok.close();
         } catch (Exception e) {
            e.printStackTrace();
         }

      // 関係性情報の取得要求の場合
      } else if (data_get.get(1).equals("relationshipList_request")) {
         DbController dbController = new DbController();

         // usdl_info から関係性情報収集
         relationshipList = dbController.getRelationInfo(data_get.get(0));
          ...
      }
   }
    ...
}
```


図 5.10: UCSReplier クラス

表 5.15: **DbController クラスのメソッド**

`public DbController()`	コンストラクタ
`public ArrayList<String> getInheritInfo(String id)`	データベースから継承情報の取得
`public ArrayList<String> getRelationInfo(String id)`	データベースから関係性情の報取得

id	inherit	relationship
char(255)	char(255)	char(255)

図 5.11: **usdl_info テーブル**

5.5 本章のまとめ

本章では 6 章の設計に従い，異種サービス間でのハンドオフ支援機構である uMediator と，連携サーバである Usdl Control Server のプロトタイプ実装について述べた．

第6章

評価

本章では，SH 支援機能およびサービス情報管理機能のプロトタイプとして実装した uMediator，Usdl Control Server を用いた実証実験と uMediator の性能評価について述べる．

6.1 サービスハンドオフ実証実験

本節では，SH 実証実験について述べる．本論文では異なる環境に存在し，異なる開発者により開発された共通のインタフェースを備えないストリーミング動画再生サービス間での SH を想定し実証実験を行った．

本実験では異なる開発者が独自定義した SH プロトコルインタフェースを備えるストリーミング動画再生サービスをそれぞれ異なる環境に設置した．その後，uMediator が実装されているユーザ端末を所持したユーザが環境間を移動した場合に SH が正常に行われるか実験した．実際の SH は環境内に複数存在するサービスの中から SH 先サービスを発見する必要があるため，SH 先サービスが存在する環境にはストリーミング動画再生サービス以外にも複数のサービスが存在するよう，ダミー usdl ファイルを用意した．usdl ファイルを配信するサービス情報配信サーバは実装せず，ユーザ端末に各実験環境の usdl ファイルを保存しておき，SH 管理機能が各実験環境のファイルパスを選択し uMediator に送信することで代替した．またユーザの環境移動を検知するとともに usdl ファイルを収集し，uMediator に usdl ファイルを渡す SH 管理機能は本実験用に簡易実装した．実験環境については次項で詳細を述べる．

6.1.1 実験環境

本実験では，図 6.1 に示すように右の環境を環境 α，左の環境を環境 β と想定し，それぞれ端末は別環境に存在すると想定する．環境 α の端末が SH 元サービス提供端末で，環境 β の端末が SH 先サービス提供端末である．

また，実験用計算機として uMediator が処理を行うユーザ端末，Usdl Control Server，SH 元サービス提供端末，SH 先サービス提供端末を用意した．それぞれ，ユーザ端末，サーバ端末，端末 A，端末 B とする．表 6.1 に各計算機の詳細を示す．

表 6.1: **実験用端末の性能表**

	ユーザ端末	サーバ端末	端末 A	端末 B
PC	MacBook Pro	Mac Pro	iMac	iMac
CPU	2.8GHz Core2Duo	2x2.8GHz Quad-Core	2.66GHz Core2Duo	2.66GHz Core2Duo
メモリ	4GB	16GB	4GB	4GB
OS	Mac OS X 10.5.8	Mac OS X 10.5.8	Mac OS X 10.5.8	Mac OS X 10.5.8

また，環境 β に用意したダミーを含む usdl ファイルの概要，usdl ファイルに応じた Usdl Information データベース概要，SH 管理機能の簡易実装の概要，SH 元サービスの概要，SH 先サービスの概要について以下で述べる．

図 6.1: **実験環境**

環境βの usdl ファイル概要

環境βでは SH 先サービスの usdl ファイル以外に環境内に複数のサービスが存在するよう見せるためにダミーの usdl ファイルを用意した．表 6.2 に環境βに用意した 4 つの usdl ファイルの概要を示す．

Usdl Information データベース概要

本実験用に表 6.2 で示した usdl ファイルに応じ，Usdl Information データベースの usdl_info テーブルにデータを用意した．図 6.2 に usdl_info テーブルのスクリーンキャプチャを示す．

SH 管理機能の簡易実装概要

SH 管理機能は実行されると環境βに存在する usdl ファイルの保存先ファイルパスを uMediator に送信するよう，ユーザ端末上に簡易実装した．本来は SH 管理機能が

表 6.2: **環境βの usdl ファイル概要**

サービス	SH 元サービスとの関係性
ストリーミング動画再生サービス	ホップ数 1（SH 先サービス）
動画再生サービス	ホップ数 2
音楽再生サービス	ホップ数 3
ナビゲーションサービス	関係性無し

id	inherit	relationship
player		
movie_player	player	
music_player	player	
streaming_movie_player	movie_player	
streaming_movie_playerA	streaming_movie_player	
streaming_movie_playerB	streaming_movie_player	
streaming_music_player	music_player	
streaming_music_playerA	streaming_music_player	
navigation_service	navigation	
navigation_serviceA	navigation_service	
url_streaming_movie	url	
start_play_min	min	[start_play_sec]/60,[start_play_msec]/1000/60
start_play_sec	sec	[start_play_min]*60,[start_play_msec]/1000
start_play_msec	msec	[start_play_sec]/60,[start_play_min]/60/60

図 6.2: **実験用 usdl_info テーブル**

ユーザ端末のネットワークの切り替り等を検知することでユーザの環境移動を判定し SH の処理を開始すべきであるが，本実験では環境を移動した際にユーザが手動で SH 管理機能を実行することで SH の処理を開始する．

SH 元サービスの概要

SH 元サービスは指定 url のストリーミング動画再生機能，再生中動画の停止機能，サービスの状態情報である SH データの送受信機能，SH データの解析機能，解析結果に応じたストリーミング動画の再生機能を有する，ストリーミング動画再生サービスである．本サービスでは，動画の url を指定しプログラムを実行するか，何も指定せずに実行し SH 待機状態にするかの二通りのサービス開始方法を選択できる．本実験では，SH 元サービスである本サービスは動画の url を指定しサービスを開始する．また SH のリクエストは常時，待機状態にあり uMediator より SH リクエストメッセージを受信するとデータタイプが CSV の SH データを返信し再生中の動画を停止する．本サービスは C 言語 [5] で実装され，送受信可能なデータタイプは CSV 形式のデータである．図 6.3 に本サービスの usdl 記述を示す．

```
<entity>
  <properties>
    <id>streaming_movie_playerA</id>
    <name lang="ja">ストリーミング動画再生サービス A</name>
    <inherit>streaming_movie_player</inherit>
      ...
  </properties>
    ...
  <handoff>
    <handoff_send_protocol data_type="csv" requestMes="hello_serviceA">
      <csv_data>
        [url_streaming_movie],
        [start_play_msec]
      </csv_data>
    </handoff_send_protocol>
    <handoff_receive_protocol data_type="csv">
      <usdl:csv>
        <usdl:csv_data>
          [url_streaming_movie]
        </usdl:csv_data>
        <usdl:csv_data>
          [start_play_msec]
        </usdl:csv_data>
      </usdl:csv>
    </handoff_receive_protocol>
  </handoff>
</entity>
```

図 6.3: SH 元サービスの usdl 記述

SH 先サービスの概要

SH 先サービスはサービスの状態情報である SH データの送受信機能，SH データの解析機能，解析結果に応じたストリーミング動画の再生機能，再生中動画の停止機能を有する，ストリーミング動画再生サービスである．SH 元サービスとは異なる開発者によって，java 言語で実装された異種サービスである．本サービスは SH のリクエスト待機状態として実行され，データタイプが XML の SH データを受信することでストリーミング動画の再生が行われる．SH データからはストリーミング動画の url と再生開始箇所を取得する．図 6.4 に本サービスの usdl 記述を示す．

