修士論文　２００３年度（平成１５年度）

分散通信アプリケーションの
動的適応と機能拡張を支援する
セッション機構に関する研究

慶應義塾大学大学院 政策・メディア研究科

権 藤 俊 一

修士論文要旨　２００３年度（平成１５年度）

分散通信アプリケーションの動的適応と機能拡張を支援するセッション機構に関する研究

論文要旨

本研究の目的は，人の周囲に遍在する機器で動作するアプリケーションに対し，通信機能の動的適応と機能拡張を支援する基盤環境を提供することである．

本研究で対象とするユビキタスコンピューティング環境は，機器間の相互通信を前提とした異質かつ動的な分散環境である．このため，機器で動作するアプリケーションが相互に通信し，連携して動作する必要がある．しかし，従来の分散アプリケーション構築手法では，通信機能が通信アプリケーションと同化しているため，通信機能において多様性への対応を実現することが困難である．

このため本論文では，通信機能の動的適応と機能拡張を支援するセッション機構を提案し，u-session システムとして設計・実装した．u-session システムは，通信アプリケーションから通信機能を分離し，目的を指定することにより通信機能の適応動作を可能とする機構である．また，特定のプログラミング言語や実行環境に依存しない設計であるため，異質で多様なユビキタスコンピューティング環境において，“いつでも・どこでも”十分に機能する基盤環境として動作できる．そして，本論文で提案した機構により，機器の柔軟な動作環境を構築することが可能となる．

キーワード

1 セッション基盤機構，2 ユビキタス通信ミドルウェア，3 動的適応と機能拡張，4 環境非依存性

慶應義塾大学大学院 政策・メディア研究科

権 藤 俊 一

Abstract of Master's Thesis Academic Year 2003

u-session: wish based dynamic adaptation and extension architecture for distributed communication software

Summary

In this thesis, we describe the architecture called u-session which provides a wish based dynamic adaptation and extension mechanism of the communication function for distributed communication software that runs on the device around a user.

Ubiquitous Computing Environment is the heterogeneous and dynamic distributed environment. In such environment, it is required for the application software on the device to communication each other to do a coordinated task over the network. However, the present architecture for distributed application software assume application function and communication function as a single function, it is difficult to achieve flexibility on the communication function.

In this thesis, to solve this problem, we propose u-session which is an session architecture to support dynamic adaptation and extension of communication function, and we designed and implemented it. u-session separates a communication function from each application software, and enables application software to run independently from the communication platform. Then, the communication platform can be controlled by specified wish of developer, user and provider. u-session is designed independently from programming language and run-time, so it can run anytime, anywhere on heterogeneous Ubiquitous Computing Environment. With u-session, the developer enables flexibility on communication application software easily, and achieve flexible user environment of devices and distributed communication software.

Keywords

1 Session Layer Architecture, 2 Communication Middleware for Ubiquitous Computing Environment, 3 Dynamic Adaptation and Extention , 4 Platform Independency

Keio University. Graduate School of Media and Governance,

Shunichi Gondo

目 次

第 **1** 章　序論　**1**

1.1　背景 2
1.2　目的と意義 4
1.3　本論文の構成 5

第 **2** 章　通信アプリケーションとユビキタスコンピューティング環境　**6**

2.1　ユビキタスコンピューティング環境の特徴 7
2.1.1　機器とアプリケーションソフトウェア 7
2.1.2　異質な動作環境 8
2.1.3　動的分散環境 9
2.2　分散協調 API の分類 9
2.2.1　ストリーム型 10
2.2.2　手続き呼出し型 11
2.2.3　移動オブジェクト型 11
2.3　環境変化に対するアプローチ 12
2.3.1　機器指向・アプリケーション指向・タスク指向 12
2.3.2　単純持続と状態継続 14
2.3.3　適応動作と機能拡張 14
2.4　問題点と限界 15
2.4.1　多様性への対応方法 15
2.4.2　挙動制御の指定方法 16

第 **3** 章　動的適応と機能拡張を支援するセッション機構　**17**

3.1　要求機能の整理 18
3.2　動的適応と機能拡張の支援 18
3.2.1　動的適応 19
3.2.2　機能拡張 20
3.3　セッション機構 21
3.3.1　通信機能の分離 22
3.3.2　柔軟性の実現 24
3.3.3　分散通信アプリケーションにおける利点 25
3.4　柔軟な挙動の指定方法 25

3.4.1 Wish による適応と拡張の指定 26
3.4.2 Wish の評価選択機構 27
3.5 関連研究 27
3.5.1 Rocks and racks 28
3.5.2 MobileSocket 28
3.5.3 SinkProxy 29
3.5.4 SoNS 29
3.5.5 関連研究の相違 30
3.6 まとめ 30

第 4 章 設計 **31**
4.1 概要 32
4.1.1 設計方針 32
4.1.2 全体構成 33
4.1.3 動作概要 34
4.2 通信基盤依存処理部 36
4.2.1 通信基盤依存処理部の機能 36
4.2.2 通信基盤依存処理部の拡張性 36
4.3 動作制御部 37
4.3.1 動作制御部の機能 37
4.3.2 制御方法 38
4.4 Wish と Wish 解析部 38
4.4.1 Wish 38
4.4.2 Wish 解析部 39
4.5 通信基盤提供部 39
4.5.1 通信基盤提供部の構成 39
4.5.2 通信基盤提供部が提供する API 40

第 5 章 実装 **41**
5.1 u-session の実装 42
5.1.1 概要 42
5.1.2 動作の流れ 43
5.2 session - 通信基盤依存処理部の実装 43
5.2.1 実装した関数 43
5.2.2 拡張性の実現方法 44
5.3 xdsession - 動作制御部の実装 45
5.3.1 実装した関数 45
5.3.2 制御方法の実装 46
5.4 Wish Manager - Wish 解析部の実装 47
5.4.1 動作制御部に対するインタフェースの実装 47

5.4.2 通信相手の検索方法 48
5.5 xdsocket - 通信基盤提供部の実装 48

第 6 章 評価 **50**
6.1 要求機能との考察 51
6.2 動作検証 52
6.3 議論 53
6.3.1 関連研究との比較 53

第 7 章 結論 **54**

図目次

1.1 ユビキタスコンピューティング環境の目的 2
1.2 オーディオプレーヤの例 3
1.3 u-session システムの概要 4

2.1 異質な動作環境 8
2.2 分散協調 API の分類 10
2.3 ホスト指向・アプリケーション指向・タスク指向 13

3.1 動的適応と機能拡張の例 19
3.2 既存の通信アプリケーションの構成と OSI 参照モデル 22
3.3 u-session の概要 23
3.4 動的適応の例 26

4.1 u-session の全体構成 34
4.2 u-session の動作概要 35
4.3 通信基盤依存処理部における拡張性 37
4.4 通信基盤提供部の構成 39

5.1 u-session システムの実装概要図 42
5.2 session 構造体 44
5.3 機能拡張を実現するマクロ 45
5.4 xdsession 構造体 46
5.5 xdsocket の概要 48
5.6 sockaddr_wish 構造体 49

6.1 各セッションの通信シーケンス図 52

表 目 次

3.1 セッション機構の位置づけ 22
3.2 評価選択機構の分類 27
3.3 関連研究の相違 30

4.1 通信基盤依存処理部の機能 36
4.2 動作制御部の機能 37
4.3 動作制御部の機能 40

5.1 実装環境 42
5.2 通信基盤依存処理部の関数プロトタイプ宣言 43
5.3 動作制御部の関数プロトタイプ宣言 45

6.1 関連研究の相違 53

第1章

序論

本章では，はじめに本研究を進める背景を明らかにした上で，本研究の目的と意義および本論文の構成について述べる.

1.1 背景

技術開発の進展が機器の小型化と高機能化を牽引し，パソコンや携帯電話をはじめとするデジタル機器の利用が，多くの人々にとっても日常的なものとなっている．これと並行して，いわゆるブロードバンドの普及に加え，屋外での無線 LAN の利用 [1] や常時接続型のモバイル通信ネットワーク [2] などの通信サービスが展開され，低コストで利便性や機能性の高いインターネットへの接続性が日増しに向上している．今日のこうした状況は，一般的に“ユビキタスネットワーク社会”の到来と言われる．これは，用途や形態の異なる様々な機器を用いて，時には機器の利用すら意識せずに“いつでも・どこでも・だれでも”情報のやりとりが可能なこと意味し，その由来は Mark Weiser 博士によるユビキタスコンピューティング環境の提唱 [3] に端を発する．

ユビキタスコンピューティング環境の目的は，大別すると図 1.1 に示す 2 つである．

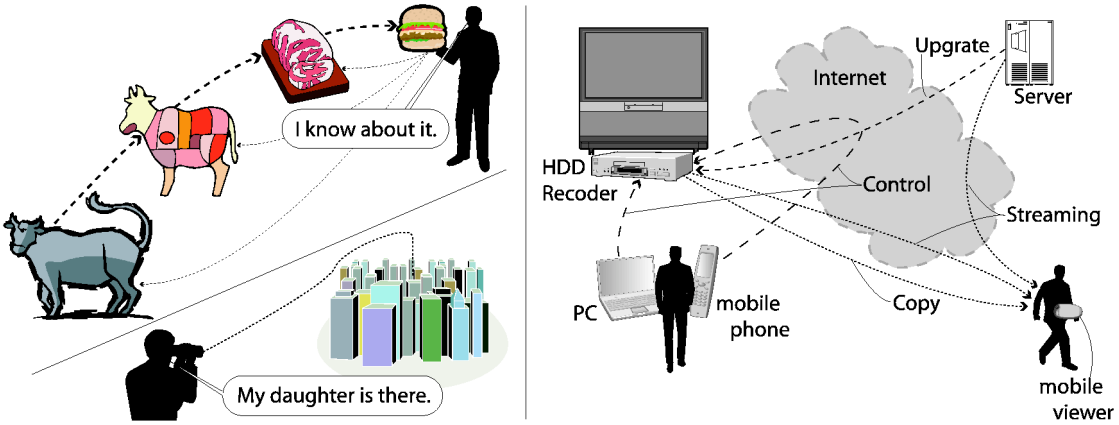

図 1.1: ユビキタスコンピューティング環境の目的

トレーサビリティとは，電子タグ [4] などを用いて，物品や人などに関する属性や状態などの情報を常に追跡可能とすることである．具体的には，商品の流通管理や建物・車両などの状態管理をはじめとして，子供などに対する位置確認などのサービス [5] が実用化されている．一方，機器の柔軟な利用環境とは，機器の動作に柔軟性を持たせることにより，機器そのものに生じる制約を解消することである．なお，これ以降の本論文において，ユビキタスコンピューティング環境という場合には，主に後者の目的で用いられる場合を想定する．

さて，ここで述べた機器そのものに生じる制約の例を示す．それは，物理的な大きさや性能や記憶装置の容量などをはじめ，機器に予め実装された機能や機器が準拠する仕様により生じるものなどである．例えば，小型な機器では，機器単体で使いやすいユーザインタフェースを提供することが，物理的なサイズやコストの面から見て難しい．そこで，通信機能を持たせることにより，この制約を解消している．その最も原始的な例が，機器本体と操作する機器を分離し，機器操作時の距離や操作性といった問題を解決した，リモートコントローラである．これは操作する機器に対応していれば，より操作性の高いものに置き換え可能であるという点で，複数の機器が通信機能により連携して

いるものと言える．一方，今日のハードディスクレコーダなどでは，パソコンやインターネット上のサーバなど外部機器と連動し，より高機能で使いやすいユーザインタフェースを実現している [6, 7]．また，この方法ではインターネット上のサーバに対する変更により，機器の操作性を向上することが可能である．同様にして，機器自体に対する変更をせずとも，パソコンや携帯端末など操作する機器に適したユーザインタフェースを提供することが可能である．そして，機器自体の制御ソフトウェアをインターネット経由でダウンロードし，その機能を拡張することも可能である．

このように，機器の通信機能が分散された機器を協調させ，機器そのものに生じる機能の制約を解消することができる．これにより，機器の柔軟な利用環境の実現も可能となる．つまり，通信機能は，機器の動作や機能に対して拡張性や適応性を実現するなど，機器の動作に柔軟性を持たせる上で不可欠であると言える．そして，“ユビキタスネットワーク社会”の到来が，こうした機器間での連携を日常的なものとするために十分な基盤を形作ったことは言うまでもない．特に今後，機器が一層の小型化と高機能化を進める結果，機器間での通信機能がより大きな役割を果たすことは自明である．例えば，オーディオプレーヤを例に考えると分かりやすい．この場合の例を図 1.2 に示す．

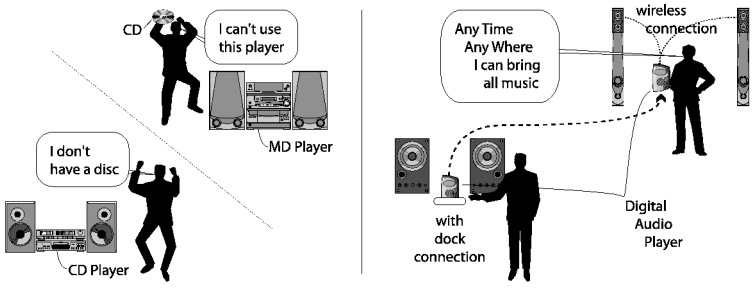

図 1.2: オーディオプレーヤの例

まず，CD や MD では音楽を聴くために専用のプレーヤが必要であり，逆にプレーヤがあっても聴きたい音楽が収録されたディスクがなければ意味がない．このため，例えば車内などでは，聴きたい音楽があってもプレーヤがないという事態や，逆にプレーヤがあっても聴きたい音楽がないという利用時の制約が生じてしまう．一方，昨今のデジタルオーディオプレーヤ [8] など，一般的な個人が所有する数に迫る楽曲を電子データとして収録可能な機器[1] が存在する．これらの機器は，そのサイズなどの面では従来の携帯型オーディオプレーヤと変わらないため，従来の機器と置き換えてその制約を解消することが可能であると言える．また，FM 電波を用いた通信機能 [9] により，例えば車内のステレオから音声を出力可能である．この場合の通信機能は単一方向であるが，双方向の通信機能をオーディオプレーヤと車載ステレオの双方に設けることで，より付加価値の高いサービスを提供することが可能となる．その例としては，運転席付近のスイッチや表示装置を用いたオーディオプレーヤの操作や，車載の広域通信ネットワークを通じた楽曲のダウンロード機能などが考えられる．これが可能となれば，携帯可能な楽曲数が無限になるものと言って良い．

[1] 2004 年 1 月現在，12cm CD にして 800 枚程度．

このように，通信機能を持つ機器が相互に連携することで柔軟な利用環境が実現され，利用時の制約を解消することでより便利になるだけではなく，新たなサービスの登場も可能となる．しかし，人が周囲に遍在する多くの機器を用いるようになるにつれ，様々な機器とその利用形態を想定する必要が生じ，それを支援する通信機能が柔軟に対応できなくなってしまう懸念がある．その原因として，以下に示す４つのものが挙げられる．まず，機器において通信機能が十分に機能分散して実装されていない．このため，例えば新たな通信方式への対応が必要な場合に，個別のアプリケーションなどで対応する実装を施す負担を強いられてしまう．また，ユビキタスコンピューティング環境という，様々な機器が存在する異質な環境下で，“いつでも・どこでも”十分に機能すると考えられる基盤環境も存在しない．通信アプリケーションの基盤環境が実行環境に対して依存する場合，それが利用可能な範囲や拡張性に対して制約を生じさせるため，機器の柔軟な利用環境を実現できない．そして，人それぞれ利用形態のニーズは多様であり，予めすべてを想定して実装することは不可能と言って良い．また，ユーザに機器の連携動作を強く意識させなければ利用できないようでは利用者の負担となり，“だれでも”日常的に利用できるとは考えられない．したがって，ここに挙げた課題を解決する機器の通信機能を実現することが，柔軟な機器の利用環境を実現する上で重要なものとなる．

1.2 目的と意義

本研究の目的は，人の周囲に遍在する機器で動作するアプリケーションに対し，通信機能の動的適応と機能拡張を支援するための基盤環境を提供することである．本論文では，分散通信アプリケーションの動的適応と機能拡張を支援を目的とした u-session システムを設計・実装する．u-session システムの概要を図 1.3 に示す．

