

博 士 論 文

ユビキタスコンピューティングシステムを
効率よく実現するソフトウェアアーキテクチャ

矢代 武嗣

論文要旨

RFID等の識別技術やセンシングに関する技術、さらには省電力無線による通信技術等の発展にともない、ユビキタスコンピューティングの実現が現実的になってきた。しかし要素技術に関する研究の進展に比べ、これらの要素を組み合わせるサービスを実現するためには未だ課題が多い。例えば、ユビキタスコンピューティングにおいては、様々なセンサやデータベースからの情報を収集し、それらの情報を統合して状況(コンテキスト)を推論した上で適切な動作を行う必要があるが、異なる情報を統合するための枠組みがないとシステム構築コストが上がり拡張性にも欠ける。これまでの情報システムの構築においては、ユーザが手作業で入力する情報と、対象となるアプリケーションが想定するいくつかの決まった形式のデータをもとに定型的な処理を行うコンピュータシステムが一般的であったが、ユビキタスコンピューティングにおける状況の認識とそれに基づく判断は、様々な種類の曖昧性を持ったデータを統合し、そこから現在の状況に対する推論を行い、サービス利用者が望むシステムの挙動を自動的に適切な動作を判断・選択して実行する必要がある。このようなシステムを既存のソフトウェア開発技術を用いて簡単に実現することは困難である。

このような問題を解決するため、本研究では利用者に対してサービスのインタフェースを提供する端末におけるソフトウェアアーキテクチャ「UC (ubiquitous communicator) フレームワーク」を提案し、実際にそれに基づくシステムの構築を行った上で、評価を実施した。UCフレームワークは、ユビキタスコンピューティングにおけるユーザインタフェース端末上でのサービスの実現を容易にすることを目的に設計されている。UCフレームワークは、コンテキスト推論フレームワーク、アプリケーション間調停フレームワーク・省電力制御フレームワークの3モジュールからなる。コンテキスト推論フレームワークは、様々なセンサやデータベースからの情報を収集・統合し、利用者の身の回りのコンテキストを推論するための仕組みである。アプリケーションフレームワークはユビキタスコンピューティングにおけるサービス間の調停を行うための仕組みであり、様々なサービスが同時に動作している状況下において、サービス間の動作を調停し、システム全体として適切な機能を実現するための仕組みである。省電力制御フレームワークは、ユビキタス情報システム端末に適した省電力制御を実現するためのモジュールである。これら3つのモジュールにはフレームワークの挙動を、簡潔に記述された外部ルールにより柔軟に変更する

機構が備わっており、分かりやすい少量の記述を行うだけで、実現したいシステム動作を容易に実現できる点に特徴がある。

UC フレームワークの実装は、組込みリアルタイム OS 「T-Kernel」 上に行った。本システムの有効性を検証するために、ターゲットアプリケーションとしてユビキタス観光情報サービスを実現するシステム「UC 場所情報システム」の構築を行った。「UC 場所情報システム」は、東京都がすすめる東京ユビキタス計画の一環として構築された場所情報を提供するためのユビキタスコンピューティングシステムの一つであり、すでに上野動物園、浜離宮恩賜庭園を含む東京都内の観光地において実用化されている。このシステムに UC フレームワークを導入することで、ユビキタス情報システムの開発効率及び完成度に与える影響を評価した。その結果、開発効率の観点からは、少ない記述量でのシステム動作を実現できたほか、新しいサービスや機能の追加などのシステム動作の変更要求に対し容易に追従できることも確認できた。この特性を利用し、本システムではサービス提供者からのヒアリングに応じた改善を繰り返し、細かい改善を積み重ねることで全体としてのシステムの完成度が向上した。これを評価するため、東京ユビキタス計画における利用者のアンケートを実施し、利用者の満足度に与える影響を調査した。結果、有意に利用者の満足度が向上することが確認でき、本システムが実際のユビキタスコンピューティングシステム構築において有用であることが示された。

謝辞

本研究を行うにあたり丁寧かつ熱心なご指導をくださった、坂村健教授、越塚登教授、小林真輔様、別所正博助教に感謝いたします。また、関連する研究プロジェクトを進めるにあたりご協力いただいた、山田純様、泉名達也様、神尾真人様、恩本浩二様をはじめとする YRP ユビキタスネットワーク研究所の皆様、ならびにユーシーテクノロジー株式会社、坂村・越塚研究室の皆様にも御礼を申し上げます。

目次

第 1 章 序論	1
1.1 背景	1
1.1.1 ユビキタスコンピューティング技術の進展	1
1.1.2 ユビキタスコンピューティングにおける情報処理	3
1.1.3 ユビキタスコンピューティングのためのフレームワークにおける課題	4
1.2 ユビキタスネットワークアーキテクチャ	5
1.3 UC フレームワーク	7
1.4 システム実装	9
1.5 本研究の成果	10
1.6 論文の構成	11
第 2 章 関連研究・技術	12
2.1 ユビキタスコンピューティングのためのアーキテクチャ	12
2.1.1 Gaia アーキテクチャ	12
2.1.2 SCaLaDE アーキテクチャ	16
2.1.3 MIMOSA フレームワーク	18
2.1.4 LOCA フレームワーク	21
2.1.5 その他のフレームワーク	24
2.2 ユビキタスコンピューティングのための省電力制御技術	25
2.2.1 Nishihara らによる省電力化手法	25
2.2.2 SeeMon フレームワーク	28
2.2.3 その他のコンテキストに基づく省電力化手法	29

第 3 章 ユビキタスネットワークアーキテクチャ	31
3.1 全体構造	31
3.2 ユビキタス ID アーキテクチャ 2.0	33
3.2.1 ucode	33
3.2.2 ucode 解決	33
3.2.3 実装	36
3.3 機器向けのソフトウェアフレームワーク	36
3.3.1 CoAP (constrained application protocol)	36
3.3.2 uID-CoAP アーキテクチャ	38
3.3.3 組込み機器向け CoAP フレームワーク	38
3.4 eTNet アーキテクチャ	42
第 4 章 UC フレームワーク	46
4.1 解決すべき課題	46
4.1.1 自動的な動作	46
4.1.2 高い開発効率	47
4.1.3 システム全体としての最適動作	48
4.2 設計方針	48
4.2.1 メカニズムとポリシーの分離	49
4.2.2 アプリケーション間調停機構の導入	49
4.2.3 コンテキストの抽象化	50
4.3 全体構成	51
4.4 コンテキスト推論フレームワーク	53
4.4.1 概要	53
4.4.2 コンテキストポリシー記述	55
4.4.3 コンテキストの再計算	56
4.5 アプリケーション間調停フレームワーク	57
4.5.1 概要	57
4.5.2 アプリケーション間調停ポリシー記述	59
4.5.3 アプリケーション間調停の実行	60
4.6 省電力制御フレームワーク	60
4.6.1 概要	60

4.6.2	電源管理ポリシー記述	60
4.6.3	省電力制御禁止機能	61
4.6.4	省電力制御処理の流れ	62
第 5 章	システム実装	63
5.1	システムアーキテクチャ	63
5.2	ハードウェア	63
5.3	Qt/T-Kernel アーキテクチャ	66
5.4	UC フレームワーク	69
5.4.1	コンテキストマネージャ	69
5.4.2	アプリケーションマネージャ	73
5.5	UC アプリケーションライブラリ	75
5.6	UC 場所情報システム	77
5.6.1	概要	77
5.6.2	アプリケーション構成	78
5.6.3	UC ブラウザ	80
5.6.4	UC マップ	81
5.6.5	UC パノラマビュー	81
5.6.6	UC ホーム	85
5.6.7	UC 履歴管理	86
第 6 章	評価	87
6.1	開発効率	87
6.1.1	コード量	87
6.1.2	アプリケーション記述	88
6.1.3	コンテキスト推論ルール	89
6.1.4	アプリケーション間表示調停ルール	92
6.1.5	UC12 システムとの比較	94
6.2	最適動作の実現性	98
6.2.1	評価方法	98
6.2.2	機器の操作性に関して	99
6.2.3	利用時の楽しさに関して	100

6.2.4	再利用の希望に関して	100
6.2.5	有料化の際の支払い金額の上限に関して	101
6.2.6	道路や街なかへの展開に関して	101
6.3	パフォーマンス	102
6.4	電力消費	105
6.4.1	実験内容	105
6.4.2	利用したコンテキスト	107
6.4.3	電力制御ルール	108
6.4.4	実験結果	109
第 7 章	おわりに	113
7.1	まとめ	113
7.1.1	関連研究	113
7.1.2	ユビキタスネットワークアーキテクチャ	114
7.1.3	UC フレームワーク	114
7.1.4	システム実装	115
7.1.5	評価	116
7.2	結論	117
付録 A	UC フレームワークの API 仕様	125
A.1	コンテキストマネージャAPI仕様	125
A.1.1	エンドポイント	125
A.1.2	メソッド	125
A.1.3	シグナル	127
A.2	アプリケーションマネージャAPI仕様	128
A.2.1	エンドポイント	128
A.2.2	メソッド	128
A.2.3	シグナル	129
付録 B	UC アプリケーションライブラリの API 仕様	131
B.1	UcApplication クラス	131
B.2	UcDirection クラス	132
B.3	UcLocation クラス	133

B.4	UcMainWindow クラス	134
B.5	UcUcode クラス	134

図一覧

1.1	超機能分散環境 (TRON プロジェクト [46])	2
2.1	GaiaOS におけるディスカバリサービス ([43] より引用)	13
2.2	Gaia MPACC フレームワーク ([43] より引用)	14
2.3	ACD によるアプリケーション構成の記述例 ([41] より引用)	15
2.4	美術館ガイドシステムにおける MA プロキシの動作 ([5] より引用) . .	17
2.5	SCaLaDE アーキテクチャ ([5] より引用)	18
2.6	SCaLaDE メタデータの例 ([5] より引用)	19
2.7	MIMOSA アーキテクチャ ([30] より引用)	20
2.8	CARE ミドルウェアにおけるコンテキスト処理のポリシー記述 ([7] より引用)	20
2.9	LOCA におけるカテゴリマッチ・ロケーションマッチの処理手順 ([1] より引用)	22
2.10	LOCA におけるネットワークマッチの処理手順 ([1] より引用)	23
2.11	一階確率論理 (FOPL) に基づく推論規則の定義例 ([38] より引用) . . .	24
2.12	Nishihara らによる提案システム ([34] より引用)	26
2.13	SeeMon フレームワーク ([24] より引用)	29
3.1	ユビキタスネットワークアーキテクチャ概要	32
3.2	uID アーキテクチャ 2.0	35
3.3	uID アーキテクチャ 2.0 における ucode 解決の流れ	35
3.4	CoAP による温度設定の取得例	38
3.5	uID-CoAP ネットワークアーキテクチャ	39
3.6	組込み機器向け CoAP フレームワーク	40
3.7	デバイスインタフェース定義のための構造体定義	41

3.8	GET メソッドの実装例	42
3.9	eTRON アーキテクチャ	43
3.10	eTNet におけるセッション通信のフロー	45
4.1	UC フレームワークの概要	52
5.1	ユビキタス情報システムのアーキテクチャ	64
5.2	UC110 の外観	66
5.3	Qt/T-Kernel アーキテクチャ	68
5.4	コンテキスト推論ルールの定義例	71
5.5	アプリケーションマネージャとアプリケーションの表示管理領域区分	73
5.6	表示調停ルールの記述例	74
5.7	省電力制御ルールの記述例	76
5.8	上野動物園場所情報サービスに追加されたクイズ機能	79
5.9	UC ブラウザの画面	82
5.10	UC マップの画面	83
5.11	UC パノラマビューの画面	84
5.12	UC ホームの画面	85
5.13	UC 履歴管理の画面	86
6.1	利用者の位置変化に対するコンテンツプッシュ処理記述	90
6.2	浜離宮恩賜庭園向けシステムのコンテキスト推論ルール (一部省略)	91
6.3	上野動物園向けシステムの表示調停ルール	93
6.4	UC12 システムの全体アーキテクチャ	95
6.5	ucode ルールの記述例	97
6.6	機器の操作性についての感想	99
6.7	サービスの利用の楽しさについての感想	100
6.8	再度利用したいか否かについて	101
6.9	有料サービスとなった場合の支払い可能金額について	102
6.10	本システムの道路や街なかへの展開について	103
6.11	各地点における現在位置変化に対するシステム処理時間 (実時間)	104
6.12	電力消費量評価実験のためのガイドルート	106
6.13	LCD 輝度制御に関する電力制御ルール	110

6.14 WiFi スキャン頻度に関する電力制御ルール	111
6.15 Dice(RFID) 受信に関する電力制御ルール	112
6.16 省電力制御を行わない場合に対する消費電力量の削減率	112

表一覧

2.1	リソースコントロールテーブルの定義例 ([34] より引用)	28
5.1	UC110 のハードウェア仕様 (概要)	65
6.1	UC フレームワーク上のアプリケーションサイズ	88
6.2	UC12 システムと提案システムのシステム動作記述のサイズ	96