```
<entity>
  <properties>
    <id>streaming_movie_playerB</id>
    <name lang="ja">ストリーミング動画再生サービス B</name>
    <inherit>streaming_movie_player</inherit>
       ...
  </properties>
     ...
  <handoff>
    <handoff_send_protocol data_type="xml" requestMes="hello_serviceB">
      <StreamingMovieServiceB>
        <start_point>[start_play_sec]</start_point>
        <url>[url_streaming_movie]</url>
      </StreamingMovieServiceB>
    </handoff_send_protocol>
    <handoff_receive_protocol data_type="xml">
      <StreamingMovieServiceB>
        <start_point>[start_play_sec]</start_point>
        <url>[url_streaming_movie]</url>
      </StreamingMovieServiceB>
    </handoff_receive_protocol>
  </handoff>
</entity>
```

図 6.4: SH 先サービスの usdl 記述

6.1.2 実験結果

ユーザが環境 α から環境 β へ移動した事を通知する SH 管理機能をユーザが実行した際に，端末 A で再生中のストリーミング動画が端末 B へ再生途中から切り替わるか実験した結果，切り替わる事を確認した．つまり，uMediator により環境 β に複数存在するサービスの中から異種サービスの発見が正常に行われ，SH データの変換が正常に行われることで SH が成功したことを確認した．図 6.5 に実験の様子を示す．

図 6.5: **実験風景**

また，実験の際に SH 元サービスが uMediator に送信した SH データと uMediator が SH 先サービスに送信した SH データを図 6.6 に示す．

SH 元サービス→ uMediator（CSV）：