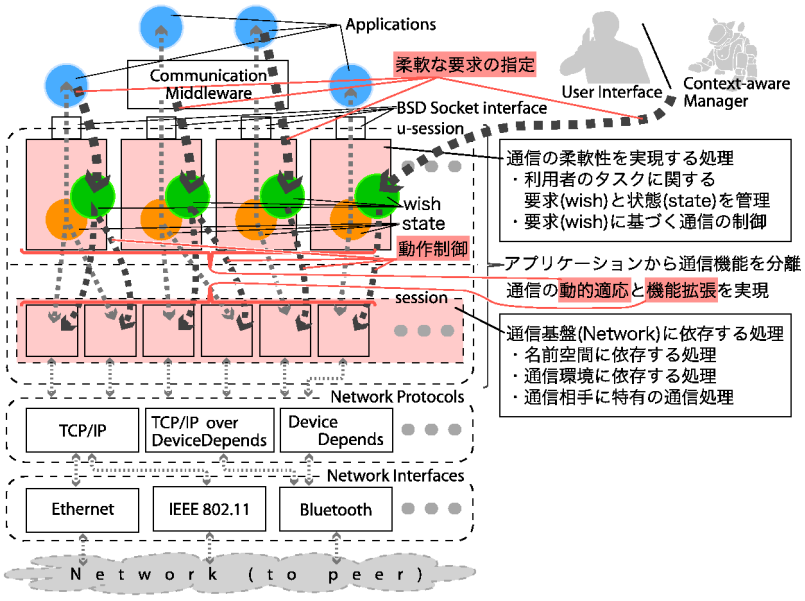

図 1.3: u-session システムの概要

ユビキタスコンピューティング環境は，様々な機器が存在する異質な機器の利用環境であることが前提とされる．そのため，機器の実装や通信方式が多様であり，それらを連携して動作させる場合，多様性や異質性に関わらず機器が連携して動作できなければ，利用者の目的を達成する上で大きな障害となる．しかし，それぞれの機器で動作するアプリケーションにおいて，通信機能が十分に機能分散されているとは言えない．このため，同一性が前提とはできないユビキタスコンピューティング環境下において，多種多様な機器を連携するための拡張性や適応性を実現することができない．つまり，アプリケーションが可能な通信や連携の動作が，予め決められた範囲に限られてしまうという問題が生じる．つまり，現状ではユビキタスコンピューティング環境のための通信機構が十分に整っているとは言えず，この分野において研究を進める余地が残されていると言える．

この問題を解決するため，新たなアプリケーションや通信方式などが登場した場合であっても，機器やアプリケーションは自らの機能を拡張して連携することができなければならない．しかし，アプリケーションそれぞれによる対応では，個別に対応する際のコストが高くなり，対応された実装が提供されるまで機器を連携できないなどといった問題が生じる．そこで，機器の柔軟な連携を実現する際に最も重要となる通信機能において予め拡張性を持たせ，様々な場合に対応するための機能の動的な適応や拡張を可能とする必要がある．

これを実現するためには，通信機能をアプリケーションから独立させ，アプリケーションから個別の通信機能に固有な処理や情報を分離することが必要である．本論文では，これを u-session により実現する．u-session は，通信アプリケーションから通信機能を分離し，通信機能の動的適応と機能拡張を実現するためのセッション機構である．この機構では，利用者の目的に応じた通信処理の制御を可能とし，アプリケーションに代わり特定の通信方式や通信相手に依存した通信処理を実行する．これにより，ユビキタスコンピューティング環境の目的として挙げた，機器の柔軟な利用環境を実現することが可能となる．

1.3 本論文の構成

本論文は，本章を含め全７章から構成される．次章では，通信アプリケーションがユビキタスコンピューティング環境において動作する際に示す特徴を述べ，既存の分散アプリケーション構築手法をそれぞれの視点の違いから，目的や環境変化に対応可能な範囲の違いについて整理する．また，既存手法での問題点と限界や解決すべき課題を明らかにする．第３章では，前節で示した問題の解決方法として u-session を提案する．u-session は，分散通信アプリケーションの動的適応と機能拡張を支援するセッション機構である．第４章では，設計について述べる．第５章では，実装について説明する．第６章では，評価について述べる．最後に第７章では，まとめと今後の課題について述べ，本論文の結論とする．

第2章

通信アプリケーションと
ユビキタスコンピューティング環境

本章では，通信アプリケーションの視点から，ユビキタスコンピューティング環境において見受けられる特徴と問題点を整理して説明する．まず，通信アプリケーションの視点から本論文で想定するユビキタスコンピューティング環境の特徴を分析し，それが機器間の相互通信を前提とした異質かつ動的な分散環境であることを明らかにする．次に，既存の分散アプリケーション構築手法をその視点の違いから分類して説明し，その目的と環境変化に対応可能な範囲の違いを整理して述べる．最終節では，既存手法での問題点と限界について述べ，解決すべき課題を明らかにする．

2.1 ユビキタスコンピューティング環境の特徴

ユビキタスコンピューティング環境では，すべての機器が何らかの通信機能を有する必然性がある．この理由は，第 1.1 節において説明した通り，機器の通信機能が分散された機器を協調させ，機器そのものに生じる機能の制約を解消することができるためである．そして，このことによりユビキタスコンピューティング環境の目的として挙げた，機器の柔軟な利用環境を構築することが可能となる．しかし，いくつかの理由により，ユビキタスコンピューティング環境において十分に機能すると考えられる通信機構が存在しないことも既に述べた．本章では，その詳しい説明も含め，通信アプリケーションの視点からユビキタスコンピューティング環境について分析する．

2.1.1 機器とアプリケーションソフトウェア

第 1.1 節にも述べた通り，ユビキタスコンピューティング環境とは，用途や形態の異なる様々な機器を用いて，時には機器の利用すら意識せずに“いつでも・どこでも・だれでも”情報のやりとりが可能な環境を意味する．今日の日常生活においても，我々の周囲には多くの機器が存在する．例えば，電話・インタフォンやテレビ・ラジオなどの通信機器をはじめ，パソコンや家電などの機器，センサやタイマなどの機器がある．このように用途や形態に合わせ多種多様な機器が存在するが，これらに共通する目的をまとめると“人と人や社会を繋ぐこと”そして“人を助けること”であると言える．言い換えれば，その目的は“人と人や社会のコミュニケーション”と“人の作業支援”である．

だが，それぞれの機器は単体で固有の機能を実装しており，それが実現できることは極めて単純である．このため，その本来の目的と実際の機能との間には大きな差があり，以下で示すような問題や不都合を生じさせる．まず，機器は予め決められた単純な動作を繰り返すことしかできない．例えば，空調やポットは人の都合や存在とは関係なく指定された温度に保つことのみが可能であり，電話はその場や人の都合に関係なく着信を伝える．同様に，問題が生じた際も停止してベルなどを鳴らすだけなどの場合が多い．一方，利用者はある機器を使うことに関して，その機器の動作を強く意識する必要がある．例えば，エレベータに乗る人は，上下どちらの方向に向かうかを告げた後，行き先の階に停止することを自ら指定し，到着を確認しなければならない．

今後，より多くの機器が我々の日常に遍在するようになるにつれ，利用する際に機器の機能を正確に把握して適切に利用することが強いられるようでは，利用者に対し大きな負担を与えることとなる．そして，機器の利用すら意識させずに人の目的を達成することも実現できない．これは，ユビキタスコンピューティング環境において，通信機能により様々な機器を連携させて利用する場合にも同様に言える．つまり，機器を連携させる際の指示判断や通信方法の選択などを常に利用者の指定に委ねることはできない．そもそも，こうした事態が懸念される原因は，機器の機能に対してその本来の目的を達成するためのシステムが未発達なためである．そして，そのようなシステムを実現するためのアプリケーションが不在であるために生じるものと言える．

しかし，ユビキタスコンピューティング環境を実現するためには，機器の柔軟な利用

環境の実現が不可欠であり，そのために機器を連携させるためのシステムやアプリケーションをいかにして実現するかということが問題となる．ここで必要とされるアプリケーションでは，人や機器が置かれる状況や変化に左右されずに，その人の目的を達成することができなければならならい．さもなければ，異質で多様なユビキタスコンピューティング環境では動作できない．これについて次節で説明する．

2.1.2 異質な動作環境

機器はその用途や形態という面で多様である．これは，利用者の需要をはじめ物理的なサイズやコストなどにより生じる．ここでは，実装形態と通信方式の多様性という 2 つの面から論じる．また，以下で想定する例を図 2.1 に示す．

まず 1 つとして，同じ機能であっても利用場面に応じて実装の形態は様々であるという面がある．例えば，DVD ビデオを再生する場合，DVD プレーヤなどの専用機器では映像処理に専用のハードウェアを用いる場合が多い．だが，パソコンなどの汎用機器で再生する場合には，同様の処理をソフトウェアにより処理する場合が多い．これは，単純処理の性能や品質を最適化することに優れたハードウェアの力と機器に様々な機能を比較的容易に拡張できるソフトウェアの力とのバランスが，用途により変化することを示す良い例である．

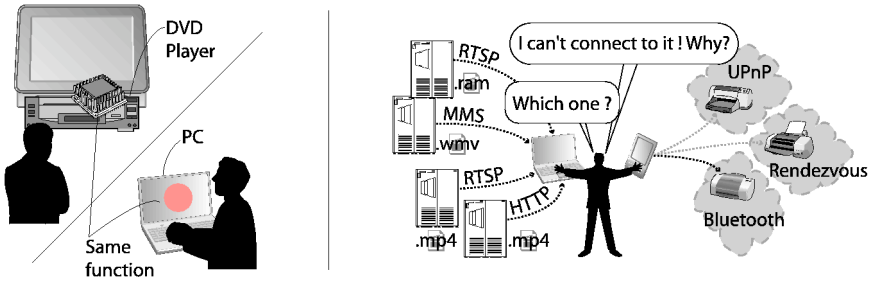

図 2.1: 異質な動作環境

一方，もう 1 つの面として，通信の方式は多様であり，変化するということがある．例えば，機器間の連携に不可欠なサービス検索や利用時の通信プロトコルも目的や用途に応じて多様である．そして，あらゆる場合に対応できるものにすれば一長一短となってしまうため，すべての場合で統一した方式になるとは考えられない．これは，利用環境が異質で多様なためである．また，オーディオやビデオのストリーミング通信プロトコルに限って見ても，現在展開されているストリーミングサービスやストリーミングプレーヤを見れば，複数の通信方式を必要とすることがわかる．

そして，通信機能の多様性や変化するという事実を無視すると様々問題が生じる．例えば，様々な通信方式が存在するために，対応する通信方式や利用するアプリケーションにより，利用する機器や接続先を適宜ユーザが選択するという事態が生じている．このため，ユーザが理解できる範囲を越えた選択肢を提示できない．また，機能的には問題なくとも通信機能の問題で機器を柔軟に組み合わせることができないという問題があ

る．これは，同一の通信プロトコルであっても実装の程度が自由な場合，アプリケーションごとに対応可能な機能が異なることにより生じる．これでは，サービス側で新たなソフトウェアやその新機能を用いたいとしても，利用者側に同等の機能が普及し十分に理解されるまでは利用できなくなってしまう．つまり，通信アプリケーションの連携に適応性や拡張性が不在であるため，新たなサービスが普及する際の障害となる．

このように，機器に機能を実装する際の自由度と通信機能の柔軟性が，異質で多様なユビキタスコンピューティング環境の基盤となる．これに対応可能でなければ，“いつでも・どこでも・だれでも”を前提とした機器の柔軟な利用環境を提供できない．

2.1.3 動的分散環境

ユビキタスコンピューティング環境において，分散して動作する機器は常に変化する．変化には，人や機器などが置かれている状況の変化をはじめ，移動や機器の故障により生じるもの，また第 1.1 節で述べた機器の機能更新等による変化などがある．こうした変化を予測して動作することは難しく，またそれ以上に，様々な変化への適応を予め機器に実装しておくことは不可能と言って良い．しかし，変化への対応をすべて利用者の判断に委ねてしまうと，利用者に対して大きな負担を生じさせる結果となる．

特に，機器の連携が進むにつれ，どの変化に対してどう対応するかという方針が複雑になる．また，機器の連携は相互通信が前提であるため，適応動作も機器が互いに連携して実行しなければならならい．つまり，適応性や拡張性が高くなればなるほど，それを実現する機能の構築方法や実行方法にも柔軟性が求められる．同時に，目的や意図する動作を指定する場合の方法についても柔軟性が求められる．その理由は，変化に対するすべての対応方法を予めプログラミングできないことに対し，その判断や指示を利用者に対しても常に求めることはできないという矛盾を解決するためである．これらの要求を達成することなく，柔軟な機器の利用環境を実現することはできない．

2.2 分散協調 API の分類

本節では，分散アプリケーションを構築するためのプログラミングインタフェースである分散協調 API[1]について説明する．既存の分散協調 API には，大きく分けて 3 種類の方式が存在する．それは，ストリーム型，手続き呼び出し型，移動オブジェクト型である．これらについて図 2.2 に示した．

分散アプリケーションの基本的な構造は，いずれの場合においても共通である．まず，物理的な機器があり，それぞれの機器ではそれぞれの目的を果たすためのアプリケーションが実装されている．また，それぞれのアプリケーションに実装された個別の機能がモジュールである．つまり，モジュールの機能が結び付き，目的を達成するためのアプリケーションとなる．このため，1 つのアプリケーションには 1 つ以上の機能すなわちモジュールが存在する．そして，複数のアプリケーションの機能を結び付け，連携させる

[1]Application Programming Interface

ものとして通信機能がある．

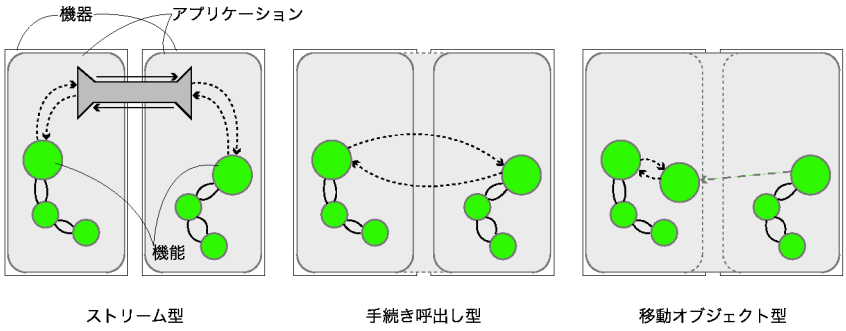

図 2.2: 分散協調 API の分類

2.2.1 ストリーム型

この方式の特徴は，通信機能がアプリケーション間に仮想的な通信路を設け，複数のアプリケーションを連絡することである．この方式の具体例として，BSD Socket インタフェース [10, 11] や Unix オペレーティングシステム [12] の PIPE システムコール [13] や，それから派生した JXTA [14] 等などがある．

この方式では，まずアプリケーションは，基盤として存在する通信機能に対し，仮想的な通信路に対する識別子の発行を依頼する．その後，アプリケーションが通信を開始する場合には，通信機能に対し識別子とともに遠隔の相手先を指定する．これをもとに通信機能は，識別子に対応する通信路と指定された相手に対する連絡を可能とする．その後，アプリケーションが通信路の識別子を用いて送受信すれば，通信機能を経由して相手先と送受信をすることができる．また，アプリケーションが通信を受ける場合には，通信機能から着信を知らされ，その後は同様に通信路の識別子を用いて，相手先との送受信が可能となる．つまり，アプリケーションは，識別子による通信路への読み書きのみで，相手先のアプリケーションと通信することができることになる．この場合，アプリケーションが通信機能に対してデータを送受信することは，相手先のアプリケーションに対して送受信することと同義となる．このため，アプリケーションは相手先を通信路の識別子で管理し，通信機能はアプリケーションが指定した相手先までの連絡方法とアプリケーションに告げた識別子との対応を管理すればよい．

このように，通信機能がアプリケーションに提供する機能が単純であるため，その実装も単純なものとすることができる．そのため，この方式は通信機能を持つほとんどの機器において実装され，この他の通信 API などにおいてもその基盤として用いられる場合が多い．しかし，基盤として提供された機能が単純であるがゆえ，複雑な通信処理はすべてアプリケーションにおいて実装しなければならない．具体的には，サービスの検索や相手の機器との間の通信処理など，アプリケーションの連携や通信に必要なすべての機能が含まれる．これらの機能が共有ライブラリにより実現されていても，アプリケーション内部で予め指定された機能から変化できないことには変わりない．このため，ア

プリケーション内の機能と通信機能を切り放して考えることができないと言える．そして，アプリケーション内での機能の隠蔽は，柔軟性という面で制約を生じさせる．

2.2.2 手続き呼出し型

この方式の特徴は，アプリケーションに対し，遠隔のアプリケーションに実装された機能と自らに実装された機能との区別を意識させなくするということである．この方式の具体例としては，RPC [15] と RMI [16] や CORBA [17]，SOAP [18] などがある．

この方式では，アプリケーションが，遠隔のアプリケーションに実装された機能を用いる場合，その機能への呼び出し先として，自らに実装されたスタブと呼ばれる仮の呼び出し先を用いる．スタブはそれぞれの機能ごとに存在し，通信機能により予め提供されるものである．通信機能はスタブが呼び出されると，遠隔のアプリケーションに実装されたその対応する機能を実際に呼び出す．つまり，遠隔のアプリケーションにある機能をスタブにより自らのアプリケーションの機能として取り込んでいるものと言える．