第 1 章

序論

1.1 背景

1.1.1 ユビキタスコンピューティング技術の進展

コンピュータの技術の進歩に伴い、ユビキタスコンピューティングの考え方がますます注目を集めている。特に近年は現実世界に存在するモノのネットワーク接続性に特に主眼をおいて「モノのインターネット」(Internet of Things; IoT) [3] といった言葉で呼ばれることが多いが、本質的な考え方は 1980 年代に登場した先駆的な研究 [52, 46] 以来変わらずに続いているものである。その中でも特に、最新のユビキタスコンピューティングのビジョンに近いものが、坂村らによって提唱された TRON プロジェクトにおける超機能分散システム [46] である。

図 1.1 は TRON プロジェクトにおける超機能分散システム (highly functionally distributed system; HFDS)、すなわち現代でいうユビキタスコンピューティングや IoT の考えを示したものである。その中心となる概念は、身の回りに存在するあらゆるモノにマイクロコンピュータが埋め込まれ、それらが通信機能を持ち、互いに情報を交換しながら協調動作することで、人間の生活をより高度にサポートするというものである。従来のコンピューティングシステムにおいては、人間がキーボードやマウスなどの機器を用いて入力を行い、それに応じた処理を行うというものが一般的であった。一方、ユビキタスコンピューティングにおいては、多数のコンピュータ群がセンサ等を通して状況を自動的に認識し、それに基づく最適な動作を行うことで人間の生活をサポートするという点が異なる。空調の制御を例にあげるならば、スイッチを押下することでそれに基づき温度調整や電源の ON/OFF 処理を行うのが

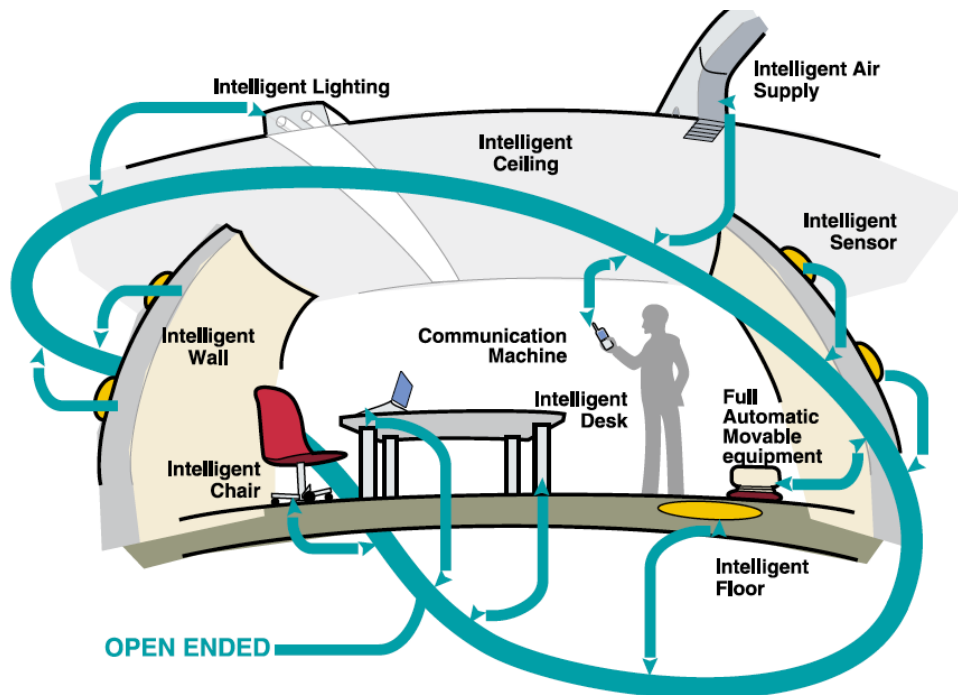


図 1.1: 超機能分散環境 (TRON プロジェクト [46])

従来型のコンピューティングシステムであるが、環境の温度や部屋内部の状況 (人の有無など) を含む様々な情報を自動的に収集し、それらに基づき最適な制御を行うのがユビキタスコンピューティングの考え方である。

このようにユビキタスコンピューティングは従来型のコンピューティングシステムとは大きく異なっていることから、その実現には多数の課題があった。要素技術の分野においては、あらゆるものに埋め込み可能な省電力なマイクロコンピュータの実現、省電力な通信プロトコルの実現、セキュリティの実現などが課題となっていた。一方で、近年のユビキタスコンピューティング分野における要素技術の発展はめざましく、ようやくユビキタスコンピューティングの実現に向けた要素技術が揃いつつあると言える。身の回りのモノにユビキタスコンピューティングの機能を実現するためには、省電力での通信や制御の実現が必要になるが、特に省電力な無線ネットワークの分野においてはここ最近の標準規格化がめざましい。物理層・MAC層における IEEE 802.15.4 [9] の標準化についてはもちろん、ネットワーク層のプロトコルとして 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks)[32] が

制定されたことが大きな成果である。6LoWPAN の実現により、前述のモノによって実現されるネットワーク (センサネットワーク) と、標準のインターネット (IPv6) とを直接的に接続するための具体案が標準化されたこととなる。これは、図 1.1 における “open ended” なネットワーク基盤が具体化されたことを意味している。また、省電力制御においても MCU の発展、リアルタイム OS 技術や TinyOS [28]・Contiki OS [26] 等のイベントドリブン型のオペレーティングシステムに基づくセンサノードのためのプラットフォームに関する研究はもちろん、power harvesting [11] に関する研究開発も盛んに行われており、要素技術に関しては多くの課題が解決へと向かいつつある。

1.1.2 ユビキタスコンピューティングにおける情報処理

ユビキタスコンピューティングの分野における要素技術に関する多くの問題が解決されてきた一方で、これらを組み合わせて情報システムを実現するには未だ多くの課題が残っている。ユビキタスコンピューティングにおいては、様々なセンサやデータベースからの情報を収集し、それらの情報を統合して状況 (コンテキスト) を推論し、その上で適切な自動処理を行うことが必要になる。これはユーザの入力に基づき処理を行う従来型のコンピュータシステムの考え方とは大きく異なっている。具体的な特徴としては以下の3つが挙げられる [4]。

動的な処理内容の変化

ユビキタスコンピューティングにおいては、ユーザのおかれた環境やそれに起因する不確定性に対応して動的に処理内容を変更することが求められる。例えば、ユーザは環境の変化により突然それまで行っていた行動をやめて新たな行動を始めるかもしれない。このほか、突然の緊急事態により、それまで行っていた作業を中断して避難を始めるかもしれない。このような変化に対し、処理内容を状況に応じて適切に切り替えるための仕組みが求められる。

多様なデバイスとリソースに関する制約

ユビキタスコンピューティングの利用者は一箇所にとどまっているとは限らず、様々な場所でそのサービスを享受できなければならない。しかしながら、環境によってはデバイスの構成も異なるほか、状況によってはそれらに計算リソースや電力等の制約がある可能性がある。たとえば、利用者がオフィスから自動

車の中に移動した場合に、サービスをその環境に適合させて実現させるための仕組みが求められる。

人間社会に密接に関連したコンピューティング

ユビキタスコンピューティングのもうひとつの特徴としては、それが使われる社会環境に大きな影響を与えることにある。たとえば、ユビキタスコンピューティングの実現のためにセンサを導入した時、その情報を他者が閲覧できてしまうというのは社会生活に大きな影響を与える可能性がある。たとえば、警察がこれらのセンサ情報に基づき、この家には人がいるか、その人がどのくらいのアルコールを摂取しているかなどの情報を入手できたとすると、これは我々の社会生活にとっては大きな影響を与えることになる。このことから、ユビキタスコンピューティングにおいてはアクセス制御と、その内容を決めるためのポリシーが重要な意味を持つ。

このように、ユビキタスコンピューティングは従来型のコンピューティングシステムとは大きく異なる特徴を持っている。ユビキタスコンピューティングを実現するためには、多様なデータをもととした状況認識とそれに基づく制御が必要となるが、これは従来の情報システムの方式では不十分である。これまでの情報システムにおいては、ユーザが手入力する情報と、対象となるアプリケーションが想定するいくつかの決まった形式のデータをもとに定型的な処理を行うものが一般的であったが、ユビキタスコンピューティングにおける状況の認識とそれに基づく判断は、様々な種類の曖昧性を持った膨大なデータを統合し、そこから現在の状況に対する推論を行い、サービス利用者が望むシステムの挙動を自動的に適切な動作を判断・選択して実行する必要がある。このようなシステムを既存のソフトウェア開発技術を用いて実現するのは困難であり、システム構築コストや拡張性の観点から解決策が求められる。

1.1.3 ユビキタスコンピューティングのためのフレームワークにおける課題

このようなユビキタスコンピューティングシステムを容易に実現するためには、従来型のシステム向けに提供されるフレームワークでは不十分であり、ユビキタスコンピューティングの諸問題を解決できるフレームワークが求められる。ユビキタスコンピューティングでは各機器がユーザの手入力によらずに自動的に制御されるが、これを実現するためには現実世界のコンテキスト (状況) 認識とそれに基づく最適な

制御の決定が必要となり、従来のコンピュータシステムとは異なる情報処理が必要となる。

これまでもユビキタスコンピューティングに適したソフトウェアフレームワークを構築する取り組みはすでに多くなされてきた [10, 5, 30, 1, 13, 29, 20, 40, 38]。しかしながら、これらの全ての取り組みは、ある状況下やある場所においてアクティブとなるシステムが単一であるというモデルに基づいて作られている点に限界がある。すなわち、様々なアプリケーションシステムが同時に動作するという想定のもとでの設計となっていない。このため、これらの提案方式を単純に適用してしまうと、たとえばある場所においてプッシュ型のコンテンツサービスを提供されるとき、そのサービスが動作している間は他のアプリケーションシステムは排他され、動作しないことになってしまう。ユビキタスコンピューティングにおいては、様々なサービスが動的に状況に応じて提供される仕組みが求められることから、このような仕組みでは不十分であると考えられる。

1.2 ユビキタスネットワークアーキテクチャ

そこで、本研究ではユビキタスコンピューティングにおける情報処理・制御の問題を統合的に扱うためのプラットフォームとして、ユビキタスネットワークアーキテクチャを提案する。ユビキタスネットワークアーキテクチャは、ユビキタスコンピューティングを実現するためのネットワークアーキテクチャであり、全体としてはインターネット (IPv6) に基づくオープンなネットワーク上のサービスアーキテクチャとして構築される。

ユビキタスネットワークアーキテクチャの機能は、SOA (Service Oriented Architecture) [2] によってコンポーネント化されて提供される。本アーキテクチャにおけるネットワーク上の構成要素としては以下の要素が挙げられる。

ユビキタスアプリケーションサービス

本研究の提案するユビキタスコンピューティングの実現のためのアーキテクチャは、アプリケーションシステムがインターネット上のサービスとして実現されることを想定している。すなわち、たとえばスマートハウスの実現を考えたとき、それが各家庭内のホームサーバによって制御されるのではなく、インターネット上のサービスとして実現され、そこから制御が行われる方式である。従来のクローズドなシステムによるユビキタスサービスの実現とは異なり、グ

ローバルなネットワーク上にアプリケーションを実現することで、サービス間の連携を行うことやサービスを複数の利用者に対して提供することが可能になるほか、システムの維持管理を中心としたコストを抑えることが可能となる。このモデルに基づき、インターネット上にユビキタスコンピューティングを実現するアプリケーションサービスがホスティングされることとなる。

ユビキタス情報サービス

ユビキタスコンピューティングにおける基本的な情報を管理するための情報サービスであり、様々な人やモノ、あるいはサービス間で用いられる情報を統合して用いるための機能を提供する。

ユビキタスコンピューティングにおいては現実空間における様々な企業や組織によって様々なサービスが提供されることが想定されるが、実用的なユビキタスコンピューティングシステムの実現のためにはこれらの中でのオープンな情報交換メカニズムが必要になる。たとえば、ユーザの個人情報(言語やアクセシビリティに関する情報など)等をオープンな形で管理する仕組みを導入することで、複数のアプリケーションサービスで提供される設定を共通化することが可能となる。また、機器の提供するネットワーク API に関するプロファイル情報を共有データとして利用できるようにすることで、スマートハウスシステムやその他の各種ユビキタスコンピューティングにおけるアプリケーションのすべてからこれらの情報を利用でき、オープンアプローチに基づくシステム全体の統合によるメリットが得られることが分かる。

ユビキタス機器

ユビキタス情報サービスを通して得られる状況情報に基づき、ユビキタスアプリケーションサービスは具体的な制御を行うこととなるが、その際の制御対象となるのが家電を始めとした組込み機器である。本研究のモデルでは、組込み機器がインターネットに接続され、ユビキタス情報サービスへの情報を提供するほかユビキタスアプリケーションから操作されるためのネットワーク通信インタフェースを提供する必要がある。このような機能を持った組込み機器をここでは「ユビキタス機器」と呼ぶ。