```
http://www.youtube.com/watch?v=XXXXX,1831
```

uMediator → SH 先サービス（XML）：

```
<StreamingMovieServiceB>
  <start_point>18.31</start_point>
  <url>http://www.youtube.com/watch?v=XXXXX</url>
</StreamingMovieServiceB>
```

図 6.6: 本実験で送受信された SH データ

図 6.6 より分かるように，SH 元サービス→ uMediator の際には 1831 とミリ秒単位として表現されていたデータが，uMediator → SH 先サービスの段階では 1000 分の 1 倍され秒単位に換算されている．これは uMediator が usdl_info テーブルの relationship 情報を元に正常にデータ変換を行ったことを示している．

本実証実験により，uMediator は複数存在するサービスの中から異種サービスを発見し，各サービスの SH プロトコルに応じた SH データの変換を行える事を実証した．また，異種サービス間での SH に uMediator が有効であることを実証した．

6.2 uMediator の基本性能の評価

本節では，uMediator の基本性能の評価について述べる．本論文では特に異種サービスの発見と SH データの変換に注目し評価を行った．

異種サービスの発見は，uMediator の異種サービス判定部が USDL 受信部から SH 元サービスと SH 先サービスの usdl ファイルを受信し，異種サービスを判定するまでの処理を指す．SH データの変換は，異種サービスの判定後の処理から SH データ作成部で SH データを作成し終えるまでの処理を指す．図 6.7 に評価対象部の概要を示す．

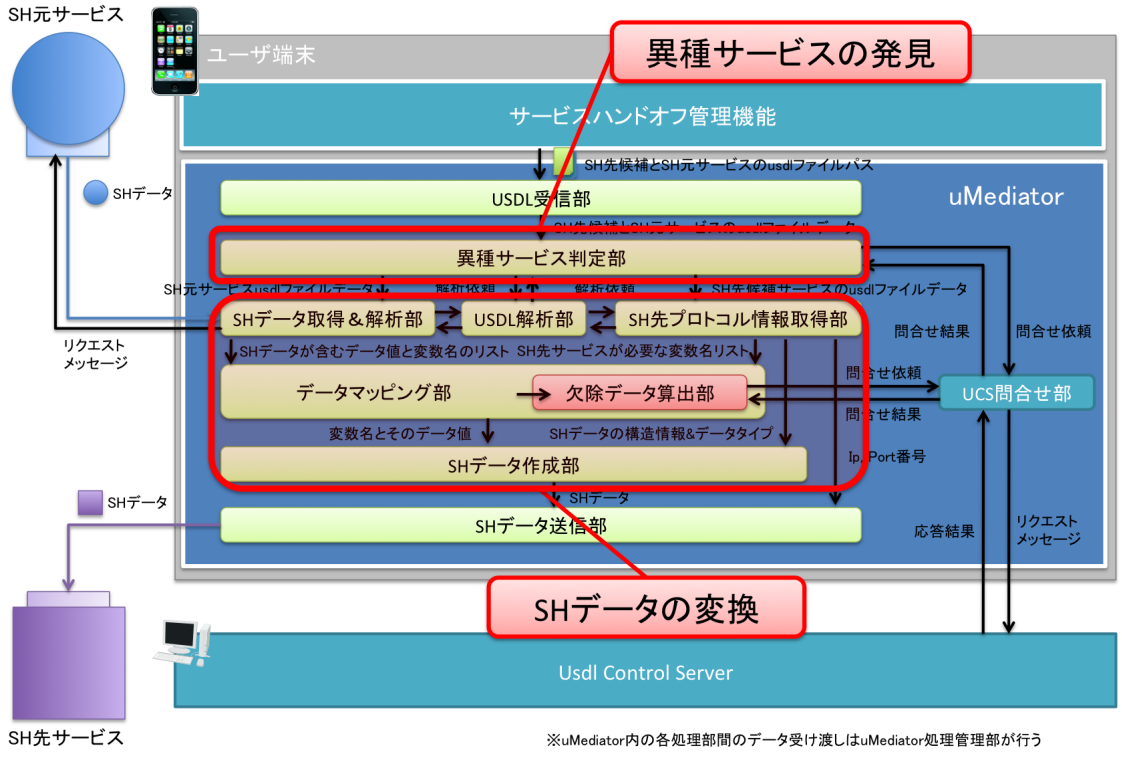

図 6.7: **uMediator の評価対象部の概要**

本論文では，環境内に存在する SH 先候補サービスが増加するに伴い異種サービスの発見にどのような影響が出るのか，また SH データのデータ数が増加するに伴い SH データの変換にどのような影響が出るのか，それぞれ処理時間を計測し評価した．SH データの変換は，XML から XML，XML から CSV，CSV から CSV，CSV から XML へデータ変換する 4 パターンが存在するのでそれぞれのパターンで処理時間を計測し，

データタイプの違いによる影響について評価した．本評価では uMediator の処理性能を評価するため，uMediator が他の端末と通信するのに要した時間は処理時間から省いている．

6.2.1 計測環境

計測用計算機として，uMediator が処理を行うユーザ端末，Usdl Control Server，SH 元サービス提供端末，SH 先サービス提供端末を用意した．それぞれ，ユーザ端末，サーバ端末，端末 A，端末 B とする．表 6.3 に各計算機の詳細を示す．

表 6.3: **評価用端末の性能表**

	ユーザ端末	サーバ端末	端末 A	端末 B
PC	MacBook Pro	Mac Pro	iMac	iMac
CPU	2.8GHz Core2Duo	2x2.8GHz Quad-Core	2.66GHz Core2Duo	2.66GHz Core2Duo
メモリ	4GB	16GB	4GB	4GB
OS	Mac OS X 10.5.8	Mac OS X 10.5.8	Mac OS X 10.5.8	Mac OS X 10.5.8

6.2.2 計測結果

以下に，異種サービスの発見と SH データの変換に要した処理時間の計測結果を示す．

異種サービスの発見

図 6.8 のグラフに異種サービスの発見に要した平均処理時間を示す．各処理時間はそれぞれ 500 回の処理の平均である．グラフより分かるように，異種サービスの発見に要する処理時間は usdl ファイル数（環境内に存在する SH 先候補サービス数）に比例して増加した．これは，SH 元サービスの継承と各 usdl ファイルの継承リストを比較する走査を全ての usdl ファイルと行っているためである．本計測ではそれぞれ 3 つの継承リストを保持する usdl ファイルをテストデータとして使用したため，usdl ファイルが一つ増加するたびに 9 回の比較処理が発生した．しかし，実際に環境内に存在するサービスの継承数はサービスごとに異なり，異種サービスの発見に要する処理時間は usdl ファイル数に比例し増加するとは限らない．だが，SH 元サービスの継承と各 usdl ファイルの継承リストの比較回数に比例して処理時間が増加することが本計測より分かった．

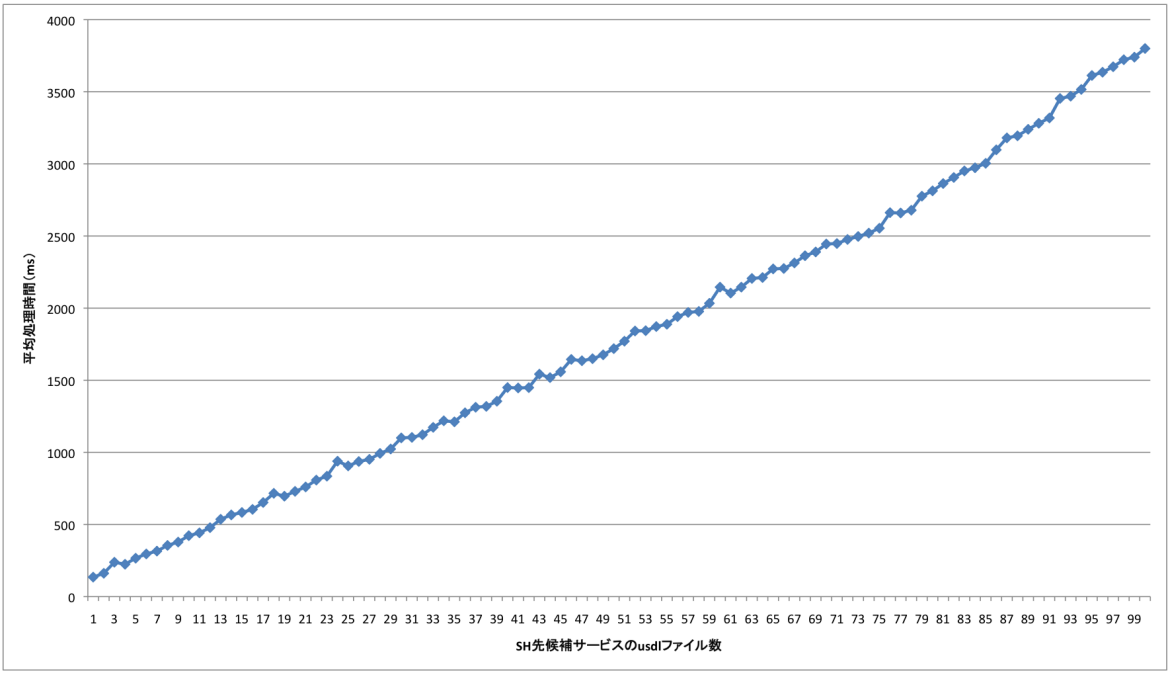

図 6.8: **異種サービスの発見に要した平均処理時間のグラフ**

SH データの変換

図6.9のグラフにXMLからXML，XMLからCSV，CSVからCSV，CSVからXMLへデータ変換した場合に要した平均処理時間を示す．各処理時間はそれぞれ100回の処理の平均である．変換元データのデータタイプごとに結果を見ると，XMLの場合の処理の方がより急な増加曲線を示している．これは，SHデータ取得&解析部での処理内容に大きく影響を受けている．SHデータ取得&解析部では，データタイプがCSVのデータを受信した場合には，usdlファイルの変数名のリストの順にコンマ区切りでデータ値を受信でき，本研究の実装では変数名とデータのマッピングはO(n)の処理で行っている．しかし，データタイプがXMLのデータを受信した場合は，受信データよりデータ値と親要素名のリスト，usdlファイルより変数名と親要素名のリストを作成し，それら二つのリストをマッチングさせることでusdlファイルの変数名とデータ値のリストを作成する必要があり，本研究の実装では$O(n^2)$の処理で実装している．ゆえに，データタイプがXMLのデータを受信した場合の方が処理に時間を要したといえる．また計測回数の問題で平均したデータにばらつきは有るものの，概ね変換後のデータタイプがCSVの処理の方がにXMLの場合より処理に時間を要している．これは，CSVへのデータ変換は一度XMLへ変換した後に再度XMLを解析しCSVに変換しているためである（詳細は4.4.10参照）．

以上より，SHデータの変換処理時間は変換元データのデータタイプに大きく影響されることを確認した．

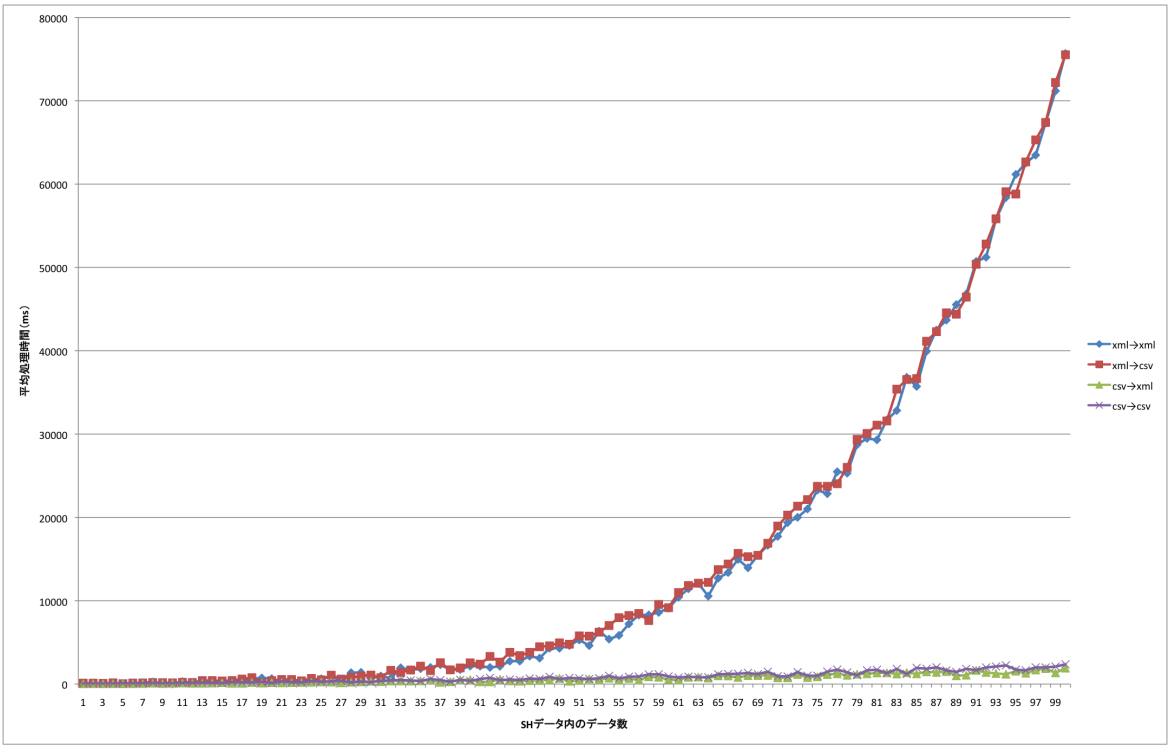

図 6.9: **SH データの変換に要した平均処理時間のグラフ**

6.3 本章のまとめ

本章では，uMediator と Usdl Control Server を用い，異なるユーザにより開発され共通のインタフェースを備えないストリーミング動画再生サービス間での SH の実証実験を行うとともに，uMediator の基本性能の評価を行った．

実証実験では，uMediator は複数存在するサービスの中から異種サービスを発見し，各サービスの SH プロトコルに応じた SH データの変換を行える事を実証した．また，異種サービス間での SH に uMediator が有効であることを実証した．

uMediator の基本性能評価では，SH 元サービスから受信する SH データのデータタイプが uMediator の処理時間に大きな影響を与える事を確認した．具体的には本研究の実装では $O(n^2)$ の処理を有しているため，SH 元サービスから受信する SH データのデータタイプが XML の場合はデータ数の増加に伴い膨大な処理時間を要するという問題を確認した．

第7章

結論

本章では本論文のまとめを述べ，最後に今後の課題について述べる．

7.1 まとめ

本論文では，はじめに現在の情報環境において新たに発生すると考えられる移動しながら環境に応じよりリッチなユーザエクスペリエンスを提供するサービスに切替え利用したいという要求と，実現の際に発生する問題について述べた．具体的には異なる環境に複数存在するサービスの中から切替え先となる異種サービスの発見問題，環境の管理者もサービスの開発者も異なり共通のインタフェースを備えない異種サービス間での互換性確保問題の二つの問題があった．互換性確保問題は各サービスが独自に作成したデータを送受信しあえる互換性と，お互いに独自定義したデータを解釈できる互換性を確保する必要があった．

そこで本研究では，これらの問題を解決する研究領域としてサービスハンドオフを定義し，サービスハンドオフを実現する環境としてサービスハンドオフフレームワークを提案した．本フレームワークの提案に向け，サービスを分類し対象サービス群を明らかにした後に，機能要件をあげ，アプローチ方法の検討を行った．具体的には，サービス切替え後も通信データを継続的に変換し続ける必要がないサービスを対象とし，モバイル端末上の支援機構による状態データの変換移送を行うことで本フレームワークを実現することにした．

本研究では，サービスハンドオフフレームワークで必要とされる機能のうちコアとなるサービスハンドオフ支援機能と，本機能の処理に大きく関わるサービス情報管理機能に焦点を当て研究を行った．サービスハンドオフ支援機能はユーザ端末上のミドルウェア：uMediator として，サービス情報管理機能は任意の範囲に一つ設置が求められるサーバ：Usdl Control Server として設計，実装を行った．また，サービス情報を記述するために USDL の拡張記述を設計・定義した．以上により，問題点として挙げた切替え先サービスの発見と，異種サービス間でのデータ送受信，解釈の互換性確保の問題を解決した．

本論文では uMediator と Usdl Control Server を利用し，異なる環境に存在し異なる開発者が開発した共通のインタフェースを備えないストリーミング動画再生サービス間でのサービスハンドオフの実証実験を行い，uMediator の有効性を実証した．しかし uMediator の性能評価では，実装に $O(n^2)$ の処理を有していたために SH 元サービスの SH データタイプが XML の場合は，データ変換に膨大な処理時間を要するという問題を確認した．

7.2 今後の課題

本節では，今後の課題について述べる．以下に課題を列挙する．

uMediator の性能向上

本論文では uMediator の性能評価で SH 元サービスの SH データタイプが XML の場合は，データ変換に膨大な処理時間を要するという問題を確認した．SH はユーザの移動に伴い発生すると考えられるため，短時間でのサービス切換えが求

められるが，現在の実装では SH データ内のデータ数が増加するにつれ実用性が失われることになる．ゆえに，今後は $O(n^2)$ の処理を改善しより短時間でデータ変換を行うよう改良し性能を向上させる．

SH フレームワークの他の処理機構の研究

本論文では，SH フレームワークのコアとなる SH 支援機能，SH 支援機能と通信し処理に影響を与えるサービス情報管理機能に焦点を絞り研究を行った．今後は，SH 管理機能，サービス情報配信機能へ焦点を当て研究を行う必要がある．

特に SH 管理機能は，SH 実用化のためにはどのような設定が必要か，よりユースケースに踏み込み研究を行う必要がある．また SH のトリガであるユーザの情報環境移動は，何を基準にどのように判断すべきなのかなど多くの研究課題が存在する．

非対応サービスへの対応

本研究では SH の汎用面と開発コストの面から，モバイル端末上の支援機構による状態データの変換移送を行うアプローチを採った．しかし，本アプローチ方法は SH 後もデータの変換が必要となるサービスに関しては非対応である．ゆえに，サービス固有の変換サーバを構築する方法についても検討し，非対応のサービスへも対応する必要がある．

登録済みサービス情報の閲覧を可能にする

本研究では他サービスの usdl を継承し，その継承を調べることでサービス同士の関係を求め異種サービスを判定している．しかし，他サービスの開発者が記述した usdl を参照する仕組みは存在しない．ゆえに Usdl Control Server の情報をサービス開発者が閲覧し，開発サービス情報を継承を利用し記述可能にする必要がある．

謝辞

本研究を進めるにあたって貴重な御指導を賜りました，慶應義塾大学環境情報学部 徳田 英幸教授に深く感謝致します．また，重要な御助言を頂きました， 慶應義塾大学環境情報学部 武藤 佳恭教授，並びに，慶応義塾大学環境情報学部 清木 康教授に深いお礼を申し上げます．

慶應義塾大学徳田・高汐・中澤研究室の皆様に多くの御助言を頂きました．特に活動の中心となった，move! 研究グループの榊原 寛氏，米澤 拓郎氏，本多 倫夫氏には大変多くの助言を頂きました．海よりも深く感謝致します．また，中井 彦一郎氏，野沢 高弘氏，荒木 貴好氏，米川 賢治氏，上田 真央氏中原 洋志氏，堀川 哲郎氏，西 和也氏，丹羽 亮太氏の心遣いに感謝致します．

折に触れ，高汐 一紀准教授，中澤 仁講師より重要かつ的確な御助言を頂きました．深く感謝します．また，同じ立場で研究の日々を共に過ごした生天目 直哉氏，伊藤 友隆氏，河田 恭兵氏，Thi-huong-giang Vu 氏，橋爪 克弥氏，今枝 卓也氏に感謝します．

平成 22 年 1 月 12 日
中川 直樹

参考文献

[1] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Trans. on Software Engineering*, May 1998.

[2] Amazon Web API. http://www.amazon.co.jp/gp/feature.html?docid=451209.

[3] Andrew J. Viterbi, Audrey M. Viterbi, Klein S. Gilhousen, and Ephraim Zehavi. Soft handoff extends cdma cell coverage and increases reverse link capacity. *Mobile Communications Advanced Systems and Components*, 1994.

[4] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. Easyliving: Technologies for intelligent environments. *HUC2000*, pp. 12–29, Sep 2000.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, 1988.