アプリケーションの開発者や利用者の視点から見れば，連携する際の通信は通信機能により隠蔽されるため，特に意識する必要はない．つまり，分散されたアプリケーションを単一のアプリケーションとして見ているといえる．また，アプリケーションが利用できる機能は，作成時に予めスタブとして分散させておいた機能に限られる．このため，分散の形態をアプリケーション構築時に決定する必要があり，事後に変更や拡張をすることができない．そして，この方式は，極度の連携動作を必要とする一部の分野では用いられているが，多くの機器や環境で使われてはいない．多くの場合，機能変更をする場合にあるアプリケーション単独の変更では限界があり，連携して動作するすべてのアプリケーションにおいて修正を加えなければならないという問題がある．これは，分散されたアプリケーションを単一のアプリケーションとして見ているゆえに生じる問題でもある．このため，開発者がアプリケーションのすべてに関知できる場合以外には，分散した連携が機能しなくなることは自明である．つまり，特定の通信 API に依存した密な連係動作は，異質性や多様性への対応という面で制約を生じさせる．

2.2.3 移動オブジェクト型

この方式の特徴は，アプリケーションを個別の機器から独立させ，アプリケーション自体が複数の機器間で移動して連携することである．この方式の具体例としては，Process Migration [19] や Mobile Agent [20] などがある．

この方式では，ある機器で動作するアプリケーションが別の機器に固有の機能を用いる場合，アプリケーション自らがその機器に移動して目的の機能を利用する．このため，機器間でアプリケーションが移動して動作可能な環境を整えることがすべての前提となる．アプリケーションが動作中の状態で移動可能なため，アプリケーションの処理や動作状態など複雑な情報をアプリケーション内部に隠蔽したまま，異なる機器で動作を続けることができる．このため，アプリケーション開発者や利用者は，動作しているアプ

リケーションがまったくそのままの状態で別の機器においても実行され続けることを期待できる．これは，アプリケーションが環境の変化に対して能動的に適応するという方針をとらず，共通の実行環境において動作するために環境間の相違を受動的に吸収することを前提としているためである．つまり，機器や環境に適した機能に最適化して実行するよりは，すべての機器や環境で最大公約数的に実装された機能を常に実行するものであると言える．

この方式の場合，通信機能が果たす役割は２つある．それは，アプリケーションが移動する手段を提供することとこの方式に対応しないアプリケーションと通信する手段を提供することである．これらは，これまでに述べた２つの方式のいずれかにより実現される．このため，それぞれの場合で述べた問題は，以前として解決されずに残る．また，機器の多様性に対して，あらゆる機器に移動可能な環境を整えることは現実的ではない．つまり，アプリケーションが実行可能な範囲は，移動可能な環境に制約を受ける．また，基盤環境に大きく依存するため，実装方法の多様性という面でも制約を生じさせる．

2.3 環境変化に対するアプローチ

ユビキタスコンピューティング環境に限らず，アプリケーションが環境変化に対応する場合，視点や手法におけるさまざまな相違が問題となる．つまり，目的の違いが問題を解決する際に視点の違いとして現れ，視点の違いが解決手法を選択する際に重要となる．本節では，用語の定義という意味も含め，これら環境変化に対するアプローチの違いを３つに分け整理して説明する．

2.3.1 機器指向・アプリケーション指向・タスク指向

環境変化に対して適応した動作をする場合，何を移動の対象として捕らえるかにより，どのような適応動作を選択するか方針が異なるものとなる．ユビキタスコンピューティング環境の場合に対象となるものは，機器・アプリケーション・人の３つである．ここでは例として，人がある作業をしている際，環境変化に関わらず作業を可能とするために適応するという場合について考える．また，ここで説明する例を図 2.3 に示す．ここでの課題は，異なる場所や環境に適応して同一の作業を続けることの実現である．

機器指向：機器に視点を置いた適応方法

まず１つの解決方法として，機器に視点を置いた適応方法がある．つまり，利用する機器そのものを常に利用者から切り放さず存在させることを前提とし，どのような場合においてもその機器を用いて同一の作業を実現できるようにするという方法である．この場合は，機器が人の移動にともなって移動できなければならず，また様々な状況に対して適応できる汎用性を有していなければならない．そのため，ノートパソコンや携帯電話などの小型機器を利用する場合に適している．また，通信機能においては，機器やそ

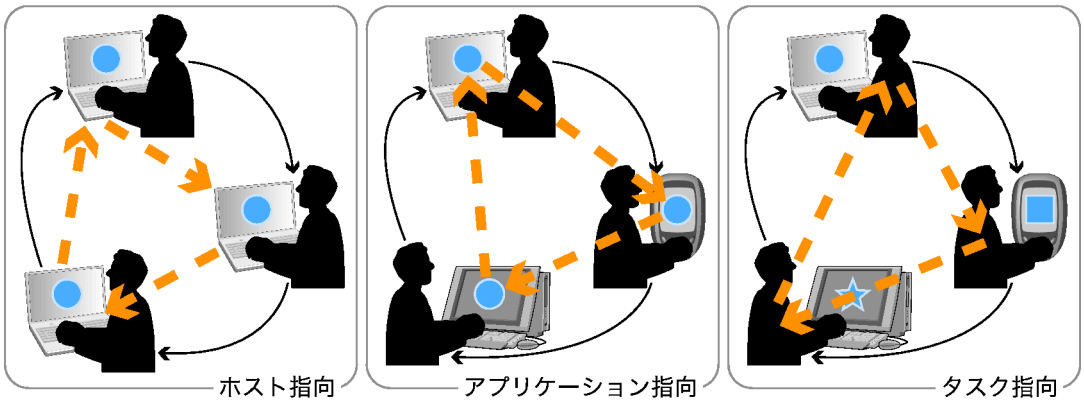

図 2.3: ホスト指向・アプリケーション指向・タスク指向

こで動作するアプリケーションに対する変更よりも，基盤となる機器自体で一括して変更する方が効率的であるため，通信の変化なども基盤となる通信機能の内部で隠蔽するという手法が適している．つまり，通信機能による移動透過性の実現が主な解決方法となる．この具体例として，アプリケーションに対し IP アドレスの変更を隠蔽する Mobile IP [21, 22] などが挙げられる．

アプリケーション指向：アプリケーションに視点を置いた適応方法

一方，もう 1 つの解決方法として，アプリケーションに視点を置いた適応方法がある．つまり，動作しているアプリケーションに視点を置き，利用者が常に同一のアプリケーションを利用することで同一の作業を実現するという方法である．この場合は，アプリケーションは利用者が用いるすべての機器で実行可能でなければならならい．そのため，すべての機器に同一のアプリケーションを予め用意しておくか，あらいは動作中のアプリケーションをそのまま移動できるようにする必要がある．一方，通信機能が提供する役割は 2 つある． 1 つは機器指向の場合と同様に，アプリケーションに対し移動により生じるアドレスの変更などを隠蔽する，移動透過性の実現である．またもう 1 つは，アプリケーションが移動するための通信路を提供することである．この具体例として，アプリケーション固有なデータの移動やアプリケーションの移動がある．前者は，フロッピーディスクや外部メモリなどに，特定のアプリケーション固有なデータを保存して異なる機器で利用可能とすることである．後者は，第 2.2.3 項で述べた，移動アプリケーションの例がある．

タスク指向：作業の内容自体に視点を置いた適応方法

最後の解決方法として，人の作業自体つまりタスクに視点を置いた適応方法がある．これは，機器やアプリケーションの実行状態よりも，利用者による作業の内容自体に視点を置き，様々な機器やアプリケーションによりその作業を実現できるようにするという方法である．この場合は，すべての機器やアプリケーションが，利用者の作業内容を

理解して実現できなければならない．そのため，特定の機器やアプリケーションには依存しない統一的な方法で，作業の内容が管理されている必要がある．一方，通信機能は，同一の機器か同種の機器のいずれかに対して新たに通信を確立した場合でも，利用者の作業には影響を与えないように動作する必要がある．この場合の例としても，２つのものがある．前者は，PDF[23] や XML データ [24] など汎用フォーマットのデータをフロッピーディスクや外部メモリなどに保存し，異なる機器やアプリケーションで利用可能とするものなどがある．後者は，IMAP[25] サーバなど利用者の作業内容をネットワーク上にある単一の機器に集約して保存し，異なる機器やアプリケーションからそれを操作するものなどがその例として挙げられる．

2.3.2 単純持続と状態継続

利用者の作業を環境変化に関わらず実現する場合，変化に対して何を維持するかという視点の違いが問題となる．特に利用者の作業に対して一貫性を実現する場合，その違いは単純持続と状態継続の違いとして現れる．単純持続とは，機器やアプリケーションが動作しているそのままの状態を維持するものである．これに対し，状態継続とは，利用者の作業内容を機器やアプリケーションに関わらず維持するものである．つまり，これらの視点をまとめると以下のように言える．

- 単純持続：機器やアプリケーションの動作をそのまま続けることに主眼を置く
- 状態継続：利用者の目的達成に対する支援を様々な方法で続けることに主眼を置く

前項で挙げた例で言えば，ホスト指向とアプリケーション指向は単純持続であり，タスク指向は状態継続となる．ここで注意すべきこととして，単純持続には複雑な適応処理が不要という意味ではなく，場合によりは状態継続の方が単純な動作で済む場合もあるということである．すなわち，単純持続において機器やアプリケーションに対して移動透過性を実現するためには，移動などにより生じる様々な環境の変化を隠蔽し，アプリケーションの動作に影響を生じさせないための複雑な処理が必要である．逆に単純持続の場合，サーバにすべての作業状態がアプリケーションから切り放されている場合は，アプリケーションやその基盤機構は状態の記述や管理方法は意識せず，極めて単純なコマンダやビューワとして動作すればよいだけである．

2.3.3 適応動作と機能拡張

適応動作と機能拡張とは，それぞれ以下に示す通りである．

- 適応動作：アプリケーションや機器において，その時点であるものが変わること
- 機能拡張：アプリケーションや機器において，その時点でないものが加わること

つまり，環境の変化などに対して，機器やアプリケーションが予め持つ機能により自らの動作を変化させ，より適した対応とすることを適応動作という．一方，機器やアプリケーションが予め持たない機能を追加し，まったく新たな機能により対応可能とすることを機能拡張という．また，これらをまとめて柔軟な動作という．

2.4 問題点と限界

本節では，これまでの節で述べた既存手法での問題点と限界について整理し，解決すべき課題を明らかにする．ユビキタスコンピューティング環境は，機器間の相互通信を前提とした異質かつ動的な分散環境である．また，第 2.1.1 項にも述べた通り，この環境で必要とされるアプリケーションでは，人や機器が置かれる状況やその変化に左右されずに，その人の目的を達成することができなければならない．さもなければ，異質で多様なユビキタスコンピューティング環境では動作できない．つまり，機器に機能を実装する際の自由度と通信機能の柔軟性が，異質で多様なユビキタスコンピューティング環境の基盤となる．これが実現されなければ，“いつでも・どこでも・だれでも”を前提とした機器の柔軟な利用環境を提供できない．このため，通信機能においてもその機能の構築方法や実行方法に対して柔軟性が求められる．同時に，目的や意図する動作を指定する場合の方法についても柔軟性が求められる．しかし，アプリケーションの通信機能はこれらの要求を実現する上で，いくつかの問題を抱えている．そこで本節では，アプリケーションの通信機能における柔軟性について，多様性への対応方法と挙動制御の指定方法という２つの面から解決すべき課題について論じる．

2.4.1 多様性への対応方法

様々な機器で分散して動作する通信アプリケーションにおいて，既存手法では通信機能が十分に機能分散して実装されていないという問題がある．このため，機能の構築方法や実行方法に対して柔軟性がない．この原因は，大別して２つに分けて考えられる．それは，既存の通信機能が“単純すぎること”と“複雑すぎること”である．

前者は，機器の通信方式や API など，提供される通信機能が単純な場合である．具体的には，第 2.2.1 項で説明したストリーム型の分散協調 API を用いた場合などが，この場合に該当する．基盤となる通信機能が単純な場合，複雑な通信方式や連携機能はアプリケーションが自ら実装する必要が生じる．このため，通信機能とアプリケーションが同一のものとして切り放せなくなってしまう．例えば，新たな通信方式への対応が必要な場合に，個別のアプリケーションなどで対応する実装を施す負担を強いられてしまう．

一方，アプリケーションから通信機能を完全に分離し，時にはその存在すら透過的にする高機能な通信ミドルウェアが存在する．具体的には，第 2.2.2 項に説明した手続き呼び出し型の分散協調 API を用いた場合などがこれに該当する．これらのように基盤となる通信機能が複雑な場合，移植性や性能などの問題からどのようなプラットホームでも動作するわけではない．また，通信するすべてのアプリケーションが同一の基盤ミドル

ウェアを用いることが前提となる．つまり，これら２点の面から，実行環境に制約を生じており，利用できる適応範囲を限定してしまう．このため，基盤ミドルウェアとアプリケーションを切り放して考えることができない．また，ここで挙げた例に限らず，高機能な通信機能ミドルウェアで広域ネットワーク環境でも動作しているものは存在しない．その理由として，複雑さがもたらすスケーラビリティの問題があることを指摘しておく．

このように，既存の分散アプリケーションでは，それぞれの機器で動作するアプリケーションにおいて，通信機能が十分に機能分散されているとは言えない．このため，同一性が前提とはできないユビキタスコンピューティング環境下において，多種多様な機器を連携するために必要となる通信機能の拡張性や適応性を実現することができない．これが実現できなければ，アプリケーションが可能な通信や連携の動作が，予め決められた範囲に限られてしまい，通信機能の柔軟性がなくなってしまう．そして，ユビキタスコンピューティング環境という様々な機器が存在する異質な環境下で，“いつでも・どこでも”十分に機能すると考えられる基盤環境を構築できない．

2.4.2 挙動制御の指定方法

人それぞれ利用形態のニーズは多様であり，予めすべてを想定して実装することは不可能と言って良い．一方，ユーザに機器の連携動作を強く意識させなければ利用することができないようでは，“だれでも”利用できるとは考えられない．しかし，既存の通信機能では，目的や意図する動作を指定する方法に柔軟性がない．

異質で多様なユビキタスコンピューティング環境では，通信アプリケーションにおける適応や拡張の要求も様々なものとなる．このため，具体的にどのような適応や拡張を実行するか，アプリケーションに対して指定する方法も柔軟なものでなければならない．しかし，既存の手法では，実装時や通信の開始時に接続先や適応方法を指定し，様々な変化を隠蔽した適応方法により単純持続するという方針をとる．これは，環境変化に対して，機器指向やアプリケーション指向による対応を図っているためである．このため，適応や拡張の指定方法が柔軟ではないと言える．

ユビキタスコンピューティング環境には様々な機器が存在し，様々な機器により利用者の本来の目的を実現する必要がある．そのため，本来はタスク指向の対応が適しているとも言え，状態継続による適応も考慮する必要がある．例えば，開発者や利用者の意図する挙動や目的を動的に指定し，それをもとにアプリケーションの適応や拡張を実行することが考えられる．つまり，すべてを予め実装せず，同時にすべてを利用者にまかせない手法が必要である．そして，その手法は特定の通信機能などに依存しないものとする必要がある．既存のアプリケーション構築手法では，通信アプリケーションに対して，このように柔軟な挙動の指定を可能とする方法は存在しない．

第3章

動的適応と機能拡張を支援するセッション機構

前章では，ユビキタスコンピューティング環境について説明し，そこで既存の手法を用いて分散通信アプリケーションを構築した際に生じる問題点を指摘した．本章では，はじめにそれらの問題を解決するための要求機能を整理する．続く節では，分散通信アプリケーションで必要とされる柔軟性について，実際に想定される例を用いて説明する．これにより，具体的に達成すべき動的適応と機能拡張の内容を整理する．そして次に，動的適応と機能拡張を支援するセッション機構が採用すべき方針と構成について考察する．また，通信機能に対する柔軟な挙動の指定方法について考察する．その後，本論文の関連研究について説明する．

3.1 要求機能の整理

前章では，ユビキタスコンピューティング環境において，既存の手法を用いて分散通信アプリケーションを構築する際に生じる問題点を指摘した．それは，多様性への対応方法と挙動制御の指定方法に柔軟性が欠如している故に生じる問題である．つまり，既存の通信アプリケーションでは，通信機能が十分に機能分散して実装されておらず，多くの場合でその実行環境に制約があり，利用できる環境や対応できる適応範囲が限定されてしまう．また，分散通信アプリケーションに対して利用者や開発者の目的と意図する動作を指定する方法に柔軟性がない．このように，通信機能の構築方法や実行方法に柔軟性がなため，“いつでも・どこでも・だれでも”利用可能な機器の柔軟な利用環境が構築できない．

ここで，これらの問題に対して，実現すべき課題を整理すると以下の通りとなる．

- 通信機能の構築方法や実行方法に柔軟性をもたせること
- 通信機能を機器やアプリケーションから機能分散して実装可能とすること
- 柔軟な挙動の指定を可能とする方法を確立すること
- “いつでも・どこでも”十分に機能する基盤環境として構築すること