本研究においては、ユビキタス情報サービスを実現するための仕組みとしてユビキタス ID アーキテクチャ2.0 を、ユビキタス機器を実現するための仕組みとして uID-

CoAP アーキテクチャを提案している。前者は ucode [22] と呼ばれる一意識別子に基づく情報を統合・集約するためのハイブリッドデータベースシステムとして実現される。後者については、組込みリアルタイムオペレーティングシステムのデファクトスタンダードの一つである T-Kernel [48] 上に、CoAP (constrained application protocol) [47] に基づく RESTful な通信インタフェースの提供のためのソフトウェアフレームワークを提供することで実現した。

1.3 UC フレームワーク

ユビキタスネットワークアーキテクチャにおいて、ユビキタス機器の制御の多くは利用者が意識しない形で自動的な最適制御という形で実現されるが、特にユーザの知覚に大きく関わるユーザインタフェース機器においては、ユーザの意図に合わせた動作の実現が強く求められる。たとえば、スマートハウスのシステムにおいて、ユーザが GUI の操作パネルからテレビの録画予約の操作を行っているときに、お風呂のお湯が沸いた通知がポップアップすることでその操作を阻害されてしまうと、状況によってはユーザにとっては不快な動作となってしまう可能性がある。しかしながら一方では、利用者に対して適切なタイミングで適切に通知を行ったり、災害を含む緊急時に強制的に現在の操作を中止させて避難を喚起するなど、現実世界の状況にあわせた通知を含む自動的な動作の実現も必要になる。

このように、ユビキタスコンピューティングにおけるユーザインタフェース端末においては、現実世界の状況やユーザの意図等を適切に汲みとった上で、それに基づく最適な制御動作を実現することが求められる。制御対象のユビキタス機器についても同様の事が言えるが、よりユーザの知覚に深く関係するユーザインタフェース端末においてはよりその重要度が高いといえる。

この課題を解決するため、本研究ではユビキタスコンピューティングにおけるユーザインタフェース端末のためのソフトウェアフレームワーク「UC フレームワーク」を構築する。UC フレームワークは、主にコンテキスト推論フレームワーク・アプリケーション間調停フレームワーク・省電力制御フレームワークの3つの機能から成り立っており、それぞれの機能に対して与えられる外部ポリシー記述によってその動作ポリシーを定義できるしくみとなっている。

コンテキスト推論フレームワークに与えるコンテキストポリシー記述では、センサ値や環境変数などの一次的なデータ (プライマリコンテキスト) に対し、たとえば「こ

の部屋は暑い」「利用者が画面を注視している」などの、一次的なコンテキストを元に推論される意味を持ったコンテキスト(合成コンテキスト)を導出するためのルール記述が行われ、それに基づくコンテキストの推論が同フレームワークによって実行される。

一方、アプリケーション間調停フレームワーク・省電力制御フレームワークは、前述のコンテキスト推論フレームワークによって推論された現実世界のコンテキストに基づき、それらに対するアプリケーション動作・電力制御におけるアクションをそれぞれ決定するためのメカニズムを提供する。これらのフレームワークに対して与えられる動作ポリシ記述は、それぞれアプリケーション間調停ポリシ記述・電源管理ポリシ記述と呼ばれ、フレームワークからポリシ記述を分離することでシステム開発者が容易にコンテキストに依存した動作を定義し、更新することができるようになっている。

これらのうち、前者のアプリケーション間調停フレームワークは、様々なアプリケーションシステムが動作している時にそれらが個々のアプリケーションとしてではなく全体としてユーザにとって適切な振る舞いを行うよう、アプリケーション間の動作の調停を行う仕組みである。例えば、ある場所で音楽を聞いている時に災害等による緊急通知が行われるとき、その情報を正しく利用者に提供するためには音楽の再生音量を一次的に下げるなどの処理が必要になると考えられる。このような処理は音楽再生アプリケーション内部の情報で閉じて実現できるものではなく、緊急通知を行うアプリケーションにも関連するためシステム全体としての処理が必要となる。ユビキタスコンピューティングシステムにおいてはこのような自動的な制御によるシステム動作が多く求められるが、一方でこのようなシステム動作をアプリケーションレベルで実装してしまうとアプリケーション間のモジュール性が損なわれ、開発効率が低下してしまう。本研究ではこれを避けるため、アプリケーションからは全体動作の実現のために必要な属性情報を提供する方針とし、それに基づく全体動作をアプリケーション間調停ポリシ記述により与えられたルールに従って行うような仕組みとした。この属性情報のことを、本研究では**ビヘイビア** (*behavior*)と呼んでいる。

後者の省電力制御フレームワークでは、複数のコンテキストで示される条件式に対する電力制御処理の内容を列挙した電源制御ポリシ記述に基づく自動的な省電力制御が行われる。コンテキストに基づき自動的な電力制御が行われることで、アプリケーションからの直接的な電力制御が不要となり、これによってシステム開発者

は簡単かつ効率的に省電力制御を実現することが可能になる。ただしコンテキストのみに基づく電力制御では、アプリケーションの状態が無視されることから、本来は望ましくない省電力化処理が行われる可能性もある。このため、同フレームワーク上にはアプリケーションからの必要なサービスレベルを指定し、省電力制御を抑制するための機能を導入している。

1.4 システム実装

UC フレームワークの考え方にに基づき、本研究では実用的なユビキタスコンピューティングシステムである「UC 場所情報システム」を構築した。UC 場所情報システムは、別所らによる先行研究の成果である「ユビキタス空間識別基盤」[56, 6]の空間モデルに基づき、携帯端末上での場所情報サービスを行うシステムである。本研究の成果は現在、上野動物園や浜離宮恩賜庭園を含む複数の東京都内の観光地において実運用されている。

UC 場所情報システムの最大の特徴は、プッシュ型の観光情報サービスを提供することにある。すなわち、ユーザに貸し出された携帯端末が現在いる場所に基づき最適な情報を自動的に利用者に提示するということにある。このような動作により、UC 場所情報システムでは観光地の説明員(ガイド)が観光地内の見どころの情報をその場で説明する動作に近い動作を実現し、それにより利用者は端末操作を行う煩わしさを感じることなく必要な見どころ情報を得ることができる。しかしながら、利用者が不要としている情報や状況に合わない情報を提供してしまうと、利用者の立場からはストレスを感じる結果となる。例えば、歩行者ナビゲーション機能を用いてトイレへに急いでいるときに、観光地の見どころ情報が通知されるような事象は一般には望まれない動作である。このため、UC 場所情報システムにおいてはセンサ値などから得られる物理的な状況、そこから推定される利用者の状況、さらにアプリケーションやシステム内部の状態を含めた制御を行う必要があり、前節で示した UC フレームワークのケーススタディとして適切なアプリケーションシステムであると考えられる。

UC 場所情報システムでは前述の UC フレームワークの考えに基づく機能をコンテキストマネージャ・アプリケーションマネージャの2つのシステムサービス(デーモンプロセス)として実現している。コンテキストマネージャはコンテキスト推論フレームワークで定義される機能そのものを提供するシステムであり、アプリケーショ

ンマネージャはアプリケーション間の調停に基づく表示動作を行う機能を備えたウィンドウマネージャである。UC 場所情報システムにおけるプッシュ通知を適切なシステム動作として実現するためには、アプリケーション間のウィンドウのポップアップに関する調停が必要であることから、アプリケーションマネージャは各アプリケーションのウィンドウの表示制御を与えられた調停ルールに基づき実現している。このほか、アプリケーションマネージャは省電力制御フレームワークの機能も内包したもとなっている。

1.5 本研究の成果

本研究では、ユビキタスコンピューティングのユーザインタフェース端末上のサービスを容易に実現できるようにするため、以下の要件を満たすことを目指してソフトウェアフレームワークの構築を進めてきた。

1. 自動的な動作
2. 高い開発効率
3. システム全体としての最適動作

まず、1. の自動的な動作についてはユビキタスコンピューティングの定義によるものであり必須であるが、ここで問題となるのが 2. と 3. である。高い開発効率とシステム全体としての最適動作の両立は、これまでに提案されている方式では実現できなかった課題である。高い開発効率を実現するためには、アプリケーションやシステムコンポーネント間の独立性を向上させ、それぞれがモジュールとして独立して機能するようにしなければならない。一方で、システム全体としての最適な動作を実現し、利用者にとって最適なサービスを提供するためには、各アプリケーションがそれぞれ自由に動作するシステム動作モデルでは不十分であり、アプリケーションレベルでシステム全体の挙動を含んだ動作を考慮してシステムを構築する必要があった。

本研究の最大の成果は、これらの 3 つの要件を同時に満たすことのできる仕組みを提供したことにあるといえる。本研究では、アプリケーションごとの独立性を確保した上でシステムとしての全体動作を実現するために、新たにビヘイビアの概念を導入し、それに基づく調停に基づくアプリケーション間の調停を行う方式を提案

した。この方式では各アプリケーションはシステム全体動作を気にする必要はなく、各制御対象に対して取り扱いのためのメタ情報を付加するだけでよい。それらに基づく制御はアプリケーション間調停フレームワークにより実現され、アプリケーション動作とは明確に分離されている。また、アプリケーション間調停に関する規則をはじめ、コンテキストの定義や電力制御のルールも、外部ポリシ記述として分離されていることで、高い開発効率での処理を実現することが可能である。このような、アプリケーション間調停を中心としたフレームワークの構成、ならびにそのためのポリシの外部記述化が本研究の最大の成果である。

また、前述の方式の提案に基づくソフトウェアフレームワークを実際に構築し、その上で実用システムを完成させたことも本研究の大きな成果である。今回実装を行ったシステムでは、コンテキスト推論フレームワークに対応するコンテキストマネージャ、アプリケーション間調停フレームワークおよび省電力制御フレームワークの考えに基づくウィンドウマネージャであるアプリケーションマネージャが実装されており、その上で実用的なユビキタスコンピューティングの応用例である UC 場所情報システムが実現された。UC 場所情報システムは、東京都内の複数の観光地において実用サービスとして実運用されており、高い評価を得ている。同システムについては、過去に似たサービスを提供していたシステムとの比較を含む様々な観点から評価が行われ、UC フレームワークとそれに基づく UC 場所情報システムの有用性を裏付けるものとなっている。

1.6 論文の構成

本論文は以降、以下のように構成される。第 2 章では、本研究に関連する研究および技術を整理し、本研究の位置づけを明確にする。続く第 3 章では、本研究の前提となるユビキタスネットワークアーキテクチャの概要を示し、UC フレームワークの全体システムにおける位置づけを示す。第 4 章では、UC フレームワークの論理的な構成についての具体的な提案を行い、その構成要素について説明を行う。そして第 5 章では同フレームワークによって構成される UC 場所情報システムの実装について解説し、その次の第 6 章でその評価を行う。最後に第 7 章で、評価に基づく本研究の総括を行う。

第 2 章

関連研究・技術

これまでもユビキタスコンピューティングを効率よく実現するための手法や、それに基づくアーキテクチャの構築に関する研究は多くなされている。本章では、同分野に関する研究を主に以下の観点から俯瞰する。

- ユビキタスコンピューティングのためのアーキテクチャ
- ユビキタスコンピューティングのための省電力制御技術

2.1 ユビキタスコンピューティングのためのアーキテクチャ

ユビキタスコンピューティングシステムを効率よく実現するためには、それを実現するためのネットワークサービスやソフトウェアフレームワークが不可欠である。ユビキタスコンピューティングを実現するためのアーキテクチャはすでに多数提案されている。

2.1.1 Gaia アーキテクチャ

Cerqueira らは、ユビキタスコンピューティングにおけるアプリケーションシステム開発を容易にするためのプラットフォームとして Gaia [10] を提案している。ユビキタスコンピューティング環境においては、サービスを実現するデバイスは場所や状況に応じて変化するため、これまでの静的なアプリケーションモデルとは異なるシステムが必要であるとしている。アプリケーションが用いる出力デバイスを例に挙げると、環境によっては壁面ディスプレイが用いられることがある一方で、PDA の画面が用いられる場合もある。場合によってはスピーカによる音声出力が用いら

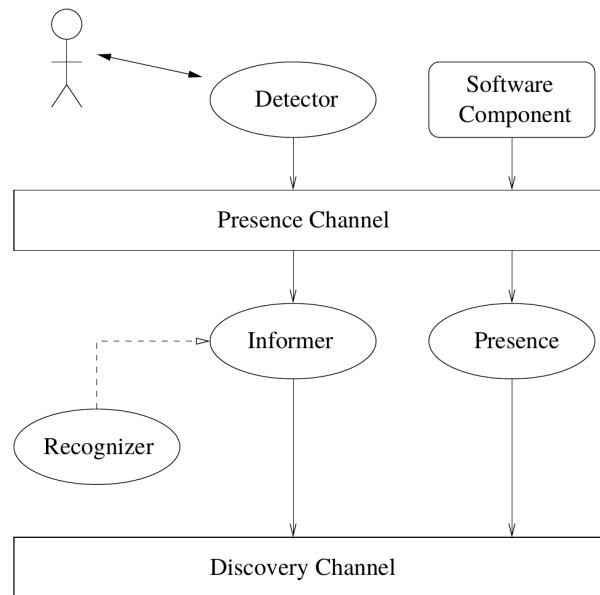


図 2.1: GaiaOS におけるディスカバリサービス ([43] より引用)