[6] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, and Blair MacIntyre. The Aware Home: A Living Laboratory for Ubiquitous Computing Research . CoBuild ' 99, pp. 191–198, 1999.

[7] EPCglobal. http://www.epcglobalinc.org/home.

[8] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software: Practice and Experience*, Vol. 21, No. 8, pp. 757–785, 1991.

[9] F.T.H. den Hartog, M.A. Blom, C.R. Lageweg, M.E. Peeters, J.R. Schmidt, R. van der Veer, A. de Vries, and M.R. van der Werff. Joint mobility tracking and hard handoff in cellular networks via sequential monte carlo filtering. *IEEE INFOCOM*, 2002.

[10] F.T.H. den Hartog, M.A. Blom, C.R. Lageweg, M.E. Peeters, J.R. Schmidt, R. van der Veer, A. de Vries, and M.R. van der Werff. First experiences with personal networks as an enabling platform for service providers. *MobiQuitous*, 2007.

[11] Google Maps. http://maps.google.com/.

[12] Google Maps API. http://code.google.com/intl/ja/apis/maps/.

[13] Gregory D. Abowd, Siewiorek D., Smailagic A., and Steenkiste P. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive computing*, Vol. 4, pp. 22–31, 2002.

[14] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu, editors. *Live migration of virtual machine based on full system trace and replay*, 2009. High Performance Distributed Computing Proceedings of the 18th ACM international symposium on High performance distributed computing.

[15] Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, and Anupam Joshi, editors. *Intelligent Agents Meet Semantic Web in a Smart Meeting Room*, Vol. 2, 2004. International Conference on Autonomous Agents archive Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems.

[16] java.sun.com. http://java.sun.com/.

[17] JINI NETWORK TECHNOLOGY. http://jp.sun.com/jini/.

[18] Masatoshi Arikawa, Shin'ichi Konomi, and Keisuke Ohnishi. Navitime: Supporting Pedestrian Navigation in the Real World. *IEEE Pervasive computing*, Vol. 6, No. 3, pp. 21–29, 2007.

[19] MIT Project Oxygen. http://oxygen.csail.mit.edu/.

[20] MySQL 5.1 Reference Manual. http://dev.mysql.com/doc/refman/5.1/en/index.html.

[21] MySQL Connector/J Documentation. http://dev.mysql.com/doc/refman/5.1/en/connector-j.html.

[22] Networking Bonjour. http://developer.apple.com/networking/bonjour/.

[23] RFC: 793. http://tools.ietf.org/html/rfc793.

[24] Seok Joo Koh, Moon Jeong Chang, and Meejeong Lee. Data link level support for handoff in wireless atm network. *IEEE COMMUNICATIONS LETTERS*, 1997.

[25] skype. http://www.skype.com/intl/en/welcomeback/.

[26] Sueng-Yong Park, Vaduvur Bharghavan, and Sung-Mo Kang. msctp for soft handover in transport layer. *IEEE COMMUNICATIONS LETTERS*, Vol. 8, No. 3, pp. 189–191, MARCH 2004.

[27] Takahiro Sakamoto, Tatsurou Sekiguchi, and Akinori Yonezawa. Bytecode transformation for portable thread migration in java. *In Proceedings of the Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA/MA)*, pp. 16–28, 2000.

[28] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, Vol. 1, pp. 1–2, 1998.

[29] UPnP Forum. http://www.upnp.org/.

[30] Vladimir Kulyukin, Chaitanya Gharpure, John Nicholson, and Sachin Pavithran, editors. *RFID in Robot-Assisted Indoor Navigation for the Visually Impaired*, 2004. Proceedings of the IEEE International Conference.

[31] Mark Weiser. The computer for the 21st century. *Scientific American*, Vol. 265, pp. 94–, Sep 1991.

[32] X.Org. http://www.x.org/.

[33] 山崎俊作, 伊藤友隆, 河田恭兵, 生天目直哉, 小河原栄一, 高橋元, 中澤仁, 徳田英幸. あしナビ：主張するアンビエントナビゲーションシステムの試作. 情報処理学会シンポジウム論文集, Vol. 2006, No. 4, pp. 95–96, Nov 2008.

[34] 上野まちナビ実験. http://www.kensetsu.metro.tokyo.jp/machinavi/index.html.

付 録A

Universal Service Description Language仕様(案)

Universal Service Description Language仕様(案)

ATR, 慶應義塾大学, NTT

平成22年2月15日

履歴

2009.2.23

- 初版

2009.3.25

- access の allow に weight 属性を追加
- 嗜好情報 (preference) を追加
- entity の connectivity が物理的な接続情報に対応
- provide と consume に level 属性を追加し、主効果と副次効果を区別可能とした。
- 状態のログを state の history に外部項目の URL として記述可能とした。
- input と output の mime-type を data タグに変更し、固定値を指定可能とした。
- preference の map タグに、エンティティの発見方法を指定可能とした。

2009.5.20

- current の出現数を 0 以上に変更 (複数の状態を書くため)

2009.5.25

- date の値フォーマットを UNIX 時間に変更した。

目 次

A.1 はじめに

A.1.1 USDL を使う意義

ユビキタス空間において物と物、人と物の関係を機械的に処理できる形式で記述しておくことによって、様々なアプリケーションを創出可能となる。また本プロジェクトにおいては、そのような記述形式を共有することにより、各機関の研究成果の統合が容易となる。

A.1.2 思想

物と物との関係においては、ユニバーサルなインタオペラビリティが重要となる。現在のユビキタス空間においては、物 (情報家電機器やセンサノード) が異なるプロトコルを用いているため、それら相互のインタオペラビリティを実現するためにはミドルウエア層の役割が重要となる。本記述言語は、異なるプロトコルを用いる物に対して共通の抽象化を適用するとき、結果として生成される抽象概念を記述するものである。　人と物との関係においては、セマンティックなデジタル情報の記述が重要となる。ある物の利用を特定の人に制限したり、ある物の動作結果が人にとってもつ意味を明らかにしたり、あるいは複数の物の動作が人にとって競合するときそれを解決するポリシーを提供したりすることが、本言語が記述対象とする事項である。　従って本言語は、それ単体では大した意味を持たず、それを解釈および運用することのできる何らかのミドルウエアと組み合わさって初めて意味を持つものである。

A.1.3 記述対象事項の整理

概要

本言語では、型を用いて人、物、空間を記述する。同時に本言語では、型が持つセマンティクスを、明示的に記述する。型は継承を許す。これらにより、型を用いた人、物、空間の検索 (service discovery)、意味を用いたそれらの検索 (semantic service discovery)、あるいは複数の物を複合して粒度の大きな物 (複合サービスなど) を定義した際にその物に対する型検査等が可能となる。　型は抽象的な概念を記述するのに

対し、実空間ではそれを具現化する複数の実体が存在する。従って、型の記述とは別に実体の記述を行う。実体の記述ではまず、その実体が適合する型を指定する。指定できる型は一つとする。次に、その実体に特有の情報を記述する。たとえば、その実体が存在する通信プラットフォーム上で、その実体にアクセスする際のアドレスなどはこの情報に該当する。

型空間

本言語では、人、物、空間のそれぞれに対して別の型を付与する。

–**人** human

–**物** object

–**空間** field

それぞれの型には、そこから派生する型が存在する。たとえば human には user と manager 等が存在し得る。object には...tv、dvdplayer などが存在する。(だんだん増やす)

実体

実空間では、その空間に存在する実体を発見し、いずれかの型にマップする主体 (何らかのソフトウエア) の存在を想定する。例えば Bluetooth デジタルフォトフレームが存在するときには、Bluetooth ネットワークから当該デバイスを発見し、本記述言語で定義するいずれかの型にマップするソフトウエアが存在する。そのようなソフトウエアをマッパー (mapper) と呼ぶ。また、この場合の Bluetooth のような通信規格を、通信プラットフォーム (communication platform) と呼ぶ。マッパーは、通信プラットフォームに固有の抽象化手法から、本記述言語の型空間が規定する抽象化手法への変換を行う。また、本記述言語を解釈および運用するミドルウエアは、それが利用する通信プロトコルやデータ形式と、当該通信プラットフォームに固有の通信プロトコルやデータ形式との変換を行う。

型と実体を区別する意味

型を、実体に依存しない形式で定義することにより次の利点がある。

- サービス同士の連携を 2 つのレベルで行える。まず、連携対象のサービスを実体により指定可能である。これにより、特定の実体同士を連携させる際に特有の制約などを考慮可能となる。ただし、実体により指定された連携は、その実体が存在する空間に依存するため、ポータビリティはない。次に、連携対象のサービス

を型により指定可能である。これにより、連携を抽象的に定義し、実際に連携対象とする実体を、その連携を起動する際に確定させることが可能となる。これにより、連携定義のポータビリティが増す一方、特定の実体に特化した記述は行えなくなる。

A.2 型の記述 type

A.2.1 概要

型は、type タグにより記述する。

A.2.2 メタデータ properties

概要

メタデータは、properties タグにより記述する。

型識別子 id

型の識別子を、id タグにより記述する。識別子は任意の規模で一意である必要がある。ただし識別子の一意性は、本記述言語自体では担保しない。別途、標準化機関等により管理されることを想定する。型識別子は、以下のような英数字および.(ドット)の組み合わせとする。

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
<id>jp.ac.keio.sfc.ht.TV</id>
```