これらの課題を達成するため，本論文では動的適応と機能拡張を支援するセッション機構による解決方法を提案する．本論文で提案するセッション機構は，通信アプリケーションと機器の通信機能を機能面から分離し，通信機能に対する柔軟な挙動制御を可能とする基盤環境である．

そして，この機構により機器の柔軟な利用環境を実現するシステムの構築を目指す．本章では，次節において支援すべき動的適応と機能拡張を具体的な例を示して明らかにする．その後，本論文で提案するセッション機構が採用すべき方針について考察する．

3.2 動的適応と機能拡張の支援

分散通信アプリケーションの動作を柔軟なものとするため，通信機能の構築方法や実行方法に柔軟性が必要となる．柔軟性とは，第 2.3.3 項で述べた通り，適応動作と機能拡張が可能なことである．つまり，通信アプリケーションが置かれた状況の変化などに対し，通信アプリケーションが予め持つ機能により自らの動作を変化させ，より適した対応とすることが適応動作である．一方，通信アプリケーションが予め持たない機能を追加し，まったく新たな機能により対応可能とすることが機能拡張である．ここでは，分散通信アプリケーションで実際に必要とされる柔軟性について，図 3.1 に示した例を用いて説明する．

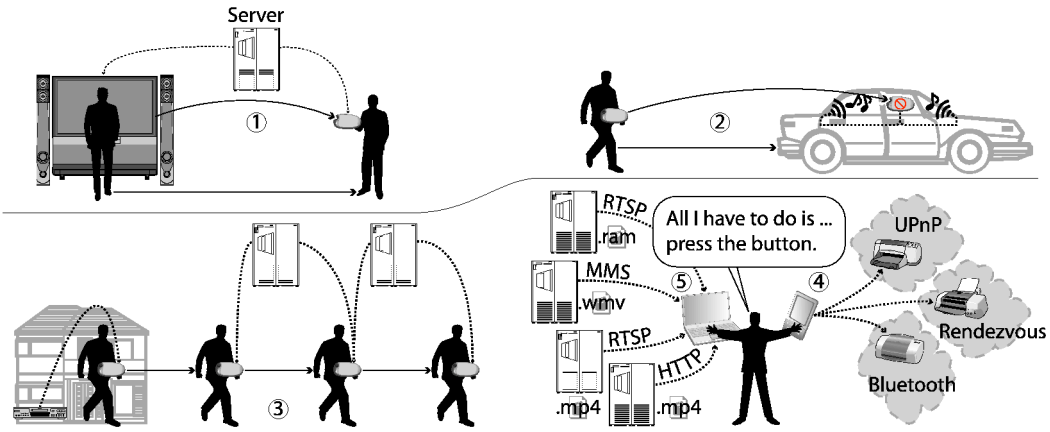

図 3.1: 動的適応と機能拡張の例

3.2.1 動的適応

動的適応の具体例として，図 3.1 では，以下の 3 つの場合を示した．

- 利用する機器やアプリケーションの変更（①）
- 機器やアプリケーションが実現する動作の変更（②）
- 機器やアプリケーションの通信機能における機能変化（③）

これらの例をもとに，通信機能で必要とされる動的な適応について考察する．

分散通信アプリケーションの通信状態を把握した動的適応

①の例では，利用者が部屋などに設置された据え置き型のディスプレイでストリーミング映像を見ている際に移動する必要が生じ，移動中もストリーミング映像の鑑賞を続けるため，利用する機器を利用者が携帯する機器などに変更する場合が想定される．ここでは，異なる機器やアプリケーションを用いてメディアフォーマットが異なる場合であっても，利用者の作業に一貫性を実現できなければならない．また，据え置き型のディスプレイが公共のものなど他の利用者と同時に利用していたものである場合，他の利用者の鑑賞作業を妨げてはならない．この場合に通信機能では，機器のアプリケーションがどのようなメディアのどの時点を再生しているか把握した上で，別の機器やアプリケーションにおいて，メディアフォーマットの違いに左右されずに，同一の時点から再生可能とするためのデータ転送を確立する必要がある．つまり，通信機能が通信プロトコルやその状態などを把握し，ストリーミングサーバなどの通信相手を変更した場合においても，利用者の作業を一貫させる通信の動的な適応処理を実行できる必要がある．

分散通信アプリケーションの動作変更と連動した動的適応

②の例では，ストリーミング映像を鑑賞中の利用者が，自動車の運転をする場合が想定される．ここでは，利用者はそれまで用いていたヘッドホンなどに代わり，自動車に設置されたスピーカを用いてストリーミングの音声を聴く必要があるため，音声出力先の機器を変更できなければならない．また，運転中は映像は不要であり，それにも関わらず映像データをストリーミングし続けることは，移動ネットワークの帯域を無意味に消費するなど無駄な動作となる．つまり，機器やアプリケーションは，場面に応じてその動作を変更できなければならない．この場合に通信機能では，機器やアプリケーションの動作変更に対して，音声出力のストリーミング先を変更したり，映像データのストリーミングを停止するなどの適応動作をする必要がある．つまり通信機能は，利用者の要求に基づいた機器やアプリケーションの動作変更に連動し，自身の動作を変更する通信の動的な適応処理を実行できる必要がある．

分散通信アプリケーションの動作状況や動作環境に対する変化に応じた動的適応

③の例では，ストリーミング映像を鑑賞中の利用者が移動中など場合に，場所や状況など場面の変化に応じて，通信処理を適したものとする場合が想定される．具体的には，まず，場所や状況などの場面に適したストリーミングサーバに変更することがある．また，サーバの変更前後やサーバに対する通信が不安定な場合に，複数のサーバに接続を多重化したり，通信プロトコルやバッファリングを調整するなどして対応することもある．この場合に通信機能では，機器やアプリケーションが置かれた状況や要求に基づき，通信機能を動的な変化させる必要がある．つまり通信機能は，機器やアプリケーションが置かれた状況の変化に対して，自身の動作を変更する通信の動的な適応処理を実行できる必要がある．

3.2.2 機能拡張

機能拡張の具体例として，図 3.1 では以下の２つの場合を示した．

- サービス検索プロトコルの多様性（④）
- 利用時の通信プロトコルの多様性（⑤）

これらの例をもとに，通信機能で必要とされる機能の拡張について考察する．

名前空間やサービス検索プロトコルに対する機能拡張

④の例では，機器がそれぞれ別のサービス検索プロトコルに限った対応であるために，それぞれの機器やアプリケーションが属する名前空間が異なる場合が想定される．ここでは，異なる名前空間に存在する異なるサービス検索プロトコルに従った機器は，協調して動作することができないばかりか互いの存在すら知ることができない．例えば，

Rendezvous 対応のオーディオプレーヤと Bluetooth 対応のヘッドセットでは機器やアプリケーション自体が持つ機能そのものでは連携することが可能であっても，サービス検索プロトコルの違いにより連携させて利用することができない．特に，機器を物理的に接続しない無線などを用いる場合は，このような違いを直感的に理解することは難しく，こうした問題が生じた際は利用者に対して大きな負担となる．この場合に通信機能では，機器やアプリケーションを，サービス検索プロトコルや名前空間から独立させて動作可能とする必要がある．つまり通信機能は，分散通信アプリケーションに対して，特定の名前空間やサービス検索プロトコルに依存しない相手先の検索方法や指定方法などを提供し，機器やアプリケーションの連携を拡張可能な通信基盤として存在する必要がある．

通信先の選択と通信プロトコルに対する機能拡張

⑤の例では，同一の作業を実現するための機器やアプリケーションに様々な種類が存在し，それに対応する通信プロトコルも複数ある場合が想定される．例えば，同じストリーミング映像の鑑賞という作業においても，それを実現するための機器やアプリケーションは多種多様である．それらの中で機器やアプリケーションが適するものや対応するものを利用者が選択することは大きな負担である．この場合に通信機能では，利用者からの明示的な指定がなくとも，機器やアプリケーションが対応するサーバなどを選択して接続する必要がある．同時に，新たな通信プロトコルの規格・標準などが出現した場合でも対応できる必要がある．さもなければ，進展し続ける通信プロトコルの改良や登場に対してその普及が遅れ，新たな通信サービスを提供する際の足枷ともなってしまう．つまり通信機能は，分散通信アプリケーションに対して，適した通信先サーバの選択や新たな通信プロトコルなどにおける拡張性を提供し，機器やアプリケーションの連携を拡張可能な通信基盤として存在する必要がある．

3.3 セッション機構

本論文では，第 3.1 節で述べた課題を解決するため，分散通信アプリケーションのためのセッション機構である u-session を提案する．本節では，u-session が採用すべき方針とその構成について考察する．u-session は，分散通信アプリケーションから通信機能を機能分散し，通信機能の挙動制御を柔軟な指定方法により可能とするセッション機構である．また，分散通信アプリケーションに対して，“いつでも・どこでも”十分に機能する基盤環境として構築する．これにより，前節で説明した通信機能における動的適応と機能拡張の支援し，機器の柔軟な利用環境を実現する．

セッション機構は，機器やアプリケーションに固有の機能を実行する部分とネットワーク依存の機能を処理する通信基盤の部分との間に存在する．そして，通信機能において前節で説明した動的適応や機能拡張を実現する．本論文で構築するセッション機構と分散通信アプリケーションなどの位置づけを整理するため，表 3.1 にその目的や機能をまとめる．

表 3.1: セッション機構の位置づけ

分類	目的	機能
分散通信アプリケーション	固有の機能を実行	決められた固有の機能を実装し，利用者の目的を実現 通信動作の開始や終了，セッション機構との連絡
セッション機構 (u-session)	柔軟な通信機能を提供	(分散通信アプリケーションの) 通信状態を把握した動的適応 動作変更と連動した動的適応 動作状況や動作環境に対する変化に応じた動的適応 名前空間やサービス検索プロトコルに対する機能拡張 通信先の選択と通信プロトコルに対する機能拡張
通信基盤	実際の通信処理を実行	機器やネットワーク依存の通信処理 転送データの保証やその際の再送処理などを含む

3.3.1 通信機能の分離

分散通信アプリケーションにおいて通信機能を機能分散する際，どの部分で機能を分散するか問題となる．分散通信アプリケーションに対して，セッション機構を“いつでも・どこでも”十分に機能する基盤環境として構築するためには，それが依存する基盤機構としても“いつでも・どこでも”利用可能な通信基盤を選択する必要がある．そこで，はじめに既存の通信アプリケーションの構成について分析する．

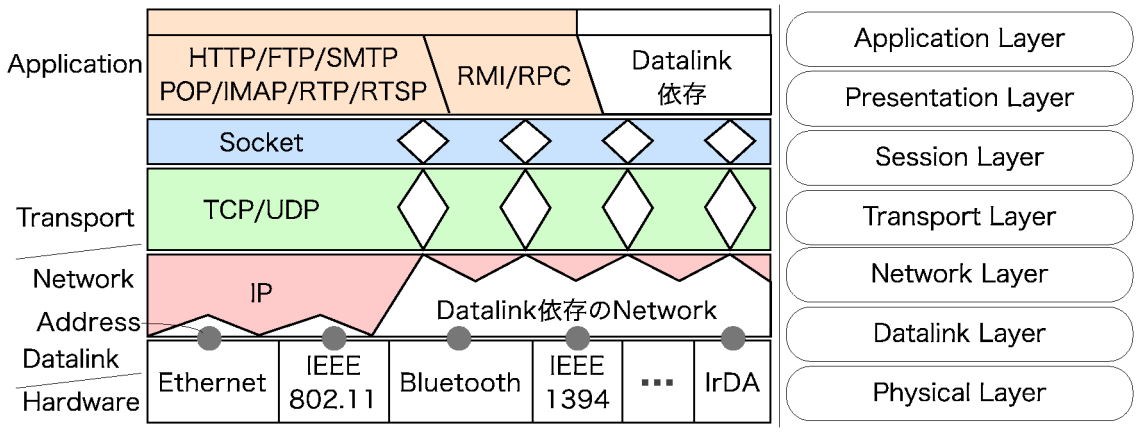

図 3.2: 既存の通信アプリケーションの構成と OSI 参照モデル

図 3.2 に既存の通信アプリケーションの構成と，通信機能の構築モデルである OSI 参照モデル [26] を示す．OSI 参照モデルは，多くの通信アプリケーションや通信プロトコルにおいて採用されている通信機能の構築モデルである．既存の通信アプリケーションでは，通信機能が機能ごとにまとめられたプロトコルスタックとして実装されることが多く，大半の部分で OSI 参照モデルの考え方を取り入れている．例えば，Ethernet[27] など

の物理的なデバイスの動作は，物理層やデータリンク層に分類され，TCP/IP[28, 29, 30]はネットワーク層とトランスポート層に分類される．

一方，既存の通信アプリケーションでは，セッション層やプレゼンテーション層は個別のアプリケーションかライブラリなどとして実装されており，独立して実装されることは少ない．このことが，通信アプリケーションから通信機能が機能分散されない原因でもある．そして，通信アプリケーションが，本来は機器のネットワーク上での識別子である物理的なアドレスなど，通信基盤内の情報を管理することが必須となっている．その上，自らと通信相手との間の通信処理も，すべてアプリケーションで実行しなければならない．このため，通信機能の処理と密接に動作することが要求される．

より既存のアプリケーションでは，アプリケーション層とトランスポート層の間に存在するものは，アプリケーションがトランスポート層以下の通信機能を制御するための分散協調 API となる．このため，セッション機構は図 3.3 に示す通り，分散協調 API の部分で実装することが望ましい．ここで考慮すべきことは，セッション機構が“いつでも・どこでも”十分に動作可能なものである必要があることであり，そのためそれが依存する分散協調 API でこの課題を達成することが求められる点である．

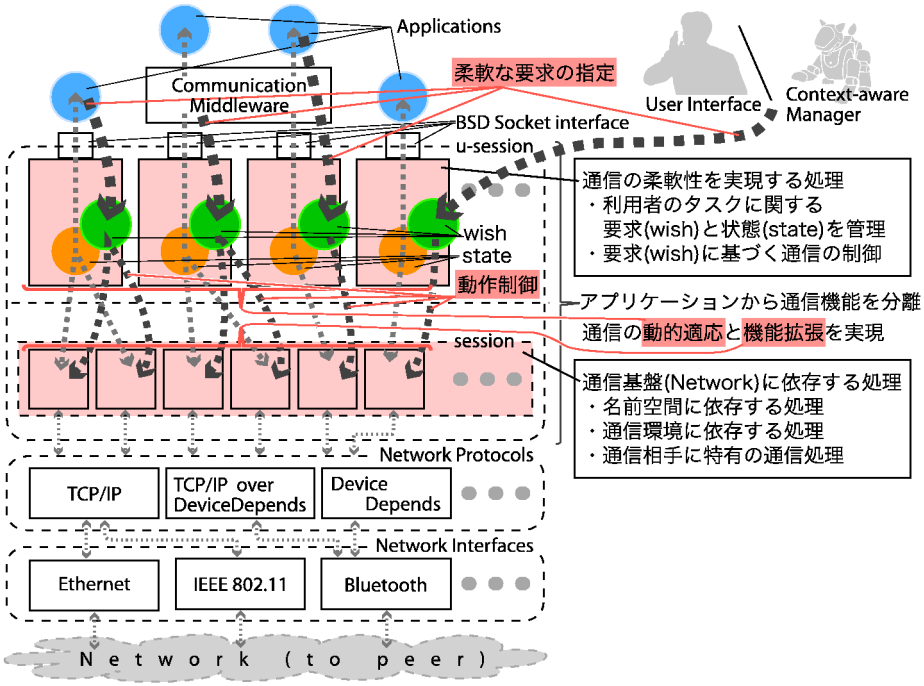

図 3.3: u-session の概要

第 2.2.1 節で述べた通り，多くの分散協調 API が BSD Socket インタフェースをその基盤環境として用いている．つまり，BSD Socket インタフェースの内部で通信機能の機能分散を実現すれば，それを用いる他の分散協調 API においても実現することが可能となる．また，適応処理を BSD Socket インタフェースの内部で隠蔽して透過的に動作可能とすれば，既存の通信アプリケーションを改変せずとも利用することができるようになる．この点で，BSD Socket インタフェースはプラットホーム低依存であると言え，BSD

Socket インタフェースの内部にセッション機構を実現することが望ましい．その上，BSD Socket インタフェースを用いることにより，物理的な機器に依存する通信機能もデバイスドライバの改変による [31][32] 対応手法を用いて，汎用的なストリーム型の通信機器として様々なアプリケーションから利用することも可能となる．

3.3.2 柔軟性の実現

u-session の内部は，機能面から見て大きく 3 つの部分に分けて構成される．それは，通信の柔軟性を実現する処理を実行する部分，通信基盤に依存する処理を実行する部分，そして，BSD Socket インタフェース互換機能を提供する部分である．