れる可能性もあるだろう。使われるデバイスや方法は環境中で利用可能なリソースやセキュリティ・プライバシーによる制約により変化する。システムを実行する環境のこのような変化に適応できるコンピュータシステムを実現することを目標に彼らは Gaia を提案している。

Gaia 提案の中核をなすコンポーネントは、GaiaOS [43] および MPACC フレームワーク [42] である。前者の GaiaOS は、既存のオペレーティングシステムの上に実現されるミドルウェアシステムであり、CORBA に基づくユビキタスコンピューティングのための基本的なシステムサービスを提供する。この基本サービスには、システムコンポーネント間の情報の伝達を行うためのイベント管理機能、環境中に存在する機器や人やその他のオブジェクトを追跡するためのディスカバリサービス、動的环境上におけるファイルシステム機能を提供するデータオブジェクトサービス、セキュリティに関する各種サービスが含まれる。ディスカバリサービスは図 2.1 に示すような構成となっており、ディスカバーチャンネルを通して常に環境中のリソースに対する変化を他のシステムコンポーネントやアプリケーションに伝える役割を担っている。

本研究の関連研究としてより重要な部分が、後者の MPACC (model-presentation-

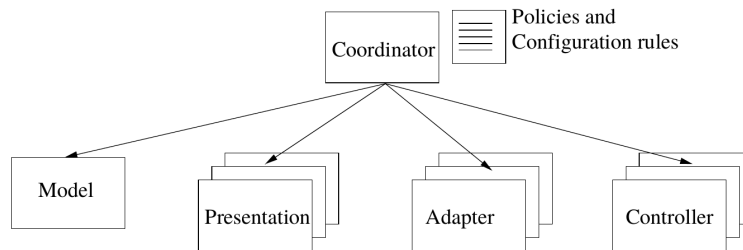


図 2.2: Gaia MPACC フレームワーク ([43] より引用)

adapter-controller-coordinator) フレームワークである。これは、環境中で利用可能なリソースやセキュリティ・プライバシーによる制約により変化に対してアプリケーションを柔軟に構成するためのフレームワークであり、その構造は図 2.2 に示すとおりとなっている。

MPACC は一般のサービスアプリケーション開発の際に用いられる MVC (model-view-controller) を拡張したモデルであり、コーディネータ (coordinator) と呼ばれる要素によって MVC のモデルを除く各階層の機能を実現するコンポーネントを選択するというのが本質的な考え方となっている。たとえば、ユーザの存在する環境内に表示デバイスがある場合には表示によるビューの実装を、表示デバイスがない代わりにスピーカが備わっている時には音声によるビューの実装を選択して動作させるということが出来る。

Gaia フレームワークにおけるこのような切り替え処理は、コーディネータに与えられる外部ルールで実現される。Gaia フレームワークにおけるこの外部ルールは ACD (Application Customized Description) と呼ばれ、Lua の文法によって記述される。ACD の記述例は図 2.3 に示すとおりである。この例ではジュークボックスアプリケーションがとりうる 2 つの構成を与えたものとなっており、左が会議室での動作シナリオ、右が家庭内での動作シナリオを各層のコンポーネントの組み合わせで示したものとなっている。

Gaia における ACD によるアプリケーション構成の外部化については、本研究におけるコンテキスト推論・アプリケーション間調停・省電力制御の外部ルール化と同様、ポリシーとメカニズムの分離という考え方に基づいており、我々の提案とその点では同じ方向性に基づいていると言える。しかしながら、Gaia で提案しているのは一つのアプリケーションの中の構成を MVC モデルに基づき分離して、それらを

<pre> Application = { Model = {{ ClassName = "JukeboxModel", Hosts={{ "amr1.as.edu"}}, }} Presentation = {{ ClassName = "MusicPlayer", Hosts={{ "amr2.as.edu"}} }}, Controller = {{ Classname = "ListViewer", Hosts={{ "plasma1.as.edu"}, {"plasma4.as.edu"} }, }}, Controller = {{ Classname="VCRController", Hosts={{ "pda.as.edu"}}, }}, Coordinator = {{ ClassName = "Coordinator", Hosts={{ "amr3.as.edu"}}, }}, } parseApplication(APPLICATION, "mroman") </pre>	<pre> Application = { Model = {{ ClassName="JukeboxModel", }} Hosts={{ "livingroom.as.home"}} }, Presentation = {{ ClassName="MusicPlayer", Hosts={{ "livingroom.as.home"}} }}, Controller = {{ ClassName = "ListViewer", Hosts={{ "pda.as.home"}} }}, Coordinator = {{ ClassName="Coordinator", Hosts={{ "livingroom.as.home"}} }}, } parseApplication(APPLICATION, "mroman") </pre>
--	--

図 2.3: ACD によるアプリケーション構成の記述例 ([41] より引用)

環境に合わせて組み替えることで様々な環境に対応する柔軟性を実現するというものであり、本研究の提案とは内容的には独立したものである。したがって、両者を組み合わせてより柔軟なユビキタスコンピューティングのためのアーキテクチャを構築することも可能である。

2.1.2 SCaLaDE アーキテクチャ

Bellavista らにより提案された SCaLaDE (Services with Context awareness and Location awareness for Data Environments) アーキテクチャ[5] は、ユーザの場所とコンテキストに依存するサービスの実現を容易にするためのミドルウェアアーキテクチャである。

SCaLaDE の最大の特徴は、モバイルエージェント (MA) とよばれるソフトウェアエージェントに基づくシステムの実現にある。一般的に、ユビキタスコンピューティングサービスをクライアント・サーバモデルで実現した場合、サーバによって提供される固定的なサービスがユーザからはアクセス可能となる。しかしながら、ここで例えばスマートハウスにおいて新たな家電を買ってきて接続した時、スマートハウスサービスを提供するサーバ上でのサービスが固定化されているとサービスの実現が困難となる。この問題を解決するため、SCaLaDE ではモバイルエージェント (MA) と呼ばれる移動可能なソフトウェアコンポーネントを導入し、そのサービスが移動先の環境に導入されて実行されるモデルに基づくシステム構築を行った。前述のスマートハウスの例においては、家電が家庭内のネットワークに接続された瞬間に家電制御用の MA が家庭内のスマートハウス管理システムにコピーされ、同家電に対する制御がスマートハウスシステムに追加される。このようにして、ユビキタスコンピューティングのサービスを動的に更新できるのが MA によるアプローチの特徴である。

さらに、SCaLaDE では MA プロキシもサポートしている。MA プロキシとは、MA を動作させるローカルドメイン間でのセキュアなプロキシ通信を行うための仕組みである。ここでは図 2.4 に示す美術館ガイドシステムを例に挙げて説明する。ユーザがある美術館 (Museum1) から他の美術館 (Museum2, 3) に移動した時に、Museum2, 3 に移動後も Museum1 での登録デスクでのユーザ登録の情報をそのまま使い回したいケースが考えられるが、これは MA の動作ドメインを超えているのでそのままでは実現できない。そこで、Museum1 で動作する MA エージェントに対するプロキシ

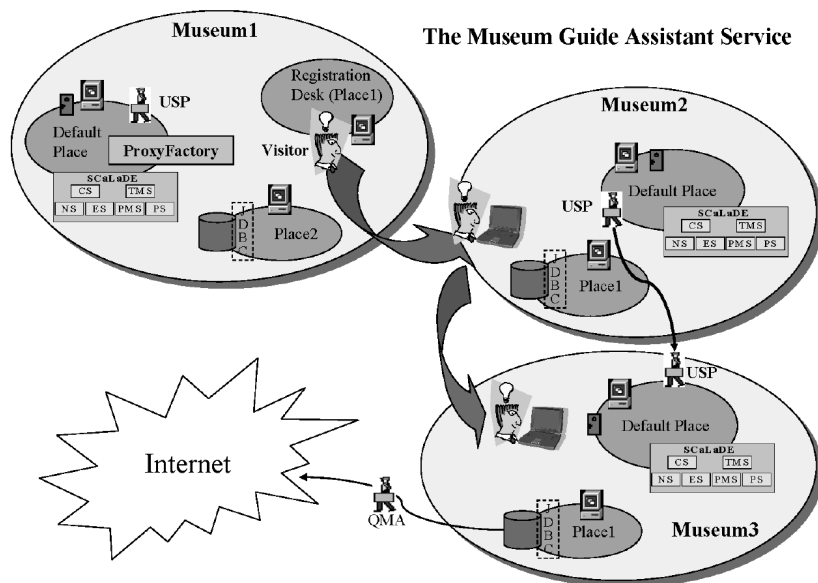


図 2.4: 美術館ガイドシステムにおける MA プロキシの動作 ([5] より引用)

を導入することで Museum1 内のドメイン内のデータサービスにアクセス可能な仕組みを実現している。

これらの MA に基づくユビキタスコンピューティングシステムのためのフレームワークが ScaLaDE アーキテクチャであるが、その全体構造は図 2.5 に示す通りとなっている。本研究で提案する UC フレームワークと同様に、外部記述である ScaLaDE メタデータによって MA の自動的な管理操作を実現している。ScaLaDE メタデータは大まかにはプロファイルとポリシーに分類され、前者ではユーザやデバイスの機能や属性、プロファイル情報を扱う。後者は Ponder [15] と呼ばれるプログラミング言語に基づくルール記述により MA の挙動を決定するものである。図 2.6 の例における a), b) がそれぞれプロファイルおよびポリシーの記述例となっている。

ScaLaDE で実現するユビキタスコンピューティングのアーキテクチャは本研究の提案方式とは大幅に異なっており、外部ポリシー記述を用いるアイデア自体には共通性もあるがその内容は全く異なるものである。ただし、ScaLaDE のような MA ベースのシステムを実現した場合にも、本研究におけるアプリケーション間調停の仕組みは役立つことが期待できる。複数の MA が一つのドメイン内で動作するとき、それらが全体として理想的な振る舞いを実現することは MA の増加と共に困難

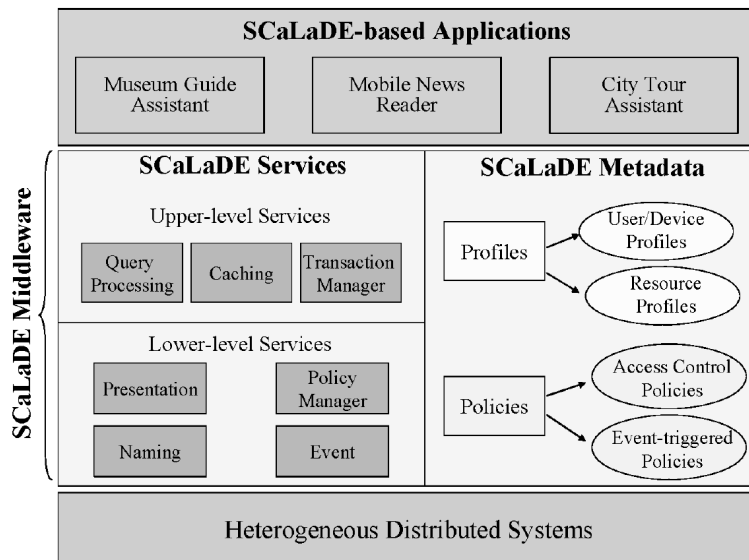


図 2.5: SCaLaDE アーキテクチャ ([5] より引用)

となると推測できる。本研究で提案するアプリケーション間調停の仕組みを MA に対して導入することで、MA ベースのユビキタスコンピューティングにおける全体最適動作を実現することが可能である。

2.1.3 MIMOSA フレームワーク

Malandrino らによって提案された MIMOSA フレームワーク [30] は、ユビキタスコンピューティングにおけるウェブアクセスの際のコンテキストアウェアネスを実現するためのフレームワークである。全体的なアーキテクチャ構成は図 2.7 に示すとおりとなっており、クライアントからは、MIMOSA フレームワークを通して現在のコンテキストに基づくウェブサービスへのアクセスが実現される。

MIMOSA フレームワークにおけるコンテキストの扱いは、CARE (Context Aggregation and REasoning) ミドルウェア [7] によって実現されている。CARE ミドルウェアはコンテキストの変化をトリガとして、ポリシーとして与えられた処理を行う仕組みを持っている。ポリシーの定義例は図 2.8 に示すとおりとなっており、プログラマ的にコンテキストに基づくシステム動作ポリシーの決定を行うことが出来るようになっている。