型名称 name

型の名称を、name タグにより記述する。型名称は、利用者に提示される情報であり、従って自然言語で記述する。また name タグは lang 属性に異なる値を指定する限りにおいて、複数個使用してよい。

–**属性** lang

–**出現数** 1 つ以上

例：

```
<name lang="en">TV</name>
<name lang="ja">テレビ</name>
```

継承する型 inherit

型が継承する別の型の型識別子を inherit タグにより記述する。

–**属性** なし

–**出現数** 0 か 1

例：

```
<inherit>jp.ac.keio.sfc.ht.TV</inherit>
```

型の定義者 provider

型を定義した者の情報を provider タグにより自然言語で記述する。型の定義に複数の者が参加している場合、本タグを複数個使用してかまわない。

–**属性** なし

–**出現数** 0 以上

例：

```
<provider>Jin Nakazawa, Keio University</provider>
<provider>Takahiro Hada, NTT Corp.</provider>
```

型のバージョン version

型のバージョンを version タグにより記述する。値の内容は M.N.O とし、MNO のそれぞれには一桁以上の数字を記載する。各部の記載には以下のルールを設ける。
M：　N：　O：

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
<version>1.0.0</version>
```

型の定義日 date

型が定義された日を date タグにより記述する。値の内容は UNIX 時間とする。

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
<date>1387682734(UNIX 時間)</date>
```

解説

その型の全体解説を description タグに記述する。

–**属性** lang

–**出現数** 1 つ以上

例：

```
<description lang="en">hoge</description>
<description lang="ja">ほげ</description>
```

A.2.3 外部接続 interface

概要

その型のエンティティのインタフェースを interface タグに記載する。同タグには operation タグによりその型が持つオペレーションを列挙する。ここでオペレーションとは、そのエンティティが持つ物理的な入出力やネットワークに対する入出力を意味する。たとえばテレビが持つアンテナ入力は、物理的な入出力の一つである。以下に、operation タグに記載可能な事項を列挙する。同タグには、name 属性と override 属性のいずれかと、sequence 属性を指定しなければならない。name 属性には、そのオペレーションの名前を指定する。この属性に指定された名前と同一の名前のオペレーションが親の型により宣言されていた場合には、エラーとなる。override 属性には、親の方に宣言されたオペレーションを上書きするとき、上書き対象のオペレーション名を指定する。sequence 属性によりネットワークとの入出力の順序を指定する。name 属性の値は、その型の中で一意でなければならない。

–**属性** name or override, sequence(optional)

–**属性値 (name,override)** 文字列

–**属性値 (sequence)** out, in, in-out, out-in　のいずれか

解説 description

そのオペレーションの自然言語による解説を description タグに記載する。lang 属性を指定して複数の descripton タグを指定することもできる。

–**属性** lang

–**出現数** 1 つ以上

例：

```
<description lang="en">turns the switch on</name>
<description lang="ja">電源が入って云々</name>
```

ネットワークからの入力 input

そのオペレーションの起動に必要なネットワークからの入力を input タグに記述する。ここでいうネットワークは、特定の通信プラットフォームを想定しない。複数の値を必要とする場合は、input タグを複数個指定する。同タグでは、子要素である data タグを用いて入力データの型 (type 属性に mime を指定) あるいは値そのもの (type 属性に fixed を指定) を指定する。また同データの意味を description タグに記述する。型定義の data タグには、MIME タイプの前半部分 (video, audio 等) のみを指定し、後半部分は.*としておくことが望ましい。後述するエンティティ定義の data タグには、完全な MIME タイプや値そのものを指定してよい。