通信の柔軟性を実現する処理を実行する部分

まず，通信の柔軟性を実現する処理を実行する部分について説明する．通信機能がセッション機構として分離されると，図 3.3 に示した通り，この部分において通信機能の動的適応や機能拡張を実現することが可能となる．しかし，本来の分散通信アプリケーションは，通信機能を用いて緊密に連携して動作するものである．このため，機能分散した結果としてアプリケーションから通信機能が制御不能となれば，分散通信アプリケーションの連携した動作に支障を来たす結果となる．つまりセッション機構では，通信基盤に依存せずに通信機能の動作を制御するための方法をアプリケーションに対して提供する必要がある．

一方，第 2.4.2 節において述べた通り，利用者のニーズは多様であり，アプリケーションに予めすべての動作制御を実装することは不可能である．そのため，利用者からの明示的な指定や，機器と利用者の現状をもとに実行すべき動作を判断する状況認識機構から，セッション機構における通信機能の動作制御を可能とする必要がある．このため u-session では，アプリケーション開発者・利用者・状況認識機構などからの要求を Wish として管理し，Wish に基づき通信の相手先を特定し，通信機能の動的な動作制御を実行する．また，Wish を指定する際の手がかりとなる情報を上で挙げたものに対して提供するため，u-session の内部でアプリケーションの通信状態を管理し，必要に応じてその通知を可能とする．このような動的かつ多方面からの挙動指定と通信状態の取り扱いにより，通信基盤に依存せずに通信機能の動作を制御する方法を実現する．つまりこの部分は，通信状態を管理し，アプリケーションの動作変更との連動や動作状況や動作環境の変化に応じた機能の動的な入れ替えを制御する部分である．

通信基盤に依存する処理を実行する部分

次に，通信基盤に依存する処理を実行する部分について説明する．この部分は，上に述べた部分からの制御により，実際に機能の動的な入れ替えを実行する部分である．ここでは，サービス検索プロトコルや名前空間，通信プロトコルなどに依存した処理をそれぞれの機能ごとに分離された session と呼ぶモジュールを用いて実現する．session は，

通信機能を制御する部分からの動作制御に応じて処理を開始する．そして，通信アプリケーションが利用する BSD Socket インタフェース 1 つに対して 1 つ以上の session が存在し，対応づけられた session が実際の通信処理を実行する．

session では，動的な通信機能の変更が可能である．例えば，通信する際のバッファリングの有無や大きさの変更などを実行できる．また，session の入れ替えにより，通信機能の拡張を可能とする．具体的には，新たなサービス検索プロトコルや名前空間の対応と通信プロトコル拡張などを実行する．このように，Wish に基づいた通信機能に対する適応の処理は，session を変化させるか別の機能を有する session に切り替えることで実行される．また，通信を多重化する場合などは，同一の session を並列に存在させ，制御する部分がそれらを並列に用いる．

BSD Socket インタフェース互換機能を提供する部分

最後の部分である BSD Socket インタフェース互換機能では，分散通信アプリケーションに対して，上に述べた 2 つの機能を BSD Socket インタフェース互換のインタフェースを用いて提供する．また，他の通信 API から呼び出される場合についても考慮しなければならない．

3.3.3 分散通信アプリケーションにおける利点

u-session により通信機能の処理がアプリケーションから分離されると，アプリケーションはそれに固有の機能を実行することに特化し，通信機能に対する最小限の制御や連絡を実行するだけで他のアプリケーションと連携することが可能となる．また，u-session による多方面からの Wish 管理により，アプリケーションの動作を適応させて動作するために必要である，複雑な状況認識や適応処理をはじめ，処理変更を選択するためのユーザインタフェースの実装等が個別のアプリケーションにおいては不要となる．

また，こうした利点は新規のアプリケーションを構築する際に限らない．既存のアプリケーションに対して u-session を用いるための名前空間を提供すれば，多くの場合において既存のアプリケーションに対する変更なしに本機構の機能を利用することが可能となる．また，既存の通信アプリケーションで通信機能のみを拡張したい場合でも，session モジュールにおいて拡張するのみで実現が可能となる．

3.4 柔軟な挙動の指定方法

u-session では，Wish を用いて通信相手先の指定や通信機能の適応方法を決定する．このため，通信アプリケーションの連携を柔軟とするためには，Wish による柔軟な挙動の指定を可能とする方法を確立することが必要である．

3.4.1 Wish による適応と拡張の指定

図 3.4 に，本論文で具体的に想定する Wish の例を示す．

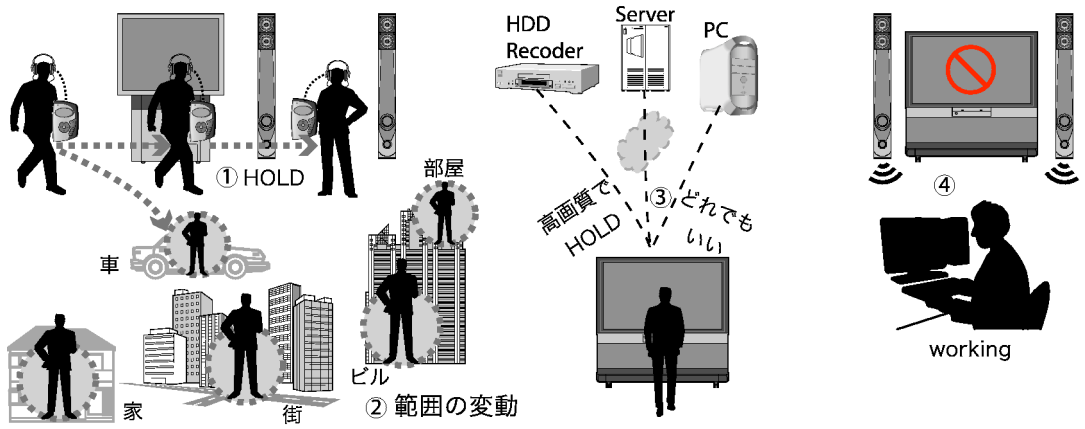

図 3.4: 動的適応の例

①の例では，ヘッドセットと携帯型メディアプレーヤにより音楽鑑賞をしながら移動する利用者を想定した場合である．この場合，利用者は“ヘッドセットと携帯型メディアプレーヤによる音楽鑑賞”を続けたいという Wish を持つ．そのため，近くにディスプレイが現れた場合でも，部屋のスピーカが利用可能となった場合でも，それらの機器に変更せず動作する必要がある．

②の例は，“近くにあるスピーカを用いて音楽を聴く”という Wish を持つ利用者を想定した場合である．図に示した通り，利用者が移動に伴い“近く”の定義は変動する．利用者が自宅にいる際は，それが自宅内となるかもしれず，街中にいる場合には周囲の区画となるかもしれない．一方，自動車に乗車した場合は，その範囲が車内に限定されなければ，不都合となる．また，携帯機器の能力やバッテリ容量などにより，範囲の変動が必要となることも考えられる．このように，利用者の現状にあわせて定義を変えない場合，検索の即応性や動作効率に対して悪影響を与えてしまう．

③の例は，利用者がある映画を見たいという場合である．この場合，利用者の目的は“ある映画の鑑賞”であり，それが保存されている機器などを意識しなくてもよければ利用者に与える負担は少なくなる．また，それがネットワーク経由のサーバにあるものか自宅のどの機器に保存されているかなどを利用者が自ら把握しなければならない場合，利用者が理解できる範囲を越えたサービスを提供することができない．しかし，同時に“高画質で鑑賞したい”という Wish も持つ場合は，これを優先する必要がある．

④の例は，仕事中などの利用者が“ビデオストリーミングの音声だけを聴きたい”という場合である．この場合，ストリーミング映像が表示されていても鑑賞しないため，ディスプレイを消して無駄なエネルギーの浪費を避けたいという Wish がある．

このように，通信機能の適応と拡張に対し，Wish を用いて“してほしいこと”と“してほしくないこと”を指定できることが，柔軟な動作を可能とするために必要である．

3.4.2 Wish の評価選択機構

u-session では，Wish に基づき通信機能の動的適応や機能拡張を実行する．この際，Wish を実際の指示に変換できなければ，挙動制御が実現できない．つまり，Wish として指定された目的を何らかの方法で評価し，適切な動作を選択する機構が必要である．既存のプログラミング手法では，このような評価選択機構として図 3.2 に示すものがある．

表 3.2: 評価選択機構の分類

方式	例	機能
手動	人手による選択	随時，利用者が選択・指示・実行を決定する．
比較推論 (決定的)	プロダクションシステム (OPS5)	事実から if-then 形式のルールにより決定する．
比較推論 (消去法)	制約論理プログラミング (CLP)	事実を制約条件に基づく絞り込みで評価して決定する．
確立推論	ベイジアン・ネットワーク	事実関係における確立の相互作用を集計して決定する．

このように評価選択の手法は，既存にも様々な方式から研究されている．そして，本論文の目的は，通信アプリケーションから通信機能を分離するためのセッション機構を実現することであり，自動化された適応システムの構築ではないため，これらの内容については本論文で扱う対象から逸脱する．このため，本機構の利用者がこれらのものから用途に応じて適したものを選択して利用できるようにすることが，最も理にかなうと考えられる．また，本論文では完全な自動処理は難しいという立場から，利用者から明示的に指定する方法を排除しない．このため，本論文では評価選択機構そのものは研究の対象外とし，u-session の外部で動作する評価選択機構を利用可能とするためのインタフェースを提供するまでにとどめる．そして，u-session の機構自体では，特定の評価選択機構に依存しない柔軟な動作の実現を目指す．

3.5 関連研究

OSI 参照モデルを起点として，セッション機構の研究は数多くなされている．本節では，それらの研究の中で，分散通信環境やユビキタスコンピューティング環境を対象とした適応型のセッション機構について，本論文で提案する機構との相違を含めて説明する．ただし，それぞれの研究で提案するセッション機構の定義は，その目的や用途などに応じ若干の相違があるため注意する必要がある．

本節では，次に示す 4 つの関連研究に関して説明する．それは，通信コネクションの切断に対し透過的な再接続を実現する Rocks and racks，アプリケーションに関連する通

信コネクションの移動を実現する MobileSocket，利用者のタスクを中心とした作業の移動を実現する SinkProxy，そして，柔軟なサービス検索を目的とする SoNS である．

3.5.1 Rocks and racks

Rocks と Racks[33] は，移動する機器で動作する通信アプリケーションに対し，アドレスの変更やネットワークの障害による通信コネクションの切断を隠蔽するセッション機構である．例えば，あるサーバとの通信中に上記の障害が原因で接続が切れた場合，アプリケーションがそれを知らずに動作できるように再接続する．つまり，ここでのセッションは “明示的な指定なしに決して切断されないコネクション” である．

Rocks は，アプリケーションの通信を中継する通信ライブラリとして，通信アプリケーションと BSD Socket の間に介在する．通常，アドレスの変更やネットワークの障害により通信コネクションの切断が生じると，通信機能から BSD Socket を経由してアプリケーションに対し通信の終了が通知される．Rocks では，通信コネクションが上記の理由で切断されると，この終了通知を無効とすると同時にアプリケーションからの通信動作を一時停止する．そして，Rocks が通信コネクションの再接続を試みる．その後，再接続が成功すると，dup システムコールを用いて新たに確立された通信コネクションの識別子をアプリケーションが使用中の識別子に対応づける．また，ダイナミックライブラリのプリローディング機能を用いており，ほとんどの場合で既存アプリケーションの再コンパイル等は不要である．一方，Racks はアプリケーションとサーバの間に介在するプロクシサーバとして動作し，Rocks の場合と同様の隠蔽処理を実現するものである．また，これらの機構では通信内容には触れないため，協調動作する分散通信アプリケーションの場合は，通信先と自らの双方でこれらの機構が利用可能である必要がある．

3.5.2 MobileSocket

MobileSocket[34] は，移動アプリケーションに対する通信コネクションの維持を実現するセッション機構である．例えば，協調動作する移動アプリケーションに対して，動作する機器の変更などにより生じる通信コネクションの変化などを隠蔽し，持続した通信を可能とする．つまり，ここでのセッションは “移動アプリケーションのための切断されないコネクション” である．

MobileSocket は，Java による BSD Socket の拡張ライブラリであり，アプリケーションが利用する BSD Socket の状態を MobileSocket のオブジェクトとして管理し，アプリケーションに伴って移動することを可能としている．そして，MobileSocket の内部では，アプリケーションに対して提供した識別子と機器の通信基盤を経由して実際に確立した通信コネクションに対する識別子との対応づけを調整する．これにより，機器の変更などに影響されない通信の維持を保証することが可能となる．しかし，この機構のみでは複数のアプリケーションが同時に移動する場合には対応出来ず，また Java 以外のアプリケーションで動作することも出来ない．

3.5.3 SinkProxy

Stream Redirection Architecture[35] は，データ送信サーバ “server” とデータ受信シンク “sink” の間にシンクプロキシ “sink proxy” という機構を介在させ，利用者の移動に対してデータ転送の持続性を実現するものである．機器を利用する場合には，利用者が携帯するコントローラ “Controller” を通じてシンクプロキシに対してコマンドを発行し，シンクプロキシにより全ての通信動作が実行される．ここで，シンクプロキシは各環境毎に用意されていることが想定され，サーバやシンクと同様に移動しない．つまり，ここでのセッションは “コントローラから指令される利用者の作業” である．

利用者が別環境に移動した場合，コントローラが移動を検知して新たに利用可能なシンクプロキシを発見する．ここで，利用していたシンクプロキシでの通信を中断し，接続状態を格納するオブジェクトを同シンクプロキシからコントローラに転送する．そして，別環境で引き続いてデータ転送を行う目的で，コントローラは新たなシンクプロキシに先ほど受信したオブジェクトを転送して接続状態を復元する．こうして，シンクプロキシは同一のサーバを用いて同一なバイトオフセット[1]からデータ転送を持続して行うことが可能となる．しかし，Java という単一の実行環境を用いており，持続されるデータ転送は送受信されるデータの内容や形式に強く依存する．同時に，サーバやシンクが常に固定機器という前提や，同一のデータ送信サーバを用い続けることが，通信の適応手法から柔軟性を失わせる．つまり，移動に対して適応的な動作を実現することは出来ていない．

3.5.4 SoNS

Service oriented Network Socket[36] は，制約論理により，様々なサービス検索方式から目的とする通信相手を特定するためのサービス検索手法を提供する．この機構を用いる場合，アプリケーションの開発者は利用したいサービスの特性をプログラムに記述しておくだけで，SoNS がそれに合わせたサービスを検索して接続する．例えば，通信相手を特定のホスト名ではなく，“大きさ 15 インチ以上で利用されていないディスプレイ” などと指定すれば良い．つまり，ここでのセッションは “サービス特性に応じたサービス利用の支援” である．

[1] データ転送開始時から転送されたバイト数．

3.5.5 関連研究の相違

最後に，ここで説明した関連研究の相違を表 3.3 にまとめる．

表 3.3: 関連研究の相違

名称	実現するもの	視点	動作の方針
Rocks and racks	切断の回復と透過性	ホスト指向	単純持続
MobileSocket	接続の移動透過性	アプリケーション指向	単純持続
SinkProxy	タスク動作を持続した移動	タスク指向	状態継続
SoNS	柔軟なサービス検索	タスク指向	単純持続

Rocks と Racks では，機器の移動により生じる通信機能の変化を隠蔽することが目的であるため，ホスト指向の移動透過性を実現するものであると言える．また，アプリケーションに対して透過的に動作するために，すべての処理を隠蔽する単純持続型の機構である．

MobileSocket では，アプリケーションの移動により生じる通信機能の変化を隠蔽することが目的であるため，アプリケーション指向の移動透過性を実現するものであると言える．また，アプリケーションに対してすべての処理を隠蔽し透過的に動作するため，単純持続型の機構である．

SinkProxy では，状況に応じて通信相手を変更して動作するため，タスク指向の移動透過性を実現するものであると言える．また，同様の理由で状態継続型の機構である．

SoNS は，柔軟なサービス検索を目的とするためタスク指向のサービス検索を提供するものであるといえる．しかし，アプリケーションに対する適応処理においては，異質性を前提とはせず同種のサービスを用いる場合を想定する．このため，単純持続型の機構に分類される．

3.6 まとめ

本章では，通信アプリケーションと機器の通信機能を機能面から分離し，通信機能に対する柔軟な挙動制御を可能とする基盤環境として，動的適応と機能拡張を支援するセッション機構による解決方法を提案した．はじめに，従来の分散通信アプリケーション構築手法では実現できない実現すべき課題を整理した．そして，ユビキタスコンピューティング環境の分散通信アプリケーションにおいて支援すべき動的適応と機能拡張を具体的な例を示して明らかにした．次に，これらの課題を実現するセッション機構として u-session を提案し，その構成などについて考察した．最後に関連研究について述べ，いずれの場合でも本研究で対象とする問題を根本的には解決できないことを示した．次章では，本章で提案した u-session について，これを実現するための具体的な設計と実装について述べる．