図 2.6: SCaLaDE メタデータの例 ([5] より引用)

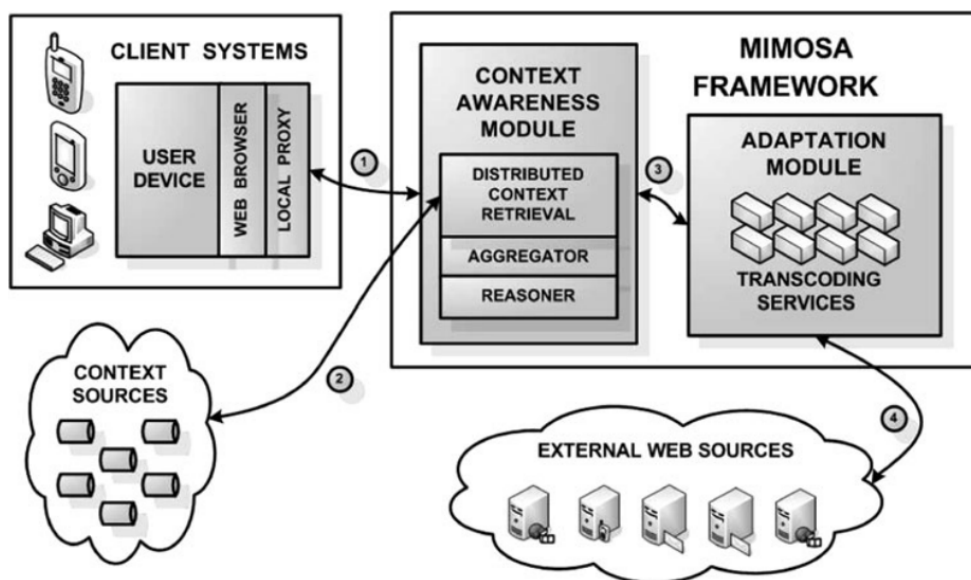


図 2.7: MIMOSA アーキテクチャ ([30] より引用)

- R1:** “If AvBandwidth $\geq 128kbps$ And Bearer = ‘UMTS’
Then Set NetSpeed=‘high’”
- R2:** “If NetSpeed=‘high’ And AvMem $\geq 4MB$
Then Set MediaQuality=‘high’”
- R3:** “If NetSpeed=‘high’ And AvMem $< 4MB$
Then Set MediaQuality=‘medium’”
- R4:** “If NetSpeed!=‘high’ Then Set MediaQuality=‘low’”

図 2.8: CARE ミドルウェアにおけるコンテキスト処理のポリシー記述 ([7] より引用)

コンテキストに対応するシステム動作をポリシー記述によって行うという考え方は、本研究で提案する省電力制御フレームワークの仕組みに近い。ポリシーの定義が CARE ミドルウェアではプログラマ的になっているのに対し、省電力フレームワークによって実現されるポリシー定義は各ターゲットに対するパターンマッチによるルールの列として定義されている点異なるが、それを除けば同一の仕組みであるといえる。しかしながら、MIMOSA フレームワークおよび CARE ミドルウェアにおけるコンテキストは、本研究で扱うコンテキストのうちプライマリコンテキストのみである。本研究での提案の一つの主要な要素には合成コンテキストを外部ルールにより定義し、それを用いることで開発効率やシステムの記述性を向上させるという点において本研究の優位性が認められる。また、本研究のような複数のアプリケーションを制御するための仕組みを持っているわけでもない。

2.1.4 LOCA フレームワーク

Ahn らにより提案されている LOCA フレームワーク [1] は、ユビキタスコンピューティング環境におけるコンテキストウェアなアプリケーションを実現するためのウェブサービスフレームワークである。全体的には SOA (service oriented architecture) に基づくアーキテクチャとなっている。

このフレームワークにおいて最も興味深い仕組みとしては、LOCA サービスブローカと呼ばれるサービスが挙げられる。このサービスはユビキタスコンピューティング環境において提供する複数のサービスの中から、利用者側の要求に適したサービスを仲介して提供するための機構である。ブローカが適切なサービスを検索するメカニズムは、以下の3つのステップにより成り立っている。

Step 1: カテゴリマッチ

サービスカテゴリのオントロジ情報 (主にサブクラス情報) に基づき、利用者側の要求するサービスに適したサービスカテゴリを検索する。

Step 2: ロケーションマッチ

Step 1 で検索されたサービスカテゴリのうち、ユーザの場所に関するオントロジ状況 (同様に主にサブクラス関係) に基づき、その場所に関するサービスを提供できるサービスプロバイダを検索する。

Step 3: ネットワークマッチ

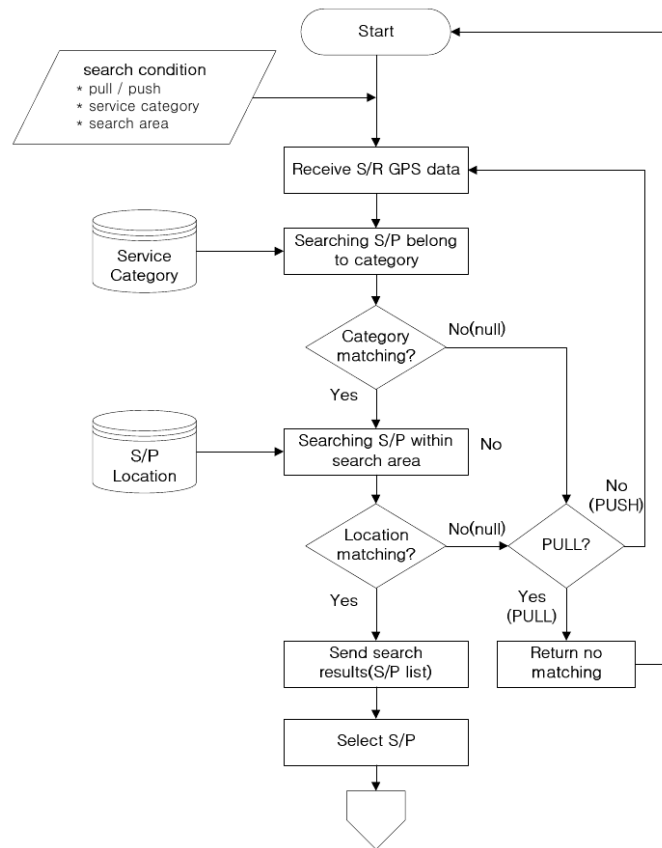


図 2.9: LOCA におけるカテゴリマッチ・ロケーションマッチの処理手順 ([1] より引用)

Step 2 で得られたサービスプロバイダに対し、ユーザとサービスプロバイダ間のネットワーク接続環境に関する検索を行う。

上記処理手順の詳細は、図 2.9 (Step 1 および 2)、図 2.10 (Step 3) に示すとおりである。

LOCA フレームワークにおけるサービス実現モデルは、複数のサービスをユーザに提供する仕組みという点で本研究と似た特徴を持っている。LOCA サービスブローカに相当するものが、UC フレームワークではアプリケーション間調停フレームワークとして実現される。このように位置づけとして似た特徴は有しているものの、LOCA におけるアプリケーション挙動モデルはその場所に対する検索で見つかったサービスのうち最もマッチ率が高いものを 1 つ選択し、それを単純に提供する機能にすぎ

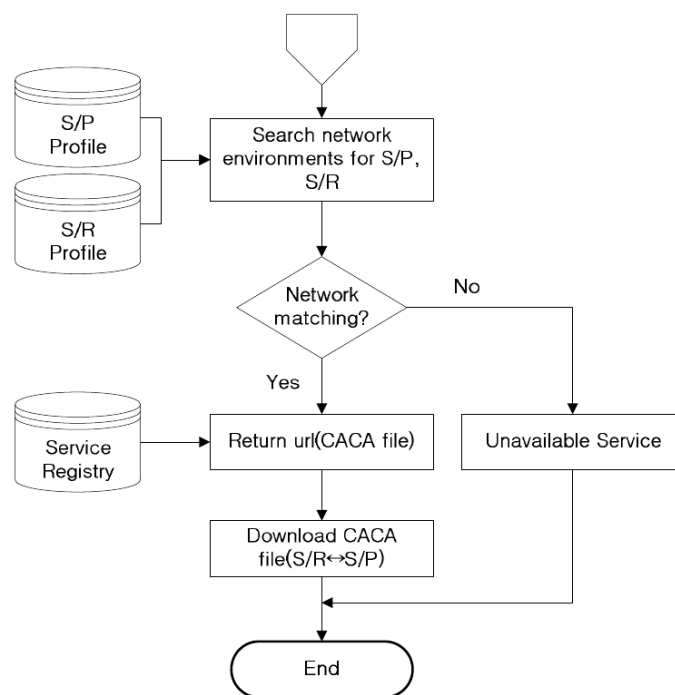


図 2.10: LOCA におけるネットワークマッチの処理手順 ([1] より引用)

Context rules	FOPL formula
CR1	$\text{Pr}(\text{IsStatus}(\text{Camera})=\text{CLOSEUP_VIEW} \text{Action}(\text{Tearcher})=\text{WRITING}, \text{IsStatus}(\text{MediaBoard})=\text{TRUE})=0.8:\text{OnTouched}(\text{MediaBoard})=\text{TRUE}$
CR2	$\text{Pr}(\text{IsStatus}(\text{Camera})=\text{CLOSEUP_VIEW} \text{Action}(\text{Tearcher})=\text{SHOWING})=0.15:\text{OnTouched}(\text{MediaBoard})=\text{TRUE}$
CR3	$\text{Pr}(\text{IsStatus}(\text{Camera})=\text{CLOSEUP_VIEW} \text{Action}(\text{Tearcher})=\text{SPEAKING})=0.05:\text{OnTouched}(\text{MediaBoard})=\text{TRUE}$

図 2.11: 一階確率論理 (FOPL) に基づく推論規則の定義例 ([38] より引用)

ない。UC フレームワークが提供する機能は、例えば同一の場所においてブラウザによるプッシュ型のコンテンツ通知が行われたり、ナビゲーション中にルートから大きく外れた場合に地図アプリケーションがその旨通知を行うなど、その時の外界のコンテキストやシステム内部の状態に応じてアプリケーションの操作を切り替えることができるものであり、より高い一般性を実現したものと言える。

2.1.5 その他のフレームワーク

Küpper らは、デバイスを中心とした場所情報システムのフレームワークとして Trax フレームワーク [13] を提案している。このフレームワークでは、場所情報に基づくサービスの実現を容易にするための API をミドルウェアによって定義するものである。このようなフレームワークは OpenGIS ローケーションサービス (OpenLS) [29]、Nexus プラットフォーム [20]、MiddleWhere [40] など多数提案されているが、いずれも場所情報に関して固定化された機能を提供するものであって、本研究で提案するような高次のコンテキストに対する処理を可能とするものではない。

Weijun らはオントロジモデルに基づくコンテキストウェアネスを実現するためのアーキテクチャを提案している [38]。このミドルウェアでは、一階確率論理 (FOPL) に基づく確率モデルによりコンテキストの推論を行う方式となっている。その記述例は図 2.11 に示すとおりとなっている。この定義は本研究におけるコンテキスト推論ルールの記述に相当するが、一階確率論理に基づく推論に基づくため計算量的な問題が生じる。実際に、同論文に示されたパフォーマンス測定結果について、5000 トリプルのデータに対して 3.2GHz の CPU でのコンテキスト推論に対し 6 秒以上の時間を要していることが読み取れる。また、このフレームワークはあくまでコンテキストの推論のみを対象としており、その上位のアプリケーションを容易に実現するためのその他の仕組みは提供されていない。

2.2 ユビキタスコンピューティングのための省電力制御技術

本節では、ユビキタスコンピューティングにおけるシステムの省電力化を実現するための関連研究に関する整理を行う。

コンピュータシステムの省電力化は古くからの課題であり、デスクトップシステムから組み込み機器、ハードウェアからソフトウェアに至るまでの各レベルにおいて多くの取り組みが行われている。その中でもハードウェアレベルでの省電力化機能の実現については特に研究が進んでおり、power gating や clock gating [53] にみられるように、各デバイスへの電源やクロックの供給をコントロールし、それに基づく省電力化が実現できるようになっている。このようなハードウェア機能に基づく省電力化を実現するためには、これらの機能を活用して省電力化を実現するためのソフトウェアが必要であるが、従来の方式ではオペレーティングシステム・デバイスドライバレベルでの電力制御が一般的であった。オペレーティングシステムやデバイスドライバでの電力制御を行う場合、アプリケーションからの直接的な電力制御を行う必要がなく、開発効率的な観点からは理想的な方式であるといえる。