–**属性** なし

–**出現数** 0 以上

–**子要素** data, description

例 1：

```
<input>
  <data type="mime">video/.*</data>
  <description lang="ja">ビデオ入力</description>
</input>
```

例 2：

```
<input>
  <data type="fixed">0x334455</data>
  <description lang="ja">コマンド</description>
</input>
```

ネットワークへの出力 output

そのオペレーションの結果ネットワークへ出力されるデータの MIME タイプを output タグに記述する。ここでいうネットワークは、特定の通信プラットフォームを想定しない。同タグでは、子要素である data タグを用いて入力データの型 (type 属性に mime を指定) あるいは値そのもの (type 属性に fixed を指定) を指定する。また同データの意味を description タグに記述する。mime-type タグには、MIME タイプの前半部分 (video, audio 等) のみを指定し、後半部分は.* としておくことが望ましい。

–**属性** なし

–**出現数** 0 か 1

–**子要素** data, description

例：

```
<output>
  <data type="mime">video/.*</data>
  <description lang="ja">ビデオ出力</description>
</output>
```

物理的な入力 consume

そのオペレーションの実行に必要な物理的な入力を consume タグに記述する。ここでいう物理的な入力とは、たとえばガスストーブが燃焼する時に消費する酸素やガス、テレビが動作するときに消費する電力などである。記述形式は別途定義する。属性として level を指定して、その物理的入力がそのクラスの主物理効果であるかどうかを指定できる。level 属性に指定できる値は primary のみである。主効果でない場合は level 属性を指定しない。

–**属性** level

–**出現数** 0 以上

例：

```
<consume level="primary">o2</consume>
```

物理的な出力 provide

そのオペレーションの実行結果として空間に生成される出力を¡provide¿タグに記述する。ここでいう物理的な入力とは、たとえばガスストーブが燃焼した時に生成する二酸化炭素や、テレビが動作したときに生成される光などである。記述形式は別途

定義する。属性として level を指定して、その物理的入力がそのクラスの主物理効果であるかどうかを指定できる。level 属性に指定できる値は primary のみである。主効果でない場合は level 属性を指定しない。

–**属性** なし

–**出現数** 0 以上

例：

```
<provide level="primary">co2</provide>
```

A.3 実体の記述 entity

A.3.1 概要

物の実体は、いずれか一つの型の実装として記述する。また作用範囲やベンダー情報などその物独自の静的情報、および位置情報その他の動的情報を記述する。記述には、entity タグを用いる。

A.3.2 メタデータ properties

概要

メタデータは、properties タグにより記述する。静的な情報である。

エンティティ識別子 id

各エンティティの一意な識別子を id タグに URI として記述する。URI の構築方法やその一意性確保は、本言語を解釈、運用するミドルウエアが担保すること。

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
bonjour://133.1.2.3/01234
```

型 inherit

その物が実装する型を inherit タグにより記述する。

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
<inherit>jp.ac.keio.sfc.ht.TV</inherit>
```

名称 name

物の名称を、name タグにより記述する。名称は、利用者に提示される情報であり、従って自然言語で記述する。また name タグは lang 属性に異なる値を指定する限りにおいて、複数個使用してよい。

–**属性** lang

–**出現数** 1 つ以上

例：

```
<name lang="en">Panasonic Diga 50inch</name>
<name lang="ja">パナソニック Diga50 インチテレビ</name>
```

ベンダー情報 vendor

その物を作成したベンダーの情報を、vendor タグにより記述する。

–**属性** なし

–**出現数** 必ず 1 つ

例：

```
<vendor>Panasonic, http://www.panasonic.jp/support</vendor>
```

解説 description

そのエンティティの全体解説を description タグに記述する。

–**属性** lang

–**出現数** 1 つ以上

例：

```
<description lang="en">hoge</description>
<description lang="ja">ほげ</description>
```

ラベル label

このエンティティに関する情報を外部化 (視覚化など) する際に使用する文字列を、その文字列に対応する識別子と組み合わせて label タグで宣言する。エラーメッセージや状態情報などを宣言しておく。

–**属性** id、lang

–**出現数** 0-*

```
<label id="-1" lang="en">Fatal Error</label>
<label id="-1" lang="ja">エラーです。サポートセンターに連絡してください</label>
<label id="-9" lang="en">Occupied</label>
<label id="-9" lang="ja">占有されています</label>
```

影響範囲 scope

このエンティティが影響を与える範囲を、幾何学図形とその大きさで表現する。影響範囲はこのエンティティが実行するオペレーションにより異なるので、properties タグ内では影響範囲の形状を複数宣言しておき、operation タグ内の provide と consume タグの属性値にその識別子を指定するものとする。

–**属性** id

–**属性値** operation タグ内で参照可能とするための任意の文字列

–**出現数** 0-*

例：

```
<scope id="audio">
  <shape>circle</shape>
  <size>5m</size>
</scope>
<scope id="video">
  <shape>fan</shape>
  <size>5m</size>
</scope>
```

実デバイスの指定 map

このエンティティの実体が、ネットワーク上に存在広告されている、もしくはネットワーク上で検索可能な何かであるとき、その実体を指定する。たとえば UPnP 対応エアコンの場合、UPnP 識別子や型を指定してそのエアコンを特定する。

–**属性** なし

–**属性値** なし

–**出現数** 0 or 1

例 1 ：
```
<map><!-- このUSDLエンティティをBonjourで広告されている特定のデバイスにマップ -->
<bonjour:device>Viera._tcp</bonjour:device>
</map>
```
例 2 ：
```
<map><!-- このUSDLエンティティをUPNPデバイスにマップ -->
<upnp:device>urn:schemas-upnp-org:device:aircon:1</upnp:device>
</map>
```

A.3.3 外部接続 interface

概要

エンティティが実装するインタフェースを、interface タグ内に operation タグにより列挙する。静的な情報である。ここでオペレーションとは、そのエンティティが持つ物理的な入出力やネットワークに対する入出力を意味する。たとえばテレビが持つアンテナ入力は、物理的な入出力の一つである。そのオペレーションが、型定義に記述されたいずれかのオペレーションを実装したものであれば、inherit 属性を用いて型定義中の該当するオペレーションの名前を示す。同属性を用いなかった場合には、そのオペレーションはそのエンティティ独自のものとなり、型検査の対象から外れる。以下に、operation タグに記述すべき内容を記載する。なお、型記述中の operation タグに記載できる事項は、本節に改めて記述していない。

–**属性** override(optional)

–**出現数** 1 つ以上

例：
```
<operation override="out">...</operation>
```

コネクティビティ connectivity

そのオペレーションを実装する通信プラットフォーム依存の事項を connectivity タグに記述する。同タグの内容は通信プラットフォームごとの schema により定義する。platform に phy を指定すると、物理的な入出力を意味する。

–**属性** platform

–**出現数** 必ず 1 つ

例：

```
<connectivity platform="ip">
  <ip:host>133.27.1.2</ip:host>
  <ip:port>12345</ip:port>
</connectivity>
```

物理的な入出力の範囲 consume/provide の scope 属性

実態の記述においてのみ、オペレーションの物理的な入出力に scope 属性を指定可能となる。scope 属性の値には、properties タグ内に宣言した scope 図形の識別子を指定する。

例：

```
<operation override="turned on">
  <description lang="en">
    transmits the current channel when the power is turned on
  </description>
  <description lang="ja">
    電源が投入されたときに現在のチャンネル番号を送信する
  </description>
  <output>
    <data type="mime">.*/.*</data>
    <description lang="en">the current channel</description>
    <description lang="ja">現在のチャンネル</description>
  </output>
  <provide scope="video">light</provide>
  <provide scope="audio">sound</provide>
</operation>
```

A.3.4 嗜好情報 preference

特定のエンティティ (人や物や場) に対して、それらのエンティティが優先的に利用したい他のエンティティや、利用を避けたい他のエンティティを指定できる。子要素として like と dislike を使用する。両タグはいくつでも使用でき、記述した順番に評価される。両タグの値として、すべてのエンティティを意味する予約後 all を導入する。