第4章

設計

本章では，分散通信アプリケーションに対して動的適応と機能拡張を支援するセッション機構である **u-session** の設計について述べる．**u-session** では，通信アプリケーションと機器の通信機能を機能面から分離し，通信機能に対する柔軟な挙動制御を可能とする．まずはじめに，**u-session** の全体構成と動作手順の概略について述べ，続いて個別の機構ごとに説明する．

4.1 概要

本節では，前章で提案した分散通信アプリケーションに対して動的適応と機能拡張を支援するセッション機構である u-session の概要について述べる．はじめに設計方針を整理し，それをもとに全体構成と動作の概要について説明する．そして次節以降では，ここで述べた機構の各部分について詳しく説明する．

4.1.1 設計方針

本研究の目的は，分散通信アプリケーションに対して柔軟な連携を可能とするため，通信機能の動的適応と機能拡張を支援する基盤環境を提供することである．この目的を実現するため，u-session システムを設計・実装する．u-session システムの設計方針として５点を以下にまとめる．

アプリケーションから個別の通信機能に固有な処理や情報を分離すること

分散通信アプリケーションの柔軟な連携を可能とするためには，柔軟な動的適応を可能とし様々な適応を可能とするための機能拡張を実現できなければならない．こうした要求を通信アプリケーションが個別で対応しなければならない場合，アプリケーションそれぞれで対応するコストが無視できない．そのため，通信機能における動的適応や機能拡張は，通信アプリケーションから独立して可能とすることが望ましい．通信アプリケーションから分離する通信機能は，通信プロトコルに依存する処理やサービス検索プロトコルに依存する名前空間などである．これにより，多種多様な通信環境において，分散通信アプリケーションが連携して動作することを可能とする．

分離した機能が制御不能とならないこと

既存の分散通信アプリケーションにおいてアプリケーション間の連携は，通信機能の処理などと大きく結び付いて動作している．そのため，通信機能を分離した結果，通信アプリケーションから通信機能が制御不能となれば，連係動作が不可能となることは自明である．つまり，通信機能を通信アプリケーションから分離した場合でも，通信アプリケーションや利用者などからの要求や指示に基づいて通信機能が動作可能である必要がある．これを実現するため，アプリケーションの開発者や利用者などの要求や指示を本機構が受取り，それに基づいて処理を実行する．これにより，通信アプリケーションから分離された通信機能に対する制御を可能とする．

通信機能が拡張可能であること

既存の通信アプリケーションは通信機能と分離されていないため，通信プロトコルなどの変化に対して寛容ではない．また，複雑な通信プロトコルの場合は，その実装に必

須の部分とそうでない部分を設けるため，同一の通信プロトコルであっても必ずしも協調して動作可能とはならない．また，サービス検索プロトコルや名前空間は，それが対象とする目的に応じて，その範囲や粒度や検索頻度などが違るため，安易に統一することはできない．このような通信機能の性質に対応するためには，通信機能がその機能を拡張可能である必要がある．つまり，個別の通信方式に依存するプロトコルやそれに関する情報は，通信機能の本体と分離して動作させたほうがよい．

特定のプログラミング言語や実行環境に依存しないこと

特定のプログラミング言語や実行環境に依存した設計では，“いつでも・どこでも”動作できる機構とは言えない．そのため，セッション機構がより多くの環境で動作し，他の分散通信ミドルウェアなどを用いた場合でも動作可能であることが望ましい．

BSD Socket インタフェースに完全互換であること

u-session では，通信アプリケーションに対して，BSD Socket インタフェース互換の通信 API を提供する．この際，従来の通信アプリケーションがそのまま動作できるように，既存の BSD Socket インタフェースをそのままの形で利用できる必要がある．つまり，既存のプログラミング手法を変更せずに通信機能の分離が可能であることが望ましい．

4.1.2 全体構成

前項の設計方針に基づき，u-session の全体構成と動作手順について述べる．本機構は，図 **??**に示す通り，４つの部分から構成される．

通信基盤依存処理部

通信基盤依存処理部は，通信プロトコルなどに依存した通信機能を実際に処理する部分である．ここでは，通信プロトコルごとのコネクションなどを抽象化し，動作制御部からの制御を可能とする．実際の通信相手との間で処理すべき機能や管理すべき情報は，すべてこの部分により扱われる．

動作制御部

動作制御部は，通信アプリケーションの開発者や利用者の目的を達成するために，通信基盤依存処理部に実装された通信プロトコルなどに依存した個別の通信機能を制御する．同時に，個別の通信機能に依存しない適応動作や機能拡張はここで実現する．

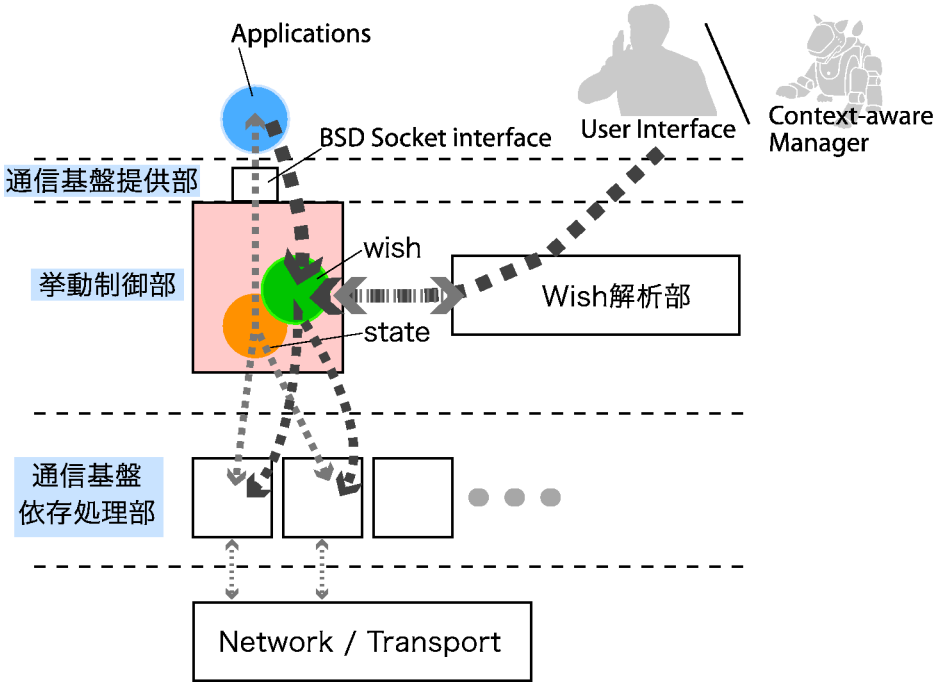

図 4.1: u-session の全体構成

Wish 解析部と Wish

Wish とは，第 **??**項に示した通り，通信機能の適応と拡張に対し“してほしいこと”と“してほしくないこと”を指定するための情報である．Wish 解析部は，通信アプリケーションの開発者や利用者などからこの wish を受取り管理する．そして，wish に基づく通信機能の適応処理や機能拡張を実行する．

通信基盤提供部

通信基盤提供部は，通信機能を通信アプリケーション分離するための分散通信アプリケーションに対する通信基盤として，u-session と分散通信アプリケーションを連絡する部分として動作する．また，通信アプリケーションに対して，この部分により BSD Socket インタフェース互換の通信 API を提供する．

4.1.3 動作概要

本項では，前項で述べた機構の動作概要について説明する．動作概要を図 4.2 に示す．

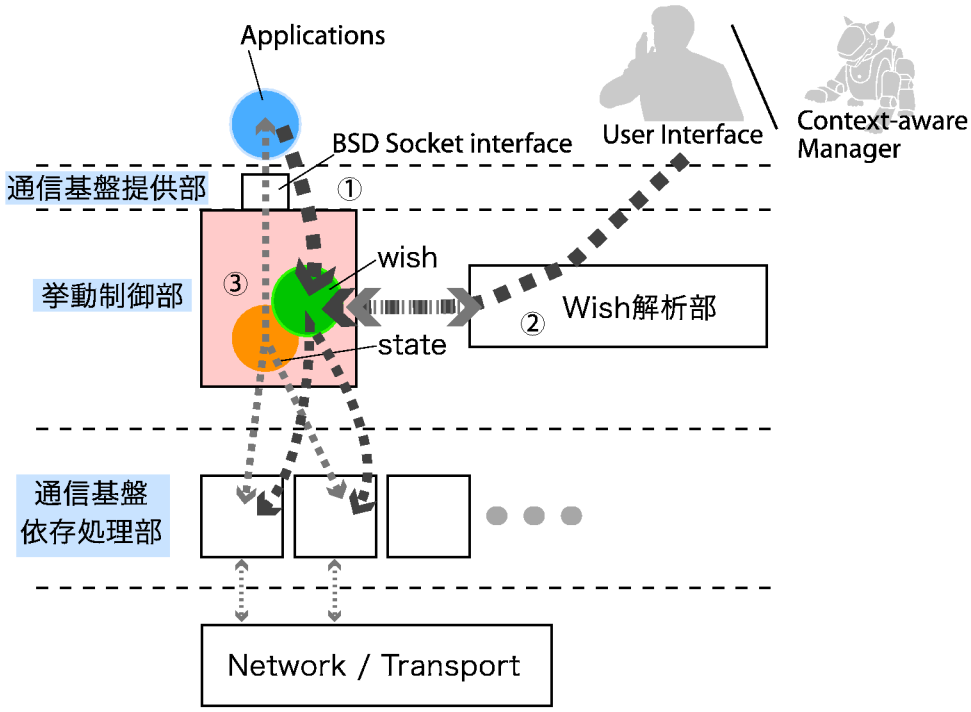

図 4.2: u-session の動作概要

通信開始まで

1. アプリケーションが通信を開始する際，開発者の Wish が通信基盤提供部を経由して u-session に指定される．すると u-session は，Wish 解析部に Wish を伝える．(必要な場合)Wish 解析部が，機器やサービスの提供者，および利用者からの Wish を受け取る．

2. Wish 解析部が Wish を解析し，具体的な通信の相手先と通信の手法などを特定し，u-session にそれらの情報を設定する．

3. u-session は，受け取った情報に基づき，通信基盤依存処理部を構成する．

通信中の適応と拡張

1. Wish の変更は，利用者などからは Wish 解析部経由で，アプリケーションからは通信基盤提供部を経由して，随時変更することが可能である．

2. Wish が変更されると，再び Wish 解析部においてそれが解析され，それに基づき動作制御部が処理を実行する．

4.2 通信基盤依存処理部

通信基盤依存処理部は，特定のサービス検索プロトコルや名前空間に依存する情報を隠蔽して処理し，個別の通信プロトコルに関わる処理を透過的に実行する．機能拡張を容易とするため，すべての通信基盤依存処理部モジュールは機能ごとに分けられた統一的な構成とする．また，通信処理の汎用部分と個別の仕様に依存する部分を分けて実装可能とすることが望ましい．

4.2.1 通信基盤依存処理部の機能

通信基盤依存処理部では，図 4.1 に示す機能を提供する．

表 4.1: 通信基盤依存処理部の機能

機能名	実行する機能
開始	通信基盤依存処理部の開始，初期化処理
終了	通信基盤依存処理部の終了，データ領域の解放等
接続	通信相手と接続する
切断	通信相手と切断する
受信	受信処理の実行
送信	送信処理の実行

通信基盤依存処理部は，Wish 解析部からの指示により，動作制御部から呼び出されて実行を開始する．各通信基盤依存処理部モジュールは，自身の開始機能が呼び出された際，管理すべき情報のデータ構造等を作成し，初期化処理を実行する．この後，接続処理を実行するが，実際の接続処理を実行する前に，通信基盤依存処理部が通信相手先の名前から実際のアドレス等に解決する必要がある．そのため，接続処理では，機器やアプリケーションの名前から実際のアドレスを解決するなどの前処理を実行した後，実際の接続処理が実行される．また，接続が確立された直後は，認証や通信相手との間での調整などの後処理を実行する必要がある．そして，接続機能の実行が完了した際は，受信機能と送信機能により通信相手との送受信を可能とする．

4.2.2 通信基盤依存処理部の拡張性

通信機能の機能拡張を柔軟に動作させるため，通信基盤依存処理部は組み合わせて動作可能とする．これにより，図 4.3 に示すような通信基盤依存処理部の拡張性を実現する．まず，標準送受信モジュールは，通信相手先との通信コネクションを抽象化し，上位のモジュールからの送受信と通信相手先とを連絡する．次に，プロトコルモジュールは，個別の通信プロトコルなどに依存した処理と情報の管理を実行する．ここでは，HTTP や RTSP などのアプリケーションレベルでのプロトコルを実行する．そして，補助送受

信モジュールは，送受信データのバッファリングやアプリケーションレベルにおける送受信データの一貫性保証や再転送などの機能を実行する．

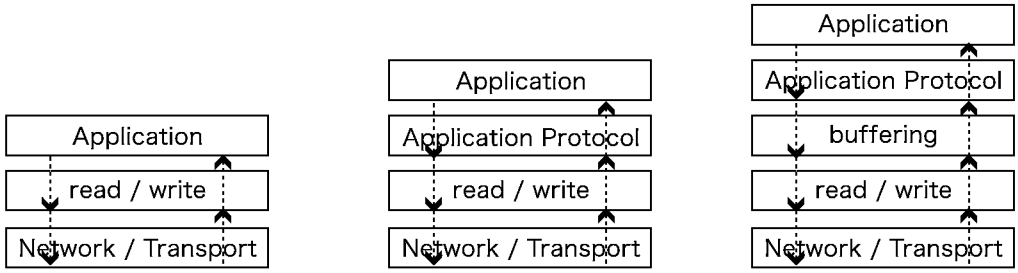

図 4.3: 通信基盤依存処理部における拡張性

4.3 動作制御部

動作制御部は，通信基盤依存処理部において抽象化された通信機能を制御し，適応処理や拡張処理の実行を制御する．ここでは，アプリケーションや利用者により指定された Wish の達成を目的とし，Wish 解析部で具体化された方針に基づき，通信基盤依存処理部の制御や切り替えを実行する．また，この部分よりもアプリケーションよりの部分では，通信基盤として動作するためのインタフェースを提供することにとどめる．このため，他の部分で扱うことができないすべての通信機能は，この部分で処理する．

4.3.1 動作制御部の機能

動作制御部では，図 4.2 に示す機能を提供する．

表 4.2: 動作制御部の機能

機能名	実行する機能
開始	動作制御部の開始，初期化処理
終了	動作制御部の終了，データ領域の解放等
接続	通信基盤依存処理部に接続を指示する
切断	通信基盤依存処理部に切断を指示する
受信	通信基盤依存処理部の受信処理を実行する
送信	通信基盤依存処理部の送信処理を実行する
Wish 更新	Wish を更新する
状態取得	通信状態を通知する
適応	適応処理を実行する

動作制御部の開始と終了は，通信基盤依存処理部の場合と同様にデータ領域の確保や

初期化処理とその解放処理である．これらの処理は，通信基盤提供部を通じて通信アプリケーションから呼び出されて実行される．アプリケーションから接続要求があると，動作制御部が管理するアプリケーションからのら Wish を Wish 解析部に渡し，その解析結果を得る．これに基づき，対応する通信基盤依存処理部が開始され，通信データの送受信が可能となる．また，Wish 更新機能は，アプリケーションか Wish 解析部を通して利用者のユーザインタフェースなどから，Wish が変更されたときに呼び出され，管理する通信機能に対する Wish を更新する．状態取得は，Wish 指定時の参考とするため，アプリケーションや利用者のユーザインタフェースなどに対して通知可能とする機能である．そして，適応機能は明示的に適応処理や拡張処理を実行する必要がある場合に，アプリケーションや Wish 解析機構から実行される．

4.3.2 制御方法

動作制御部が通信基盤依存処理部を制御する際の方法は，３種類ある．まず，通信基盤依存処理部の機能を直接呼び出すものである．また，通信基盤依存処理部において呼び出す機能を変更するものである．そして，通信基盤依存処理部の属性を変更するものである．これらの方法を用いて，通信基盤依存処理部の機能変更や拡張，および動作中の機能の挙動変更などを実現する．

4.4 Wish と Wish 解析部

u-session では，アプリケーションの通信目的を指定し，その目的に基づいて通信の動的適応と機能拡張が実行される．この際，その目的を指定するために Wish を用いる．例えば，ストリーミングサーバを状況に応じて適したものに変更するなど，第 3.4.1 項で説明した目的を実行する場合，従来の通信アプリケーションでは，アプリケーションが自らサーバを検索し，相手先の変更といった適応の処理を実行する必要がある．これに対し Wish を用いた場合では，アプリケーションが u-session に対しストリーミングサーバへ接続するという目的を Wish により指定すれば，それに基づき通信相手の選択や適応処理などは u-session で実行される．