一方で、電力リソースの限られた組み込み機器においては、システム全体の動作に関連したよりアグレッシブな電源制御が必要となる。たとえば、スマートフォンにおいてはLCDバックライトを適切なタイミングでOFFにする省電力化処理が一般的に行われているが、ユーザがスマートフォンのGUI操作を行っているときにこのような省電力制御処理が行われてしまうと画面が暗くなってしまいユーザの動作が阻害されることとなる。バックライト制御についてはタッチパネルの無操作時間など、ある程度の一般的な省電力制御のメカニズムが確立されつつあるが、ユビキタスコンピューティングにおける省電力化制御においてはこれに限らない様々なデバイスや状況を考慮する必要が生じる。このため、より一般的なコンテキストに基づく省電力化技術の確立が必要である。

そこで、以下の各節では特にコンテキスト情報に基づく省電力化を中心とした関連研究・関連技術について整理する。

2.2.1 Nishihara らによる省電力化手法

Nishihara らはスマートフォンを始めとする携帯端末上でのコンテキストに基づく省電力化手法を提案している [34]。この研究では、コンテキストを用いた省電力制御を実現する方式であるということから我々の研究との関連性が高いこと、全体

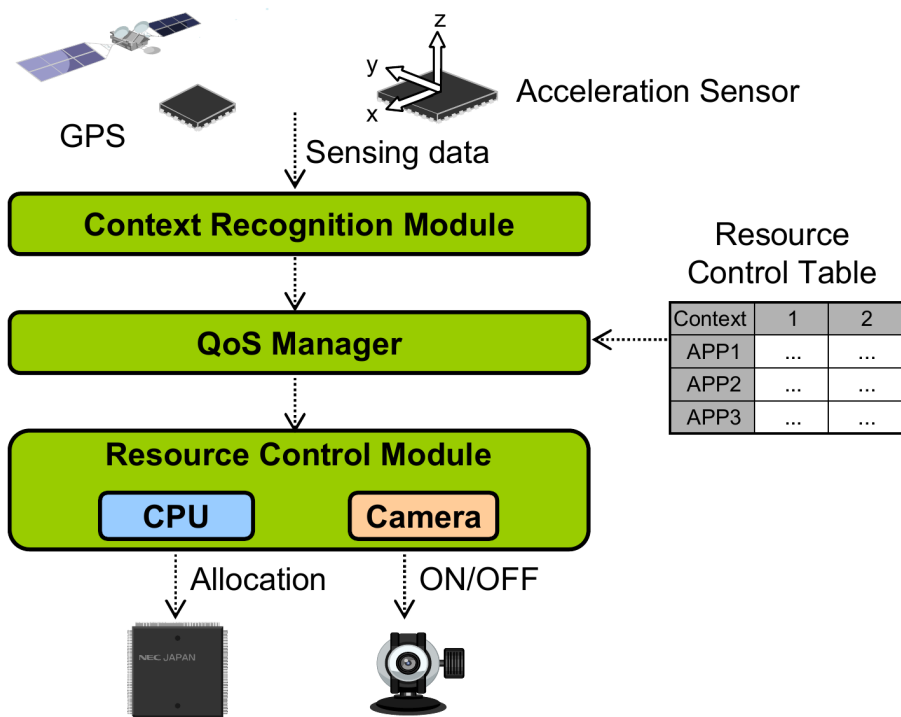


図 2.12: Nishihara らによる提案システム ([34] より引用)

で 45%の省電力化を実現しているなど成果が大きい点から本節でその詳細を取り上げる。

Nishihara らによる提案手法においては、必要とされるサービスのクオリティ(QoS)を実現しながら消費電力を抑えること、その際にアプリケーションの修正を必要としないことを目標としている。これを実現するための仕組みは図 2.12 に示すとおりとなっており、その処理は以下の 3 ステップにより構成されている。

Step 1: コンテキスト認識

最初のステップとしては コンテキスト認識モジュールが GPS と加速度センサからの情報に基づき、利用者が屋内にいるか否か、歩いているか否かを判定する。ユーザが屋内にいるか否かは GPS 信号の受信強度によって判定し、歩いているか否かは加速度センサの値が 3 秒ごとに 3 回以上 1.2G を超えるか否かによって判定する。

このメカニズムによるコンテキストの推定については実験の結果、屋内にいる

か屋外にいるかの判定で 96%・歩行しているか静止しているかの判定で 97%の精度での推定が可能であることが確認された。

Step 2: リソース割り当ての決定

次に、得られたコンテキストに基づきどのようにデバイス等のリソースを割り当てるかに関する決定を行う。この処理は、QoS マネージャによって、リソースコントロールテーブルと呼ばれるあらかじめ定義されたテーブルを参照することで行われる。リソースコントロールテーブルは、各アプリケーションについて各コンテキストに対してどのようなリソースの割り当てを行うかを示したテーブルであり、その値は資源に応じてバイナリ値 (ON および OFF) または割合を示す連続値 (0 から 1) によって表現される。

Step 3: リソースの制御

Step 2 で得られたリソース割り当て結果に基づき、各リソースに指定された QoS を実現するために必要な省電力制御を実行する。

彼らはこのメカニズムに基づく省電力制御メカニズムを実装し、表 2.1 に示すリソースコントロールテーブルを定義した場合の消費電力量に関する評価を行った。結果、何も電力制御を行わない場合と比べ 45%の消費電力の削減が実現できることを示した。

このように、彼らの提案手法ではシステムで定義済みのコンテキストを用いて、それに対応する QoS を指定することで省電力を実現することができるようになっていく。リソースコントロールテーブルの内容を書き換えることで、今回の例では示されていない他のコンテキストに基づく電力制御処理も実現できる。しかしながら、本方式で扱うコンテキストはシステムであらかじめ定義されたものでなければならず、本方式ではコピキタスコンピューティングにおける様々なコンテキストを扱うための柔軟性・拡張性に欠けている。また、リソースコントロールテーブルでのコンテキストの列挙による方式は、コンテキスト数の増加に伴いテーブルサイズが増加してしまう点に問題がある。一般に、表 2.1 のようなテーブルを構築する場合、取り扱うコンテキスト数に対して指数関数のオーダでテーブルサイズが増えてしまう。これは、システムの空間的な計算量コストはもちろん、システム開発者の開発コストの観点からも望ましくない。これに対し、本研究で提案する手法においては、合成コンテキストにより高次コンテキストを定義するための仕組みを有していること、省

表 2.1: リソースコントロールテーブルの定義例 ([34] より引用)

Context	Indoor, Walking	Indoor, Stopped	Outdoor, Walking	Outdoor, Stopped
Video	GPS: × Cam: × Proc: 0.5	GPS: × Cam: × Proc: 1	GPS: × Cam: × Proc: 0.5	GPS: × Cam: × Proc: 1
Face Recog.	GPS: × Cam: × Proc: 0	GPS: × Cam: × Proc: 0	GPS: × Cam: ○ Proc: 1	GPS: × Cam: ○ Proc: 0.5
Location Info.	GPS: × Cam: × Proc: 0	GPS: × Cam: × CPU: 0	GPS: ○ Cam: × CPU: 1	GPS: × Cam: × CPU: 1
AR	GPS: × Cam: ○ Proc: 1	GPS: × Cam: × Proc: 0	GPS: × Cam: ○ Proc: 1	GPS: × Cam: ○ Proc: 0.5

電力制御ポリシー記述がパターンマッチをベースとしたものとなっていることで、大幅にこれらのコストを下げる事が可能となっている。

2.2.2 SeeMon フレームワーク

Kang らにより提案された SeeMon フレームワーク [24] は、携帯情報端末などの多数のセンサを持ったリソースの限られたシステム上でコンテキストの監視を行う際の省電力制御を実現するためのフレームワークである。このフレームワークの全体的なメカニズムは図 2.13 に示すとおりであり、センサに基づくコンテキスト情報の問い合わせ内容や得られたセンサデータの内容に基づき、それをエネルギー効率を考慮したセンサの制御にフィードバックさせるという仕組みになっている。エネルギー効率の最適化を実現する問題は、同論文では Minimum Cost False-query covering Sensor Selection Problem (MCFSS 問題) として定式化されており、その問題に対する貪欲アルゴリズムに基づく近似解を解くことでエネルギー的に最も効率的な制御を行うような仕組みとなっている。

SeeMon フレームワークはコンテキストに基づく省電力制御を行うためのフレームワークであるが、その全体的なメカニズムは本研究で提案するものとは全く異なっている。SeeMon では省電力制御がアプリケーションからの問い合わせにおいて与えられた要件に基づいて行われることを想定している。利用者の位置座標を取得する API を例に挙げると、SeeMon では現在の位置座標を単純に取得する API を提供す

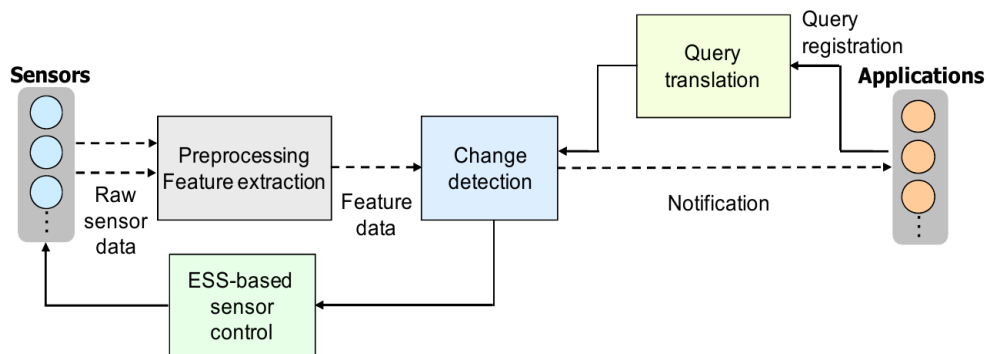


図 2.13: SeeMon フレームワーク ([24] より引用)

る代わりに、現在の位置座標をどれくらいの精度でどれくらいの更新頻度で提供して欲しいかをアプリケーションから指定するというアプローチになる。本研究で提案する省電力制御フレームワークでは、アプリケーションでこのような情報を細かく指定する代わりに、現在のコンテキストを用いてある程度省電力制御を自動化するという考え方に基づいている。アプリケーションからの省電力制御禁止機能はあくまでそれに対する補助的な位置づけである。この点で、SeeMon フレームワークは本研究の考え方とは大きく異なるが、本研究における省電力制御フレームワークの制御対象を SeeMon が要求する QoS を与えるようにし、省電力制御フレームワークを通して SeeMon のメカニズムが動作する構成は両者の優れた点を活かすことができ、有用性が高いと考えられる。

なお、SeeMon フレームワークと同様のアプローチに基づく研究としては、ほかに Murao らによる研究 [33] も挙げることができる。

2.2.3 その他のコンテキストに基づく省電力化手法

現在のコンテキストを省電力制御に利用するという考え方自体は新しいものではなく、本研究以前にもいくつかの提案がなされている。

Harris らによって提案された省電力化手法 [17, 18] においては、オフィス環境におけるセンサを用いてデスクトップ PC の省電力制御を最適化している。具体的には、PC のディスプレイがどのタイミングで使われているかを推定するために、ユーザの大まかな位置を Bluetooth で認識した上で、詳細な状況の推定を行うためにマイク

ロフォンとウェブカメラを用いている。これによく似たものとして、Dalton らによる研究 [14] も挙げることが出来る。

Kjærgaard らは、加速度センサを用いてユーザが移動しているか停止しているかを推定し、GPS 位置情報の更新頻度をそれに基づき制御することで、位置情報精度と消費電力のバランスを最適化する手法を提案している [25]。このシステムに対する評価は住宅街の歩行者追跡システムによって評価され、一定時間ごとに GPS 位置情報の更新を行う場合に対して、100 メートルの誤差精度においては 62.3%、200 メートルの誤差精度では 69.7%の省電力化を実現した。

Küpper らは、位置情報を用いたコミュニティ内サービスにおいて、ユーザ間の距離に基づく必要な位置情報精度を計算し、それに合わせて最低限な位置情報精度を実現することで携帯情報端末における消費電力の削減を行った [27]。

このほかにも、多数のコンテキストを用いた低消費電力化のための手法やフレームワークが提案されている [21, 54, 37, 44, 17] が、これらはいずれもシステムレベルでの電力管理を一定の決められた仕組みに従って行うものである。本研究は、コンテキストポリシ記述・省電力制御ポリシ記述として外部化されたルールに基づく電源制御を行う点に本質がある。関連研究ではある一定のデバイス構成のもとでのコンテキストに基づく省電力化を実現するというものであり、ユビキタスコンピューティングシステムにおけるデバイスや環境の多様性に対応したものではない。