以下は、object1 の利用を最も優先し、テレビの利用を次に優先し、object2 は利用したくない場合の記述例である。

–**属性** なし

–**出現数** 0 か 1

例：

```
<preference>
  <like weight="1">all</like>
  <like weight="3">object://object1.id</like>
  <like weight="2" class="jp.ac.keio.sfc.ht.TV">all</like>
  <dislike>object://object2.id</dislike>
</preference>
```

選好 like

値に指定した識別子を持つエンティティの利用を優先する。優先する際の順位を weight 属性を用いて指定できる。weight 値が大きい物ほど優先して利用される。同一の識別子が like と dislike の双方に指定された場合には、後で指定された内容で上書きされる。class 属性を指定して、タグ値に指定した識別子の適用範囲を限定することもできる。

–**属性** weight, class

–**出現数** 0 以上

非選好 dislike

値に指定した識別子を持つエンティティの利用を避ける。like タグで使用できる優先順位は、dislike タグには使用できない。class 属性を指定して、タグ値に指定した識別子の適用範囲を限定することもできる。

–**属性** class

–**出現数** 0 以上

A.3.5 アクセス制御 access

概要

アクセス制御に関する情報を access タグにより記述する。同タグの子要素として allow と deny を使用する。両タグは幾つでも使用でき、記述した順番に評価される。

両タグの値として、全ての人や物を意味する予約語 all を導入する。以下は、特定の識別子を持つ人またはグループ等からのアクセスのみを許可するための記述例である。

–**属性** なし

–**出現数** 0 か 1

例：

```
<access>
  <deny>all</deny>
  <allow weight="1">human://human1.id</allow>
  <allow weight="2">human://human2.id</allow>
</access>
```

アクセス許可 allow

値に指定した識別子からのアクセスを許可する。許可する際の優先順位を weight 属性を用いて指定できる。weight 値が大きいほど優先して許可される。同一の識別子が allow と deny の双方に指定された場合には、後で指定された内容で上書きされる。

–**属性** weight

–**出現数** 0 以上

アクセス拒否 deny

値に指定した識別子からのアクセスを拒否する。allow タグで使用できる優先順位は、deny タグには使用できない。

–**属性** なし

–**出現数** 0 以上

A.3.6 状態情報 state

概要

このエンティティの現在の状態を state タグを用いて記述する。state タグには、このエンティティが所有あるいは利用する、もしくは利用されている他のエンティティを using、usedby タグを使用して列挙する。また current タグを用いて現在の状態ラベルを、location タグを用いて現在の位置情報を記述する。以下に、state タグに含めるべき情報を示す。

例：

```
<state>
  <current timestamp="1387682734(UNIX 時間)">-1</current>
  <history>http://somewhere/somewhat</history>
  <location field="field://heya2.id" timestamp="1387682720">
    <x>213</x><y>200</y><z>1</z>
  </location>
  <using>
    <field>field://heya.id</field>
    <object>object://tv.id</object>
    <object>object://vcr.id</object>
  </using>
  <usedby>
    <human>human://hoge.hogeid</human>
    <object>object://tv.id</human>
    <object>object://vcr.id</human>
  </usedby>
</state>
```

現在の状態 current

現時点の状態の識別子を current タグを用いて記述する。状態識別子は、properties タグ内の label タグに指定されたラベルの識別子を current タグの値として記述する。当該状態を観測した時刻は timestamp タグに UNIX 時間で記述する。

属性 timestamp

出現数 0 以上

過去の状態 history

過去の状態履歴が何らかのデータベースに格納されているとき、そのデータベース項目の URL を指定する。

属性 なし

出現数 0-1

位置情報履歴 location

現時点のこのエンティティの位置情報を location タグを用いて記述する。位置情報は、任意の field 内で用いられている座標系によって x、y、および z の 3 軸により記述

する。当該位置情報を観測した時刻は timestamp タグに UNIX 時間で記述する。

属性 field,timestamp

出現数 0-1

利用もしくは所有するエンティティ using

このエンティティが利用もしくは所有するエンティティの識別子を using タグを用いて記述する。同タグの中には、human、object、または field タグを用いてエンティティの区別を行う。

–**属性** なし

–**出現数** 0 以上

例：(ある部屋で 2 台のデバイスを使っている人の例)

```
<state>
   ...
  <using>
    <field>field://heya.id</field>
    <object>object://tv.id</object>
    <object>object://vcr.id</object>
  </using>
  <usedby>
     ...
  </usedby>
</state>
```

利用もしくは所有されているエンティティ usedby

このエンティティを利用もしくは所有している他のエンティティの識別子を usedby タグを用いて記述する。同タグの中には、human、object、または field タグを用いてエンティティの区別を行う。

–**属性** なし

–**出現数** 0 以上

例：(一人の人と 2 台のデバイスが存在する空間の例)

```
<state>
   ...
  <usedby>
```

```
    <human>human://hoge.hogeid</human>
    <object>object://tv.id</human>
    <object>object://vcr.id</human>
  </usedby>
</state>
```

A.4 付録

A.4.1 型定義の例：ジェネリックデバイス

電源の ON と OFF をネットワークから制御可能なデバイスの型定義である。

```
<type>
  <properties>
    <id>
      jp.ac.keio.sfc.ht.Device
    </id>
    <name lang="en">
      generic device
    </name>
    <name lang="ja">
      ジェネリックデバイス
    </name>
    <provider>
      Jin Nakazawa, Keio University, Japan, jin@ht.sfc.keio.ac.jp
    </provider>
    <provider>
      Tomotaka Ito, Keio University, Japan, tomotaka@ht.sfc.keio.ac.jp
    </provider>
    <version>
      1.0.0
    </version>
    <date>
      2009:01:01
    </date>
    <description lang="en">
      a generic device that can be turned on/off via a network
    </description>
    <description lang="ja">
      ネットワークから電源を制御できるジェネリックなデバイスの型
```

```
  </description>
</properties>
<operations>
  <operation name="turn on" sequence="in-out">
    <description lang="en">
      turns the power on, and sends the result
    </description>
    <description lang="ja">
      スイッチを入れて結果を送信する
    </description>
    <input>
      <data-type>
        .*/.*
      </data-type>
      <description lang="en">
        a request
      </description>
      <description lang="ja">
        リクエスト
      </description>
    </input>
    <output>
      <data-type>
        .*/.*
      </data-type>

      <description lang="en">
        the result is a positive number on success, and a negative one o
      </description>
      <description lang="ja">
        成功の場合、結果は正の数、失敗の場合は負の数
      </description>
    </output>
    <consume>
      energy
    </consume>
  </operation>
  <operation name="turn off" sequence="in-out">
    <description lang="en">
      turns the power off, and sends the result
```

```
  </description>
  <description lang="ja">
    スイッチを切って、結果を送信する
  </description>
  <input>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      a request
    </description>
    <description lang="ja">
      リクエスト
    </description>
  </input>
  <output>
    <data-type>
      .*/.*
    </data-type>

    <description lang="en">
      the result is a positive number on success, and a negative one on failure
    </description>
    <description lang="ja">
      成功の場合、結果は正の数、失敗の場合は負の数
    </description>
  </output>
</operation>

<operation name="turned on" sequence="out">
  <description lang="en">
    transmits a positive number when the power of this device has been turned on
  </description>
  <description lang="ja">
    このデバイスの電源が ON になったときに正の数を送信する
  </description>
  <output>
    <data-type>
      .*/.*
    </data-type>
```

```
        <description lang="en">
          transmits a positive number
        </description>
        <description lang="ja">
          正の数を送信する
        </description>
      </output>
      <consume>
        energy
      </consume>
    </operation>

    <operation name="turned off" sequence="out">
      <description lang="en">
        transmits a positive number when the power of this device has been t
      </description>
      <description lang="ja">
        このデバイスの電源が OFF になったときに正の数を送信する
      </description>
      <output>
        <data-type>
          .*/.*
        </data-type>
        <description lang="en">
          the result is a positive number
        </description>
        <description lang="ja">
          正の数を送信する
        </description>
      </output>
    </operation>
  </operations>
</type>
```