4.4.1 Wish

Wish とは，相手先の指定や適応時の方針などをテキスト形式で記述するものである．通信機能の柔軟な動作を可能とするため，Wish の指定はアプリケーションの開発者，利用者，状況認識システムやサービスの提供者など様々な立場から可能とする必要がある．アプリケーションから指定される Wish は，次節で説明する通信基盤提供部を通じて Wish 解析部に伝えられる．同時に，Wish 解析部は利用者とのユーザインタフェースや設定ファイルなどからも Wish を指定される．

4.4.2 Wish 解析部

Wish 解析部では Wish を解析し，その結果に基づいて動作制御部を制御する．第 3.4.2 項で述べた通り，Wish を解析し評価する手法は用途や目的に応じて置き換え可能であることが望ましい．つまり，Wish の解析部は u-session の本体から切り離して存在させ，状況に応じて置き換えることを可能とする必要がある．このため u-session は，Wish を管理し Wish 解析部に連絡してその解析を実行させる．

4.5 通信基盤提供部

通信基盤提供部は，分散通信アプリケーションに対して u-session が提供する通信インタフェースを実現する部分である．u-session では，第 3.3.2 で述べた通り，BSD Socket インタフェース互換の通信 API を提供する．

4.5.1 通信基盤提供部の構成

通信基盤提供部では，BSD ソケットインタフェースと互換性を持たせる必要がある．つまり，従来の動作を必要とする通信アプリケーションに対して，従来の BSD ソケットインタフェースと相違のない機能を提供し，本機構を必要とする通信アプリケーションに対して，u-session により実現される機能を提供することを可能とする必要がある．これは，図 4.4 に示す通りソケットドメインにより実現する．

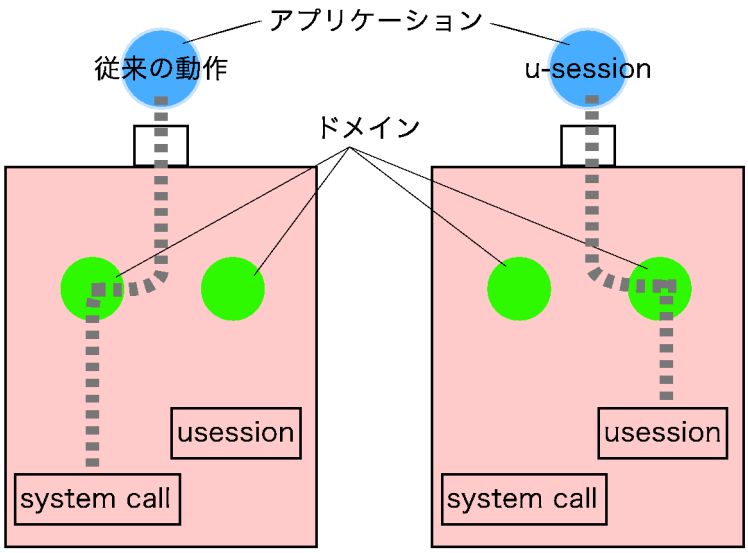

図 4.4: 通信基盤提供部の構成

通常，ソケットインタフェースに対するドメインの指定では，相手先のホスト名やア

ドレスを指定する．これに対し u-session の機能を用いる場合は，Wish ドメインを指定する．また，Wish ドメインにより，前節で説明した通信アプリケーションからの Wish 指定を可能とする．

4.5.2 通信基盤提供部が提供する API

通信基盤提供部が提供する API として，本論文では図 4.3 の API に対する拡張を実現する．

表 4.3: 動作制御部の機能

名称	実行する機能
socket	socket インタフェースの開始
close	socket インタフェースの終了
connect	接続処理を開始
read	受信
write	送信
getaddrinfo	ノード名からアドレス情報への変換
getnameinfo	アドレス情報からノード名への変換
refine	適応処理の実行

アプリケーションから socket が実行されると，通信基盤提供部は BSD Socket インタフェースの識別子を通信アプリケーションに対して通知する．この識別子は，通信アプリケーションが close を実行するまで有効となる．また，通信アプリケーションによる connect の実行により，動作制御部の処理が開始される．そして，通信の送受信は，その都度 read と write を実行することにより処理する．getaddrinfo と getnameinfo は，Wish ドメインを利用可能とするための変更が必要である．図 4.3 の最後に示した refine は，標準の BSD Socket インタフェースには存在しない．refine は，通信アプリケーションが動作制御部に対して明示的な適応処理を指示する場合に用いる．このため，通信アプリケーションからの実行が必須とされる機能ではない．

第5章

実装

本章では，**u-session** システムのプロトタイプ実装について説明する．前章で説明した設計に基づき，本システムは４つの部分から構成される．

5.1 u-session の実装

本章では，分散通信アプリケーションに対し，通信機能の動的適応と機能拡張を実現するためのセッション機構である u-session システムのプロトタイプ実装について述べる．まず，実装の概要について述べ，続いて個別の実装について説明する．

5.1.1 概要

実装環境は，表 5.1 に示す通りである．また，実装部分はに示す通り，大きく２つの部分に分けられ，通信ミドルウェアとユーザレベルのデーモンプログラムから構成される．これらにより前章で述べた設計に準ずる機能を実装した．

表 5.1: 実装環境

項目	説明
オペレーティングシステム	FreeBSD 5.1-RELEASE
実装言語	C 言語 gcc version 3.2.2 [FreeBSD] 20030205 (release)
ハードウェア	PC (Intel Pentium IV 2.4GHz)

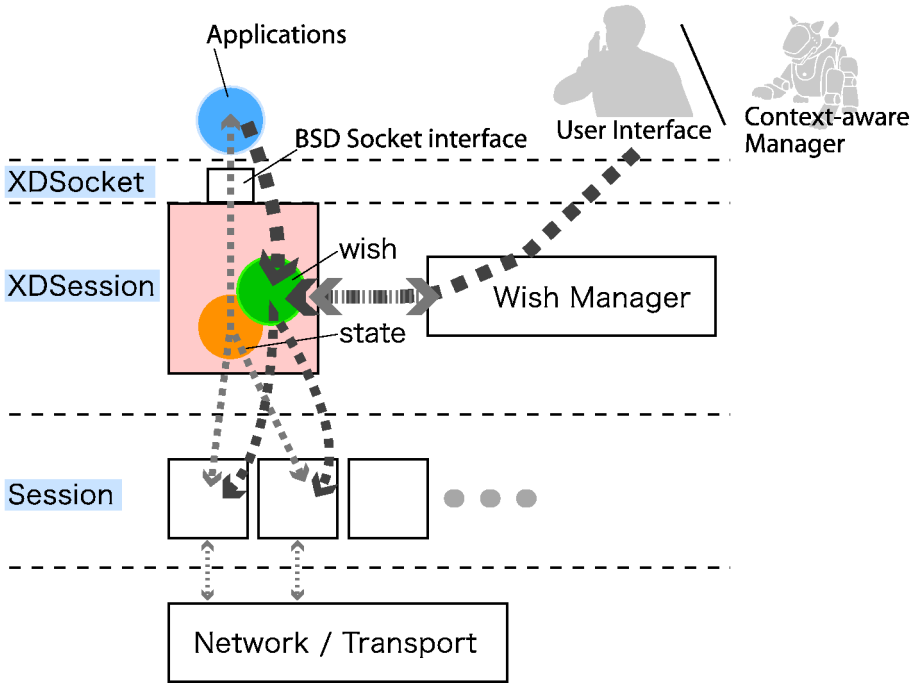

図 5.1: u-session システムの実装概要図

通信ミドルウェアは，各通信アプリケーションから本論文で実装した通信基盤提供部である xdsocket が呼び出されることにより実行が開始される．このため，通信アプリケー

ションの各 Socket インタフェースに対し，動作制御部である xdsession は 1 つ存在する．一方，Wish 解析部である Wish Manager は，各機器に 1 つ存在し，すべての xdsession から連絡を受ける．

5.1.2 動作の流れ

1. 通信アプリケーションが Socket インタフェースの開始を実行する際，getaddrinfo() の呼び出しにより，本機構が Wish Manager に対し通信する相手先の解決を要求する．この結果，Wish ドメインのアドレス構造体が通信アプリケーションに対して渡される．
2. そして，アプリケーションから Wish の指定が必要な場合は，ここで Wish ドメインのアドレス構造体に設定して次の行程に進む．
3. 通信アプリケーションから Socket インタフェースの開始が呼び出されると，u-session の実行が開始され，Wish Manager と協調して目的の通信を開始する．
4. 通信機能に適応動作が必要な場合は，Wish Manager を通じて u-session 内部で適応動作を実行する．その結果，適応した動作はアプリケーションによる次の送受信から有効となる．

5.2 session - 通信基盤依存処理部の実装

通信基盤依存処理部は，図 5.1 で session と示した部分に実装した．

5.2.1 実装した関数

第 4.2.1 項で述べた通信基盤依存処理部の機能を実装した関数の関数プロトタイプ宣言を図 5.2 に示す．これらの関数は，挙動制御部から呼び出される．

表 5.2: 通信基盤依存処理部の関数プロトタイプ宣言

機能名	関数名
開始	int session_new(struct session **sc);
終了	int session_clean(struct session *sc);
接続	int session_connect(struct session *sc);
切断	int session_close(struct session *sc);
受信	ssize_t session_read(struct session *sc, void *buf, size_t nbytes);
送信	ssize_t session_write(struct session *sc, void *buf, size_t nbytes);

また，session の管理に用いるデータ構造である session 構造体を図 5.2 に示す．

```
struct session {
 int     stat;           /* stat of myself */

 int     s;              /* fdesc of socket */
 ssize_t s_rsiz, s_wsiz; /* total rd/wr size */

 ssize_t (*sr_func)
  (struct session *sc, void *buf, size_t nbytes);
 ssize_t (*sw_func)
  (struct session *sc, const void *buf, size_t nbytes);

 int (*pstconct_func)(struct session *sc);
 int (*preclose_func)(struct session *sc);

 pthread_t        thrd_readsock;

 const char *dhost, *dport; /* destination host and port */
 struct addrinfo  hints;    /* hints of the destination */
 struct addrinfo *ai, *ai0; /* addrinfo of destination(s) */
};
```

図 5.2: session 構造体

session 構造体では，実際の通信相手との間の通信処理を実行するため，相手先のホスト名やアドレスに関する情報などを管理する．これらは，dhost, dport, hints, ai, ai0 のメンバ変数である．また，通信相手との間に確立されたコネクションの識別子が s であり，session の動作状態を管理するものが stat 変数である．その他のメンバについては，次項で説明する．

5.2.2 拡張性の実現方法

通信基盤依存処理部では，第 5.2.2 項で述べた拡張性を実現する必要がある．このため，session_read() と session_write() の実体は，それぞれ session 構造体のメンバである関数ポインタの sr_func と sw_func でポイントされた関数を実行する．標準では session_new() において，通信相手との単純な送受信関数がそれぞれの関数ポインタに指定される．この他にバッファリング機能を用いた処理や通信プロトコルなどの処理を実行する送受信関数を指定することが可能である．この場合，図 5.3 に示した session_setrwfunc マクロを用いてポイントする関数を変更する．また，読み込み時にバッファリング機能を実行する場合，相手先との通信を別スレッドで先読みする必要がある．その際にスレッドを管理するため，thrd_readsock 変数を用いる．また，通信プロトコルの処理を実行する場合，コネクション確立直後に認証処理やデータ構造の初期化などが必要であり，コネクションの終了前にはデータ構造の解放等が必要である．これらを実現するため，それぞれ pstconct_func と preclose_func の関数ポインタがあり，session_setpostconnectfunc と session_setpreclosefunc のマクロにより設定可能である．これらの関数ポインタが NULL

で無い場合，それぞれ session_connect() の終了直前と session_close() の開始直後に指定された関数が呼び出される．

```
#define session_setrwfunc(sc, rdfunc, wtfunc) \
do { \
        (sc)->sr_func = (rdfunc); \
        (sc)->sw_func = (wtfunc); \
} while (0)

#define session_setpostconnectfunc(sc, pstconctfunc) \
        (sc)->pstconct_func = (pstconctfunc)
#define session_setpreclosefunc(sc, preclosefunc) \
        (sc)->preclose_func = (preclosefunc)
```

図 5.3: 機能拡張を実現するマクロ

5.3 xdsession - 動作制御部の実装

動作制御部は，図 5.1 で xdsession と示した部分に実装した．

5.3.1 実装した関数

第 4.3.1 項で述べた動作制御部の機能を実装した関数の関数プロトタイプ宣言を図 5.3 に示す．

表 5.3: 動作制御部の関数プロトタイプ宣言

機能名	関数名
開始	int xdsession_new(struct xdsession **xdsc);
終了	int xdsession_clean(struct xdsession *xdsc);
接続	int xdsession_connect(struct xdsession *xdsc);
切断	int xdsession_close(struct xdsession *xdsc);
受信	ssize_t xdsession_read(struct xdsession *xdsc, void *buf, size_t nbytes);
送信	ssize_t xdsession_write(struct xdsession *xdsc, void *buf, size_t nbytes);
Wish 更新	int xdsession_update(struct xdsession *xdsc, struct sockaddr_wish *sw);
状態取得	int xdsession_getstat(struct xdsession *xdsc, struct sockaddr_wish *sw);
適応	int xdsession_refine(struct xdsession *xdsc);

また，session の管理に用いるデータ構造である session 構造体を図 5.2 に示す．

```
struct xdsession {
  TAILQ_ENTRY(xdsession)  hanger;   /* Tail Queue. */
  int                     stat;     /* stat of myself */

  int                     fd;       /* fdesc of socket */
  struct addrinfo         ai;       /* address info */

  int                     wishmngr; /* wishmngr */

  /* session list */
  LIST_HEAD(session_listhead, session_list) sc_head;
};
struct session_list {
  LIST_ENTRY(session_list)  hanger; /* List. */
  int                       pri;
  struct session           *sc;
};
```

図 5.4: xdsession 構造体

図 5.4 に示す通り，xdsession 構造体では通信基盤依存処理部の各 session 構造体をリスト構造で管理する．つまり，sc_head 変数を先頭とするダブルリンクドリストにより個別の通信相手との session を識別し，session の切り替えや拡張を可能とした．また，session 構造体のメンバ変数である pri はそれが管理する session 構造体の優先度を示し，この値に基づき各 session の優先度が決定される．

5.3.2 制御方法の実装

挙動制御部では，xdsession_new() 関数内で Wish 解析部に接続する．その後，xdsession 構造体の wishmngr メンバ変数を識別子として相互に通信可能となる．Wish 解析部から受け付ける制御コマンドは次の通りである．

新たな **session** の確立

NODENAME と SERVNAME で指定された通信相手に対する session を確立する．

CONNECT *NODENAME*:*SERVNAME*

session の一覧表示

現在の session 一覧を SESSION_ID とともに表示する．

LSSESSION

session に対する優先度の指定

SESSION_ID で指定された session の優先度を PRI に設定する．同時に複数の優先度を設定可能である．

SETPRI *SESSION_ID=PRI* [*SESSION_ID=PRI*]

5.4 Wish Manager - Wish 解析部の実装

Wish 解析部は，図 5.1 で Wish Manager と示した部分に実装した．この部分は，ミドルウェアから独立した単独のデーモンプログラムとして動作する．

5.4.1 動作制御部に対するインタフェースの実装

動作制御部から連絡を受ける部分は，システム標準の BSD ソケットを用いたプロセス間通信により実現した．この機能を実現するインタフェースは，ポート番号 5000 番で動作制御部からの接続を待機する．ここで受け付けるコマンドを以下に説明する．

ノード名の検索要求の受付

このコマンドは，通信アプリケーションが getaddrinfo() 関数の呼び出しにより通信相手のアドレス情報を取得する際，動作制御部から発行される．このコマンドに対し，指定された条件に合致するノード名を返す．

GETNODESERV *WISHCLASS*:*WISHTYPE* [*WISH*]

WISHCLASS では接続する機能を指定し、WISHTYPE で具体的なサービス名を指定する．また，WISH で検索時の条件を指定する．WISH は省略することが可能である．実際のコマンド例を以下に示す．

GETNODESERV SERVER:STREAMECHO NEAR

このコマンドでは，STREAMECHO の SERVER のうち近いものを検索することが指示されている．本機構では，このコマンドに対し次節で説明する手法を用いて適合するノード名を返す．

Wish の解析

このコマンドは，通信アプリケーションにより通信機能に対する Wish が変更された場合に呼び出される．本機構では受け付けた Wish と現在の動作を比較し，適応処理が必要な場合は動作制御部に対し refine コマンドを実行する．

PARSEWISH *WISH*

5.4.2 通信相手の検索方法

本論文の実装によるサービス検索では，以下に示すテキスト形式の対応表を用いた．

```
# WISHCLASS WISHTYPE     NODESERV
  SERVER    COUNTECHO    pbal:http
  SERVER    COUNTECHO    rom:http
```