第 3 章

ユビキタスネットワークアーキテクチャ

本章では、本研究の前提となるユビキタスコンピューティングを実現するための全体アーキテクチャとなる、ユビキタスネットワークアーキテクチャについて説明する。

3.1 全体構造

本研究で提案するユビキタスネットワークアーキテクチャでは、オープンアプローチによるユビキタスコンピューティングの全体的な統合を目指している。これまでのところ、ユビキタスコンピューティングの技術に基づく様々なアプリケーションシステムが開発されている [12, 19] が、これらはクローズドなアプローチによるものである。例えば、スマートハウスの実現に向けた研究は多数なされているが、旧来の研究では家庭内にサーバを配置し、そこから制御を行う仕組みとなっている。このアプローチは、単一のアプリケーションシステム構築を行う上では問題ないが、多数のアプリケーションや複数対象へのサービスの実現を考えた時のスケーラビリティや拡張性の観点からは適さない方式である。例えば、ある家庭で実現しているスマートハウスを他の家で実現するためには、同じ機能を持ったサーバが必要になる上、複数の家庭で同一のシステムが動作していることから機能追加や管理のためのメンテナンスコストが大きくなる。

図 3.1 に、ユビキタスネットワークアーキテクチャの概念図を示す。図に示されている通り、本アプローチではインターネット (IPv6) に基づくアーキテクチャをとり、ユビキタスコンピューティングのサービスは家庭内のサーバではなくインターネット上にホスティングされる形となっている。図中ではスマートホームのサー

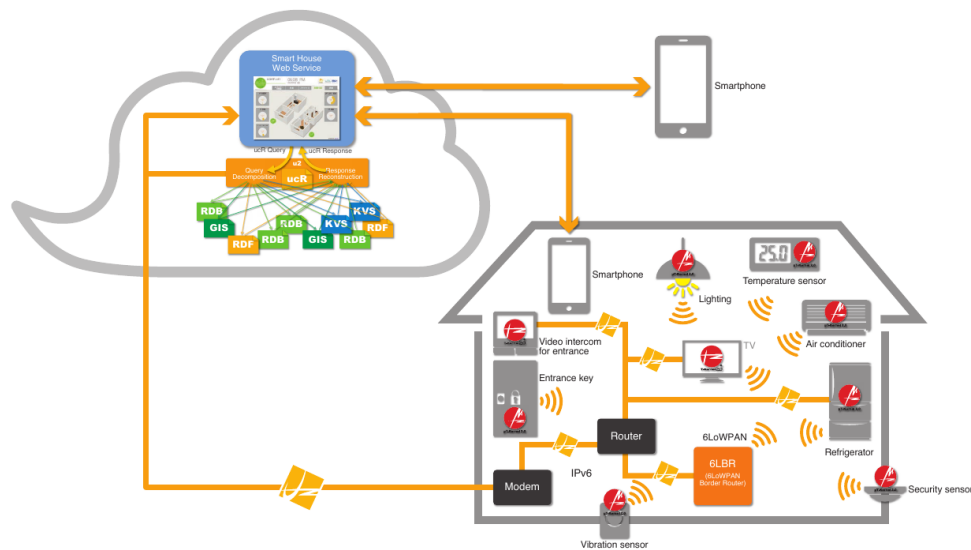


図 3.1: ユビキタスネットワークアーキテクチャ概要

ビスがインターネット上で運営されており、そのサービスを通して家庭内の機器 (冷蔵庫、洗濯機、テレビ、他) が制御される。

インターネット上のサービスは SOA (Service Oriented Architecture) [2] によってコンポーネント化されて相互接続され、サービス間での連携を行うことで全体としてのシステムを実現する方式となっている。これにより、ユーザの個人情報 (言語やアクセシビリティに関する情報など) 等を知識情報データベースで一括管理し、複数のアプリケーションサービスで提供される設定を共通化することが可能となる。例として、個人の言語設定を英語に設定した時、スマートハウスサービスだけでなく、他の各サービス (スマートシティサービス、医療情報システム) などすべてのサービスにおいて個人設定が共有され、オープンアプローチに基づくシステム全体の統合によるメリットが得られることが分かる。

図 3.1 に示されるとおり、インターネット上に実現されたユビキタスコンピューティング (図中ではスマートハウスウェブサービス) に対し、ユーザインタフェースを提供する端末としてはスマートフォンのような小型のユーザインタフェース端末が想定される。本研究の提案の中核をなす UC フレームワークは、このユーザインタフェース端末における機能を実現するためのソフトウェアフレームワークとして位置づけられる。

3.2 ユビキタス ID アーキテクチャ 2.0

ユビキタス ID アーキテクチャ 2.0 (以降、uID アーキテクチャ 2.0 または u2 と記述) は、ユビキタスコンピューティングにおける状況認識の核となる知識や情報を取り扱うためのフレームワークである。

3.2.1 ucode

ユビキタスコンピューティングにおいては、現実世界におけるモノを一意に識別する識別子が重要となる。たとえば、スマートハウスシステムにおいてある家庭内の特定の空調の制御を行うことを考えると、システム内で一意にその特定の空調を特定できなければならない。別の家庭に同じ空調製品があればそれとの区別も必要となる上、製造番号等を用いる方式ではメーカー間での番号体系の不一致による衝突が起きてしまう可能性があり、結果的にどの対象を操作してよいのか判別出来ない。uID アーキテクチャ 2.0 によって管理される知識や情報についても、どの対象に対する情報なのかが明確に識別できなければ、そのデータは役立たないものになってしまう。

このことから、uID アーキテクチャ 2.0 では、一意な固有識別子である ucode を用いて実世界の存在に対する識別を行う。ucode は 128 ビットの固有識別子であり、ユビキタス ID センタによって適切に管理されることで一意性を保証している。ucode に関する規定は ITU-T により国際標準として認められたものであり、標準規格としてグローバルに利用可能な位置づけをもつことが保証されている [22]。

uID アーキテクチャ 2.0 においては、実世界の区別が必要な存在に対して ucode を割り当てる。必要な場合には、RFID タグや 2 次元バーコード等を利用して、対象となる実体から ucode を読み取ることが可能なようにする。家電等のネットワークに接続された機器については、ネットワークアドレスと ucode との対応付けを uID アーキテクチャ 2.0 内で管理することで、ucode の取得が可能となる。

3.2.2 ucode 解決

ucode によって識別される事物に対し、それに紐付けられた情報を取得する処理を「ucode 解決」と呼ぶ。ucode 解決を行うことで、各サービスは対象となる事物に対する情報を取得し、それに応じた処理を行うことができる。スマートハウスシステムの例では、ある個人の属性 (言語設定、アクセシビリティ情報等) を取得する処

理や、住宅内の機器の情報を問い合わせる処理が ucode 解決によって行われ、その情報を用いることで各家庭や各個人に適したサービスを実現することができる。

uID アーキテクチャ 2.0 における ucode 解決のためのサービスは、図 3.2 に示すようなオープンなハイブリッドデータベースシステムによって構成されている。uID アーキテクチャ 2.0 はありとあらゆるデータベースを接続することを想定しており、リレーショナルデータベース (RDB) はもちろん、キーバリューストア (KVS) やグラフデータベース (RDF データベース等)、さらには地理情報システム (GIS) といった特殊用途のデータベースを接続することもできる。これらのデータベースは、世界中の様々な提供者によって提供されるものであり、たとえば電力需給情報といった情報は電力会社のデータベースシステム、家電の情報 (制御 API や保証期間など) についてはメーカーのデータベースシステムに格納されているという想定になる。このように uID アーキテクチャ 2.0 においては、各データに対して責任をもつ機関がそれぞれにおいて管理を行い、それを融合する仕組みを提供することで全体としての情報フレームワークを実現する。具体的には、uID アーキテクチャ 2.0 はアプリケーションと個別のデータベースとの間の中間に位置し、アプリケーションシステムからの全データベースシステムを統合するシステムとして構成されている。これにより、アプリケーションから見ると、あたかも現実世界の事物に関する巨大な仮想データベースが存在しているように見え、一つの問い合わせにより ucode に関する様々な情報を統合的に引き出すことが可能になる。

ucode 解決の動作概要は図 3.3 の通りであるが、その流れを以下に示す。

1. まずはじめに、アプリケーション (利用者) は、ucode 解決のためのプロトコルである「ucode 解決プロトコル 2.0」(ucodeRP 2.0) に基づく問い合わせを uID センタの解決サービスに送付する。
2. これを受けた uID センタの解決サービスは、受信したクエリをバックエンドデータベースへのクエリの形に分解する。たとえば元のクエリが、ある家電製品に対する小売情報とメーカーが管理する製品情報に基づく場合、それを小売情報管理を行う組織のデータベースとメーカーの製品情報管理データベースへのクエリに分解する。
3. 分解したクエリ (サブクエリ) を、各個別データベースに送付する。
4. uID センタの解決サービスは、得られたサブクエリの結果を統合し、必要な演

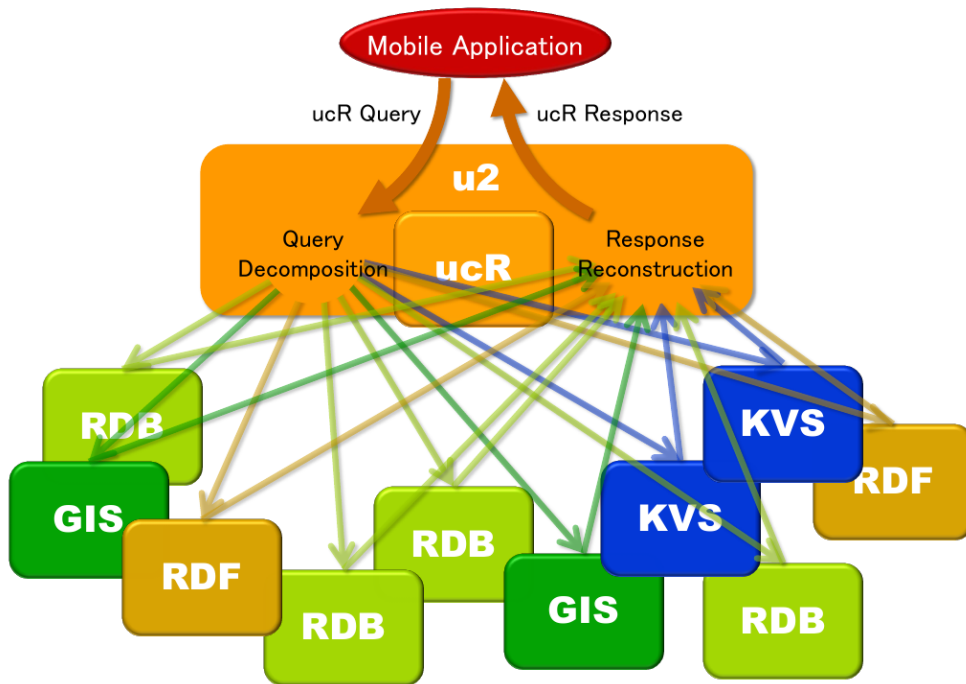


図 3.2: uID アーキテクチャ 2.0

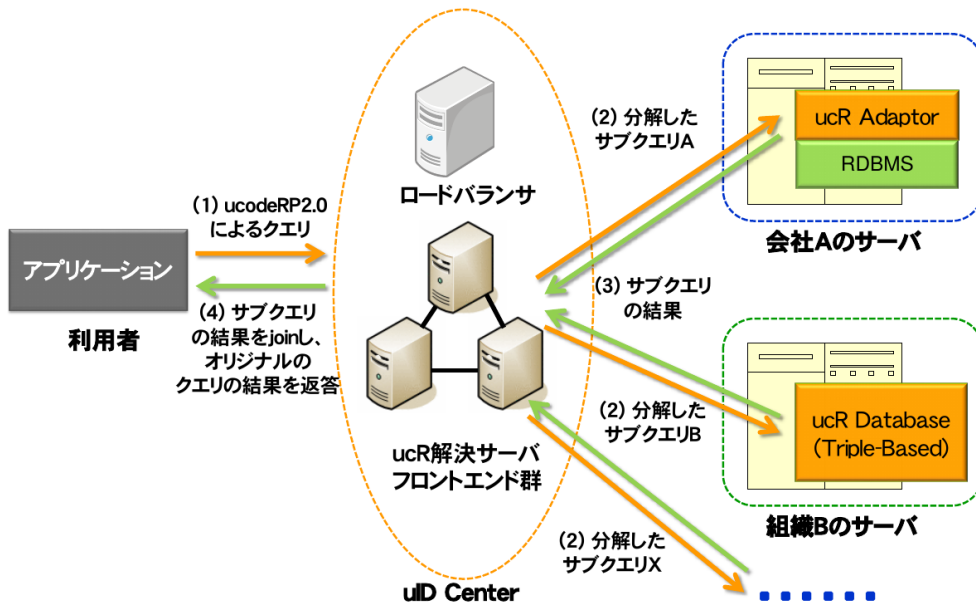


図 3.3: uID アーキテクチャ 2.0 における ucode 解決の流れ

算を行った上でもとのクエリに対するクエリ結果を生成する

5. 最終的に、得られたクエリ結果をアプリケーション (利用者) に返す。

3.2.3 実装

ここまでで示した uID アーキテクチャ 2.0 の概要に基づき、実際に ucode 解決サービスを実装した。そのうち、特に重要なコンポーネントは以下の 2 つである。