A.4.2　型定義の例：テレビ

電源の ON と OFF をネットワークから制御可能なテレビの型定義である。ジェネリックデバイスを継承する。

```
<type>
  <properties>
    <id>
      jp.ac.keio.sfc.ht.TV
    </id>
    <name lang="en">
      television
    </name>
    <name lang="ja">
      テレビ
    </name>
    <inherit>
      jp.ac.keio.sfc.ht.Device
    </inherit>
    <provider>
      Jin Nakazawa, Keio University, Japan, jin@ht.sfc.keio.ac.jp
    </provider>
    <provider>
      Tomotaka Ito, Keio University, Japan, tomotaka@ht.sfc.keio.ac.jp
    </provider>
    <version>
      1.0.0
    </version>
    <date>
      2009:01:01
    </date>
    <description lang="ja">
      テレビの型
    </description>
  </properties>
  <operations>
    <operation override="turn on">
      <!-- GenericDevice の turn on オペレーションの解説とかを上書き -->
      <description lag="en">
        turns the power on, and shows the current channel, then sends the result
      </description>
```

```
  <description lang="ja">
    スイッチを入れて現在のチャンネルを表示し、最後に結果を送信する
  </description>
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the result is the current channel on success, otherwise -1
    </description>
    <description lang="ja">
      成功の場合、結果は現在のチャンネル、失敗の場合は-1
    </description>
  </output>
  <provide>
    light
  </provide>
  <provide>
    sound
  </provide>
</operation>
<operation override="turn off">
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the result is the last channel on success, otherwise -1
    </description>
    <description lang="ja">
      成功の場合、結果は最後に表示していたチャンネル、失敗の場合は-1
    </description>
  </output>
</operation>
<operation name="next channel" sequence="in-out">
  <name lang="en">
    select channel
  </name>
  <name lang="ja">
    チャンネル切り替え
```

```
  </name>
  <description lang="en">
    selects the next channel, shows the channel, and sends an event that
  </description>
  <description lang="ja">
    次のチャンネルを選択してそのチャンネルを表示し、最後に結果を送信する
  </description>
  <input>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      select the next channel
    </description>
    <description lang="ja">
      次のチャンネルを選択する
    </description>
  </input>
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the result is the current channel on success, otherwise -1
    </description>
    <description lang="ja">
      成功の場合、結果は現在のチャンネル、失敗の場合は-1
    </description>
  </output>
  <consume>
    energy
  </consume>
  <provide>
    light
  </provide>
  <provide>
    sound
  </provide>
</operation>
<operation name="get current channel" sequence="out">
```

```
  <description lang="en">
    transmits the current channel, and shuts down the connection
  </description>
  <description lang="ja">
    現在のチャンネル番号を送信し、コネクションを切断する
  </description>
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the current channel
    </description>
    <description lang="ja">
      現在のチャンネル
    </description>
  </output>
  <consume>
    energy
  </consume>
  <provide>
    light
  </provide>
  <provide>
    sound
  </provide>
</operation>
<operation name="channel change event" sequence="out">
  <description lang="en">
    transmits the current channel when the channel is changed
  </description>
  <description lang="ja">
    チャンネルが変わっ時、現在のチャンネル番号を送信する
  </description>
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the current channel
```

```
    </description>
    <description lang="ja">
      現在のチャンネル
    </description>
  </output>
  <consume>
    energy
  </consume>
  <provide>
    light
  </provide>
  <provide>
    sound
  </provide>
</operation>
<operation override="turned on">
  <description lang="en">
    transmits the current channel when the power is turned on
  </description>
  <description lang="ja">
    電源が投入されたときに現在のチャンネル番号を送信する
  </description>
  <output>
    <data-type>
      .*/.*
    </data-type>
    <description lang="en">
      the current channel
    </description>
    <description lang="ja">
      現在のチャンネル
    </description>
  </output>
  <provide>
    light
  </provide>
  <provide>
    sound
  </provide>
</operation>
```

```
    <operation override="turned off">
      <description lang="en">
        transmits the last channel when the power is turned off
      </description>
      <description lang="ja">
        電源が投入されたときに最後に表示していたチャンネル番号を送信する
      </description>
      <output>
        <data-type>
          .*/.*
        </data-type>
        <description lang="en">
          the last channel
        </description>
        <description lang="ja">
          最後に表示していたチャンネル
        </description>
      </output>
    </operation>
  </operations>
</type>
```

A.4.3 型定義の例：ネットワークテレビ

電源の ON と OFF をネットワークから制御可能で、かつネットワークからの映像音声入力を持つテレビの型定義である。テレビを継承する。

```
<type>
  <properties>
    <id>
      jp.ac.keio.sfc.ht.NetworkedTV
    </id>
    <name lang="en">
      television
    </name>
    <name lang="ja">
      テレビ
    </name>
    <inherit>
      jp.ac.keio.sfc.ht.TV
    </inherit>
    <provider>
      Jin Nakazawa, Keio University, Japan, jin@ht.sfc.keio.ac.jp
    </provider>
    <version>
      1.0.0
    </version>
    <date>
      2009:01:01
    </date>
    <description lang="ja">
      ネットワークから映像音声入力を行えるテレビの型
    </description>
  </properties>
  <operations>
    <operation name="video in" sequence="in">
      <input>
        <data-type>
          video/.*
        </data-type>
        <description lang="en">
          receives a formatted video stream and visualize it
        </description>
```

```
        <description lang="ja">
          映像と音声の両トラックを含む動画ストリームを受信して表示する
        </description>
      </input>
      <output>
        <data-type>
          video/.*
        </data-type>
        <description lang="en">
          forwards the received video stream
        </description>
      </output>
      <consume>
        energy
      </consume>
      <provide>
        light
      </provide>
      <provide>
        sound
      </provide>
    </operation>
  </operations>
</type>
```

A.4.4　空間定義の例：部屋

テレビが1台あり、人が二人いる部屋の例である。

```
<entity>
  <properties>
    <id>
      piax://host.piax.ac.jp/area/1
    </id>
    <inherit>
      jp.ac.keio.sfc.ht.PrivateArea
    </inherit>
    <name>
      徹子の部屋
    </name>
    <vendor>
      不明
    </vendor>
    <description lang="ja">
      言わずとしれた
    </description>
  </properties>
  <operations>
    <operation override="list entity">
      <connectivity platform="piax">
        <piax:agent>
          <!-- エージェント識別子？ -->
        </piax:agent>
      </connectivity>
    </operation>
  </operations>
  <access>
    <deny>
      all
    </deny>
    <allow>
      piax://host.piax.ac.jp/human/1
    </allow>
  </access>
  <state>
    <current/>
```

```
    <location field="field://テレビ朝日.id" timestamp="123457000">
      <x>
        ???
      </x>
      <y>
        ???
      </y>
      <z>
        ???
      </z>
    </location>
    <using/>
    <usedby>
      <human>
        human://host.piax.ac.jp/human/1
      </human>
      <!-- 徹子 -->
      <human>
       human://host.piax.ac.jp/human/452342
      </human>
      <!-- ウィルスミス -->
      <object>
       object://host.piax.ac.jp/tv/1
      </object>
    </usedby>
  </state>
</entity>
```

A.4.5　空間定義の例：マイテレビ

上記の部屋にあるテレビの例である。

```
<entity>
  <properties>
    <id>
      piax://host.piax.ac.jp/object/1
    </id>
    <inherit>
      jp.ac.keio.sfc.ht.NetworkTV
    </inherit>
    <name>
      徹子のビエラ
    </name>
    <vendor>
      Panasonic
    </vendor>
    <description lang="ja">
      プラズマテレビ
    </description>
  </properties>
  <operations>
    <operation override="video in">
      <connectivity platform="rtsp">
        <rtsp:uri>
          rtsp://www.hoge.net/hoge.ram
        </rtsp:uri>
      </connectivity>
      <input>
        <data-type>
          video/ram
        </data-type>
        <description lang="en">
          receives a formatted video stream and visualize it
        </description>
        <description lang="ja">
          映像と音声の両トラックを含む動画ストリームを受信して表示する
        </description>
      </input>
      <output>
```

```
      <data-type>
        video/ram
      </data-type>
      <description lang="en">
        forwards the received video stream
      </description>
    </output>
  </operation>
  <operation override="turn on">
    <!-- GenericDevice の turn on オペレーションの解説とかを上書き -->
    <connectivity platform="dlna">
      <dlna:uri>
        ???
      </dlna:uri>
    </connectivity>
  </operation>
  <!-- 後は省略 -->
</operations>
<access>
  <allow>
    all
  </allow>
  <!-- 誰でもこのテレビの操作ができる -->
</access>
<state>
  <current/>
  <location field="field://徹子の部屋.id" timestamp="123457000">
    <x>
      ???
    </x>
    <y>
      ???
    </y>
    <z>
      ???
    </z>
  </location>
  <using>
    <field>
      field://host.piax.ac.jp/field/3
```

```
      </field>
    </using>
    <usedby>
      <human>
        human://host.piax.ac.jp/human/1
      </human>
      <!-- 徹子 -->
      <human>
        human://host.piax.ac.jp/human/452342
      </human>
      <!-- ウィルスミス -->
    </usedby>
  </state>
</entity>
```