本機構ではこの中から適合する NODESERV を選出し，動作制御部に対して返答する．

5.5 xdsocket - 通信基盤提供部の実装

通信基盤提供部は，図 5.1 で xdsocket と示した部分に実装した．実装環境の Socket プログラミングインタフェースは，オペレーティングシステム内の機能を呼び出すシステムコールとして実装されている．また，一般的な Unix 互換オペレーティングシステムでも，同様の実装方式をとる場合が多い．このため本論文の実装では，第 4.5.2 項に示した API をライブラリとして実装し，その内部でソケットドメインにより u-session の機能と従来のシステムコールの呼び出しを切り替え可能とした (図 5.5 参照)．

```
API 名 (){
        if (ソケットドメイン or アドレスファミリ == AF_WISH) {
                xdsession の機能を呼び出す;
        } else {
                API のシステムコールを呼び出す;
        }
}
```

図 5.5: xdsocket の概要

このため，従来の機能をそのまま利用するアプリケーションに対しての動作は，この部分で従来のシステムコールをそのまま呼び出すのみとなる．一方，u-session システムの機能を用いた動作を必要とする場合は，図 5.6 に示す sockaddr_wish 構造体をソケットドメインとして用いる．

```
struct sockaddr_wish {
  unsigned char    swi_len;     /* length of this struct */
  sa_family_t      swi_family;  /* AF_WISH */
  u_int8_t         swi_adstlen; /* length of swi_addr */
  u_int8_t         swi_wishlen; /* length of swi_wish */
  struct wish     *swi_wish;    /* wish of this */
  struct addrstat *swi_adst;    /* actual address & state */
};
struct wish {
        int      len;   /* length of txt */
        char    *txt;   /* wish */
};
```

図 5.6: sockaddr_wish 構造体

第6章

評価

本章では，分散通信アプリケーションに対し通信機能の動的適応と機能拡張を実現するセッション機構のプロトタイプ実装である **u-session** について評価する．まず，本研究の目的から設計したシステムの妥当性について考察し，プロトタイプ実装の動作について検証する．その後，既存のシステムなどと比較して論じる．

6.1 要求機能との考察

本研究の目的は第 1.2 で述べた通り，人の周囲に偏在する機器で動作するアプリケーションに対し，通信機能の動的適応と機能拡張を支援するための基盤環境を提供することである．そして，これにより機器の柔軟な利用環境の構築を目指す．本論文ではこの目的を実現するため，通信機能の動的適応と機能拡張を実現するセッション機構である u-session システムを提案し，そのプロトタイプ実装について説明した．この際，u-session システムに必要とされる要求機能を第 3.1 項で述べた．まず，これらの要求機能が u-session システムにおいて満たされているか確認する必要がある．以下，順に考察する．

通信機能の構築方法や実行方法に柔軟性をもたせること

u-session システムでは，通信機能における動的適応や機能拡張をその内部に隠蔽して実行可能な機構を構築することができた．例えば，相手先の変更や通信処理の機能拡張などが可能である．

通信機能を機器やアプリケーションから機能分散して実装可能とすること

u-session システムでは，アプリケーションを個別のサービス検索手法や名前空間から独立させ，通信プロトコルごとの処理とそれに関連する情報の管理などを分離して通信機構の内部に隠蔽した．これにより，通信アプリケーションと個別の通信機能に固有な処理や情報を分離することができた．このため，アプリケーションは通信機能の適応処理を意識する必要がない．

柔軟な挙動の指定を可能とする方法を確立すること

Wish に基づく動作制御により，常に多方面からの柔軟な挙動制御を可能とした．つまり，アプリケーションの開発者に限らず，利用者，機器やサービスの提供者，状況認識システムなどから，Wish に基づく通信機能の制御が可能である．また，サービス検索や評価手法など目的に応じ，選択すべき手法が異なる機能を外部に独立したプログラムとしたため，用途に応じ別のプログラムに切り替えることが可能である．

“いつでも・どこでも”十分に機能する基盤環境として構築すること

通信機能をアプリケーションから分離する際，u-session システムで用いた BSD Socket インタフェースは，多くの環境で利用可能である [11]．このため，BSD Socket インタフェースに完全互換とした設計により，“いつでも・どこでも”十分に機能する基盤環境として構築できたと言える．また，BSD Socket インタフェースを用いたため，設計の上から u-session システムが特定のプログラミング言語や実行環境に依存していないと言える．

このように第 4.1.1 項で設計方針に従うことにより，本研究で必要とされる要求機能を満たすことが確認できた．このため，u-session システムは第 1.2 で述べた目的を達成していると言える．

6.2 動作検証

ここでは，第 5 章で述べた u-session システムのプロトタイプ実装について，実際の動作を示して検証する．検証に用いた環境は実装環境と同一である．

はじめに，動作内容について説明する．まず，1 から 10000000 までの数字を一行ごとに記述したファイルを用意し，別の 2 台の Web サーバを通じてダウンロード可能となるように設置した．そして，単純な HTTP クライアントを用いて，u-session システムを通じて設置したファイルを取得し，　その際に u-session システムにより動的なサーバの切り替えを実現する．Web サーバは pbal と rom であり，u-session が動作するクライアントは polo である．なお，polo から pbal と rom の Web サーバに対するセッションは，常に多重化されている．各セッションの通信シーケンス図を図 6.1 に示す．

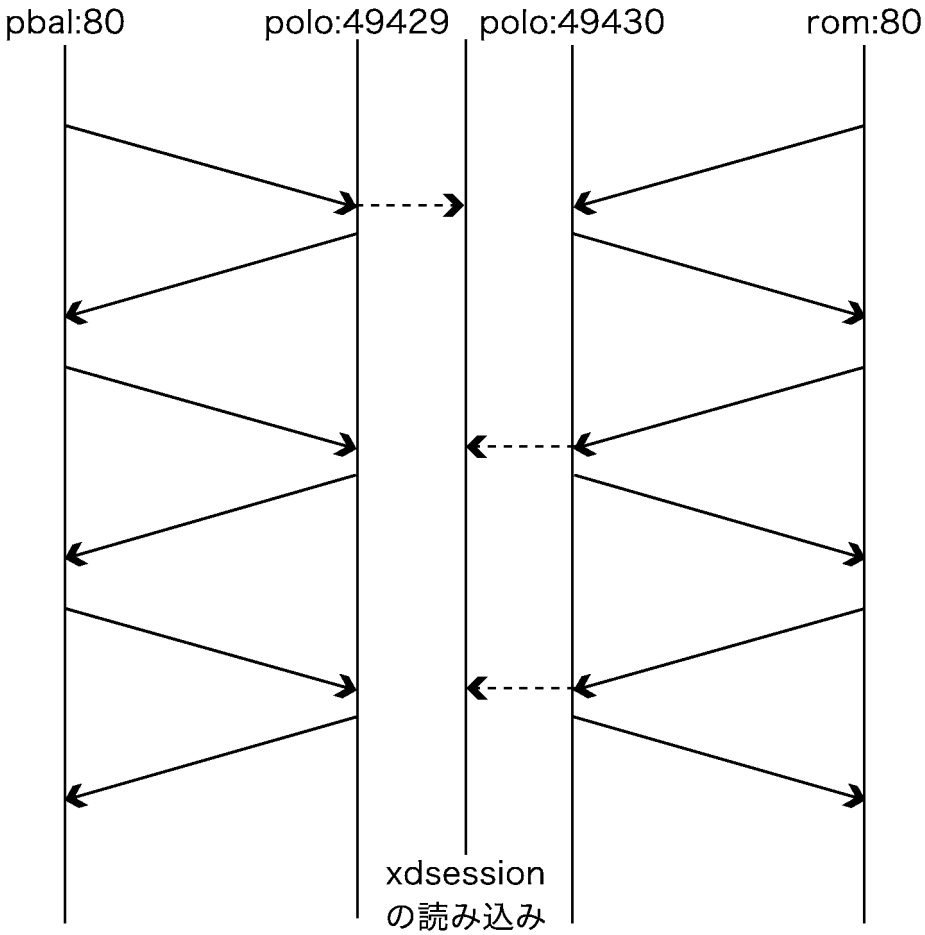

図 6.1: 各セッションの通信シーケンス図

6.3 議論

本節では，u-session システムと既存の関連研究とを比較し，u-session システムの特徴を分析する．

6.3.1 関連研究との比較

u-session システムと第 3.5 項で述べた関連研究との比較を表 6.1 にまとめる．

表 6.1: 関連研究の相違

	Rocks+Racks	MobileSocket	SinkProxy	SoNS	u-session
視点	ホスト指向	アプリケーション指向	アプリケーション指向	タスク指向	タスク指向
実行環境や構築環境の多様性対応	Socket に依存	Java に依存	Java に依存	Socket に依存	Socket に依存
柔軟な挙動制御	×:同一ホスト間	×:同一アプリケーション間	×:同一フレームワーク間	事前指定制約論理	動的指定外部依存
透過的動作	完全透過	完全透過	処理のみ	完全透過	処理のみ
サービス検索手法	−	−	独自	拡張可	拡張可
通信機能や連係処理の分離	×	×	×	動的適応	動的適応機能拡張

以下，それぞれの項目について比較した結果を述べる．視点の違いにより実行環境や動作環境に依存する部分が異なり，多様性に対応可能な範囲の違いが生じる．特に “いつでも・どこでも” 利用可能な基盤環境は，特定のプログラミング言語に依存せずに構築可能であることが望ましい．そして，指向する視点の違いが挙動制御の柔軟性に影響を与えることが明らかである．また，本論文では分離した通信機能が制御不能とならないことを要件として挙げた．このため，完全な透過により処理が制御不能となるもことは望ましくない．また，場面などに応じて適したサービス検索手法を利用可能であることが望ましい．これらの相違から，既存の分散通信アプリケーション構築手法で通信基盤において，動的適応を実現しているものは SoNS であり，動的適応と機能拡張の両方を実現しているものは本論文で提案した u-session であることがわかる．

第7章

結論

本章では，本論文のまとめと今後の課題について述べる.

本論文では，はじめに通信アプリケーションの視点から，ユビキタスコンピューティング環境において見受けられる特徴や問題点を整理した．そして，ユビキタスコンピューティング環境が機器間の相互通信を前提とした異質かつ動的な分散環境であることを明らかにした．このため，機器で動作するアプリケーションが相互に通信し，連携して動作する必要がある．しかし，従来の分散アプリケーション構築手法では，通信機能が通信アプリケーションと同化しているため，通信機能において多様性への対応を実現することが困難であった．

このため本論文では，通信機能の動的適応と機能拡張を支援するセッション機構を提案し，u-session システムとして設計・実装した．u-session システムは，通信アプリケーションから通信機能を分離し，目的を指定することにより通信機能の適応動作を可能とする機構である．また，特定のプログラミング言語や実行環境に依存しない設計であるため，異質で多様なユビキタスコンピューティング環境において，“いつでも・どこでも”十分に機能する基盤環境として動作できる．そして，本論文で提案した機構により，機器の柔軟な動作環境を構築することが可能となる．

本論文をふまえ，今後の課題を以下にまとめる．

通信プロトコル処理の拡充

> さまざまな通信プロトコルを session として実装し，u-session から利用可能な環境を整備する必要がある．これにより，本システムが対応できる適応と拡張の範囲が広がる．

異なる通信プロトコルの並行処理

> 現在の実装では，異なる通信プロトコルの場合に並行して動作させていない．複数の異なる通信プロトコルで並行して通信し，切り替え可能とすることで，異質な環境での柔軟な通信基盤の優位性を示すことが可能となる．

謝辞

本論文の執筆にあたり御指導頂きました，慶應義塾大学環境情報学部教授兼政策・メディア研究科委員長の徳田英幸博士をはじめ，本論文の副査としてご助言を頂きました慶應義塾大学政策・メディア研究科助教授の高汐一紀博士，慶應義塾大学環境情報学部教授兼政策・メディア研究科委員の村井純博士に深謝致します.

2004 年 1 月 14 日

権 藤 俊 一

参考文献

[1] エヌ・ティ・ティ・コミュニケーションズ株式会社. HOTSPOT. http://www.hotspot.ne.jp/.

[2] 日本通信株式会社. b モバイル. http://www.hotspot.ne.jp/.

[3] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, Vol. 36, No. 7, pp. 74–84, July 1993.

[4] 株式会社日立製作所. アンテナ内蔵型「ミューチップ」を開発 . http://www.hitachi.co.jp/New/cnews/030902a.html.

[5] セコム株式会社. ココセコム [位置情報・現場急行サービス] . http://www.855756.com/top.html.

[6] 株式会社東芝. ネット de ナビ . http://www.rd-style.com/products/rdx4/can/net_navi_amm.htm.

[7] 日本電算機株式会社. iBOX プロードメディアサーバ . http://www.jcc.co.jp/iboxserver/server-top.html.

[8] アップルコンピュータ株式会社. iPod. http://www.apple.co.jp/ipod/index.html.

[9] Griffin Technology. iTrip. http://www.griffintechnology.com/products/itrip/index.html.

[10] UC Berkeley Computer Systems Research Group. Berkeley Socket API. 4.1cBSD system, 1982.

[11] M. Tim Jones. *BSD Sockets Programming From A Multi-Language Perspective*. Charles River Media, 2003.

[12] K. Thompson and D.M. Ritchie. The UNIX Time-Sharing System. *Communications of the ACM*, Vol. 17, No. 7, pp. 365–375, July 1974.

[13] W. リチャードスティーヴンス/篠田陽一. UNIX ネットワークプログラミング Vol.1 ネットワーク API:ソケットと TLI. ピアソン・エデュケーション, 1998.

[14] Project JXTA. Jxta. http://www.jxta.org/.

[15] W. リチャードスティーヴンス/篠田陽一. UNIX ネットワークプログラミング Vol.1 ネットワーク API:ソケットと TLI. ピアソン・エデュケーション, 1998.

[16] Sun Microsystems, Inc. Java Remote Method Invocation (Java RMI) . http://java.sun.com/products/jdk/rmi/.

[17] Object Management Group, Inc. CORBA Common Object Request Broker Architecture. http://www.corba.org/.

[18] World Wide Web Consortium (W3C). SOAP Simple Object Access Protocol. http://www.w3.org/TR/SOAP/.

[19] ACM Computing Surveys (CSUR). Process Migration . http://portal.acm.org/citation.cfm?id=367728&dl=ACM&coll=portal.

[20] Ichirou Satoh. Mobile Space. http://research.nii.ac.jp/ ichiro/agent/agentspace.html.

[21] C. Perkins. IP Mobility Support. RFC 2002 (Updated by 2290, 2794) PROPOSED STANDARD, IETF, 1996.

[22] D.B. Johnson and C. Perkins. Mobility Support in IPv6. draft-ietf-mobileip-ipv6-06.txt Work in progress, IETF, 1998.

[23] Adobe Systems Incorporated. PDF: Portable Document Format. 1993.

[24] World Wide Web Consortium XML Core Working Group. XML: Extensible Markup Language. http://www.w3.org/XML/.

[25] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 2060 (Obsoletes RFC1730) PROPOSED STANDARD, IETF, 1996.

[26] W. リチャードスティーヴンス/篠田陽一. UNIX ネットワークプログラミング Vol.1 ネットワーク API:ソケットと TLI, 第 1.3 章. ピアソン・エデュケーション, 1998.

[27] IEEE 802.3 CSMA/CD (ETHERNET). http://grouper.ieee.org/groups/802/3/.

[28] J. Postel. Transmission Control Protocol. RFC 793 STANDARD, IETF, 1981.

[29] J. Postel. Internet Protocol. RFC 791 Standard, IETF, 1981.

[30] S. Deering and R. Hinden. Internet Protocol Version 6 (IPv6) Specification. RFC 2460 Draft Standard, IETF, 1998.

[31] P. Johansson. IPv4 over IEEE 1394. RFC 2734 Draft Standard, IETF, 1999.

[32] IEEE 802.15.1. Bluetooth Network Encapsulation Protocol (BNEP) Specification. http://grouper.ieee.org/groups/802/15/Bluetooth/BNEP.pdf.

[33] Victor C. Zandy and Barton P. Miller. Reliable network connections. In *Eighth ACM International Conference on Mobile Computing and Networking (Mobicom '02)*, pp. 95–106, Atlanta, GA, USA, September 2002.

[34] Tadashi Okoshi, Masahiro Mochizuki, Yoshito Tobe, and Hideyuki Tokuda. Mobile-Socket: Session Layer Continuous Operation Support for Java Applications. 情報処理学会 論文誌, Vol. 40, No. 6, pp. 10–14, 2000.

[35] Jorge Rafael Nogueras. A Stream Redirection Architecture for Pervasive Computing Environments. Master's thesis, Computer Science and Engineering at the Massachusetts Institute of Technology, 2001.

[36] Umar Saif and Justin Mazzola Paluska. Reliable network connections. In *First International Conference on Mobile Systems, Applications and Services, (MobiSys 2003)*, San Francisco, USA, 2003.