- ucode 解決フロントエンドサーバ
- リレーショナルデータベース向け ucR アダプタ

前者の ucode 解決フロントエンドサーバは、図 3.3 の点線内に示される ucode 解決のフロントエンドとなるサービス、すなわちクエリの受付・クエリの分割・クエリ結果の統合および返却処理を行う、uID アーキテクチャ 2.0 の根幹となるサービスである。後者は図 3.3 の会社 A のサーバにおいて「ucR Adaptor」と示されている部分であり、既存のリレーショナルデータベースを ucode 解決プロトコル 2.0 の仕様に適合させるためのラップモジュールである。いずれについても実装を終え、T-Engine フォーラムのワーキンググループにおいて限定公開の上で運用を行っている。

3.3 機器向けのソフトウェアフレームワーク

3.3.1 CoAP (constrained application protocol)

前節で示した uID アーキテクチャ 2.0 は、ユビキタスネットワークアーキテクチャ全体における情報処理を中心としたフレームワークであり、ユビキタスコンピューティングにおける多様かつ膨大な情報を扱い、状況認識とそれに基づく判断や処理の決定を容易にする仕組みである。本節で紹介する機器向けのソフトウェアフレームワークは、ユビキタスネットワークアーキテクチャの終端の機器において、以下の 2 つの機能を実現するためのものである。

- 状況判断のために必要な一次的な情報 (センサ値等) を発信するための機能
- ネットワーク経由の要求に基づき機器の制御を行う機能

これらを実現するために大前提となる機能として、組込み機器におけるネットワーク通信を行う機能の実現が不可欠である。ユビキタスネットワークアーキテクチャにおいては、組込み機器の規模に応じた複数のネットワーク通信機能の実装を提供しているが、その中で特に重要となるのが小規模機器向けの制御のためのネットワークプロトコルスタックの実装である。本研究ではこれらの小規模機器向けのプロトコルとして、CoAP (constrained application protocol) [47] を採用し、それに基づく制御フレームワークを構築した。CoAP は HTTP のような RESTful API [16] を省電力かつパケットロスのある大きな環境で行うために設計されたプロトコルである。CoAP は以下の特徴を持つ。

1. UDP に基づく通信プロトコルである
2. パケットサイズが小さい
3. URI に対するメソッド (GET, PUT, POST, DELETE) の実行が、その URI によって示されるリソースの操作に相当する

1. および 2. の特徴については、HTTP にはない CoAP ならではの特徴であり、フレーム長の限られた IEEE 802.15.4 などの通信において効率的に RESTful API を実現するために必須となる。3. は、いわゆる HTTP 等に代表される RESTful なプロトコルの特徴であるが、CoAP のエンドポイントは一般的には組込み機器であり、ウェブサービスではないことからその意味合いが若干異なってくる。通常の場合、HTTP におけるリソースはウェブ上のデータに相当することが一般的であるが、CoAP におけるリソースは機器内の制御項目に相当することとなる。たとえば、空調機に対する設定温度をリソースとみなすことで以下のような URI を想定できる。

```
coap://air-conditioner01.device.ubin.jp/temperature
```

この URI に対し、GET メソッドを発行すると現在の温度設定を取得することができ、PUT メソッドを発行することで温度設定を更新することができる (図 3.4)。これを、3-way handshake 等による通信のオーバーヘッドを生じることなく、要求・応答をそれぞれ 1 パケットかつ 1 フレーム内に収めることができるのが CoAP の特徴である。

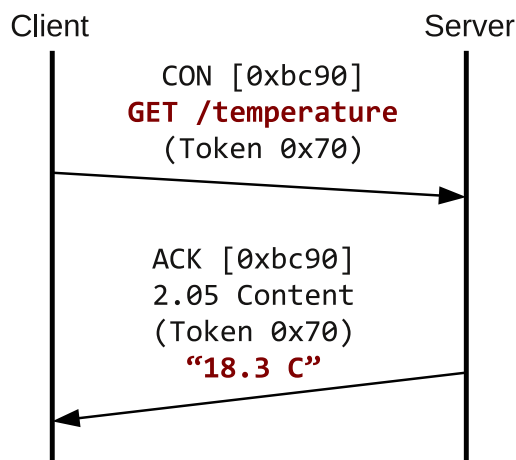


図 3.4: CoAP による温度設定の取得例

3.3.2 uID-CoAP アーキテクチャ

本研究では、前節で説明した CoAP と uID アーキテクチャ 2.0 を統合したアーキテクチャ「uID-CoAP アーキテクチャ」によって、機器制御向けのネットワークプラットフォームの実現を進めている。その全体像は図 3.5 に示すとおりとなっている。

uID-CoAP アーキテクチャにおいて重要な役割を果たすのが、6LoWPAN ボーダルータである。これは、CoAP と HTTP のプロトコルを相互変換するためのコンポーネントであり、両者によるネットワークをシームレスにつなぐことに役立つ。これにより、SOA の考えに基づき構成される uID アーキテクチャ 2.0 に対し、CoAP API を持つ組み込み機器ノードをシームレスに統合することができる。図 3.5 中の例では、スマートハウスを実現するウェブアプリケーションに携帯端末からアクセス (/house/1/info への GET メソッド) し、そこから Ajax (XmlHttpRequest) を通して HTTP により組み込み機器への制御要求 (/temperature への GET メソッド) が発行される過程を示している。

3.3.3 組み込み機器向け CoAP フレームワーク

組み込み機器を uID-CoAP アーキテクチャの構成要素として動作させるためには、組み込み機器に CoAP プロトコルに基づく RESTful API を実現しなければならない。これを容易にするため、組み込みリアルタイムオペレーティングシステムにおけるデ

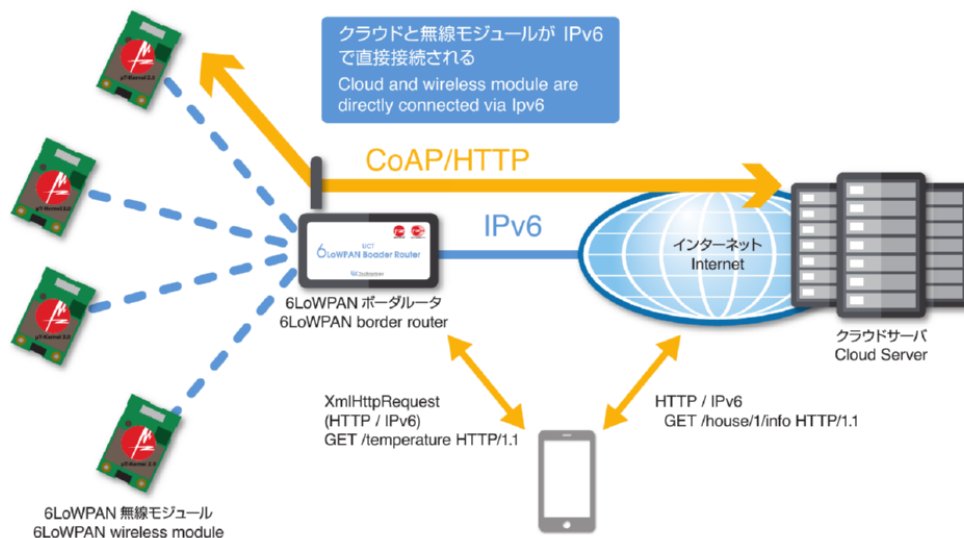


図 3.5: uID-CoAP ネットワークアーキテクチャ

ファクトスタンダードの一つである T-Kernel 上に、組み込み機器向けの CoAP フレームワークを実現した。

組み込み機器向け CoAP フレームワークのアーキテクチャは図 3.6 に示すとおりとなっている。図中について、青く色付けられた部分がフレームワークで提供されるソフトウェアコンポーネントであり、それ以外の白い部分についてのみアプリケーション開発者は実装を行う。

組み込み機器向け CoAP フレームワークにおいて、特筆すべき特徴として以下の 2 点が挙げられる。

- 既存のアプリケーションシステムと CoAP サービス定義が独立していること。これにより、既存のアプリケーションシステムを大幅に書き換えることなく、自然な形で CoAP による RESTful API を追加し、uID-CoAP アーキテクチャにおけるデバイスノードとしてユビキタスネットワークアーキテクチャに統合することが可能となる。
- CoAP のプロトコルの詳細はサービスフレームワーク内に隠蔽され、アプリケーション開発者は CoAP サービス定義において処理の内容にフォーカスした開発を行うことができる。

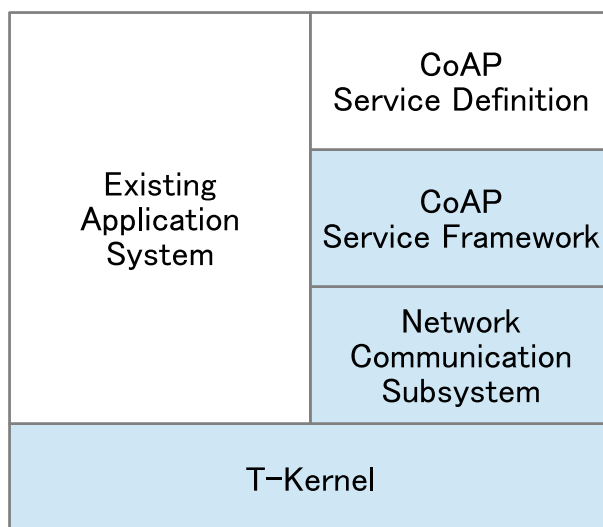


図 3.6: 組込み機器向け CoAP フレームワーク

図 3.7 が CoAP サービス定義としてサービスフレームワークに対して提供するデータ構造であり、このデバイスインタフェース定義構造体のインスタンスについて `slp_add_device()` API を呼び出すことでデバイスに対する CoAP API インタフェースが定義される仕組みとなっている。デバイスインタフェース定義構造体はデバイス識別子 (ucode) を含むいくつかの情報に加え、各メソッドのコールバック関数から成り立っている。ここで注目すべきは、各メソッドのコールバック関数においてプロトコルの詳細が隠蔽されており、全てデータバッファを通したやり取りとなっていることである。例えば、GET メソッドに対するコールバック関数 `get()` には、引数 `buf` で示される領域にメッセージボディの内容を書き込み、その長さを戻り値として返すような関数定義を与えればよい。要求の受け付けと応答の送信処理についてはフレームワーク内部で自動的に行われるため、アプリケーション開発者が記述する必要はない。具体的な GET メソッドの定義例を図 3.8 に示す。この例では、`"/temperature"` に対する GET メソッドのリクエストに対し、温度データを文字列データとして出力し返す実装となっているが、CoAP プロトコルの内部メカニズムが一切含まれていないことが確認できる。

この仕組みにより、既存の組込みシステムを容易にユビキタスネットワークアー

```

struct slpdev {
    /* Device identifier */
    ucode_t devid;

    /* Extended information */
    void* exinf;

    /* "GET" method implementation */
    INT (*get)( struct slpdev* self,
                const char* path,
                void* buf, INT max );

    /* "PUT" method implementation */
    ER (*put)( struct slpdev* self,
                const char* path,
                const void* buf, INT count );

    /* "POST" method implementation */
    INT (*post)( struct slpdev* self,
                 const char* path,
                 const void* sndbuf, INT sndcnt,
                 void* rcvbuf, INT rcvmax );

    ....
};

```

図 3.7: デバイスインタフェース定義のための構造体定義

```

/* "GET" method for specific air conditioner product */
INT aircon_get(struct slpdev* self, const char* path,
               void* buf, INT max)
{
    if (strcmp(path, "/temperature") == 0) {
        /* Read temperature from HW */
        UINT t = in_w(GPIO_TEMPATURE) * C1 + C2;

        /* Output reply data to buffer */
        return snprintf(buf, max, "%d.%d C\n", t/10, t%10);
    }
    else if (strcmp(path, "/off-timer") == 0) {
        /* Can add other paths like this */
        ....
    }
    ....
    else {
        /* Non-existent path specified */
        return E_NOEXS;
    }
}

```

図 3.8: GET メソッドの実装例

キテクチャに統合させることが可能となる。

3.4 eTNet アーキテクチャ

ユビキタスコンピューティングの実現にあたり、無視することのできない技術要素の一つにセキュリティが挙げられる。このうち、ユビキタスコンピューティングにおいて特に重要性が高まるのが、金銭や権利のネットワークを介した授受、個人や機器のアクセス権の制御を行うための枠組みである。ユビキタスコンピューティングにおいてはアクセスされる対象が単純な情報のみではなく、現実世界の機器の制御やそれらから得られる現実世界に関する情報を含んだものとなるため、これらに対するロールモデルも複雑なものとなる。たとえば、家電のネットワーク API を介した操作を考えた時に、誰がその家電を制御可能かは動的に変化しそれほど簡単に管理出来るものではないことが分かる。例えば、製造元（メーカー）から機器が出荷されたのち、ある利用者が機器を購入したあとはその利用者に家電の制御権が移ることになる。さらには、インターネット上にホスティングされたス

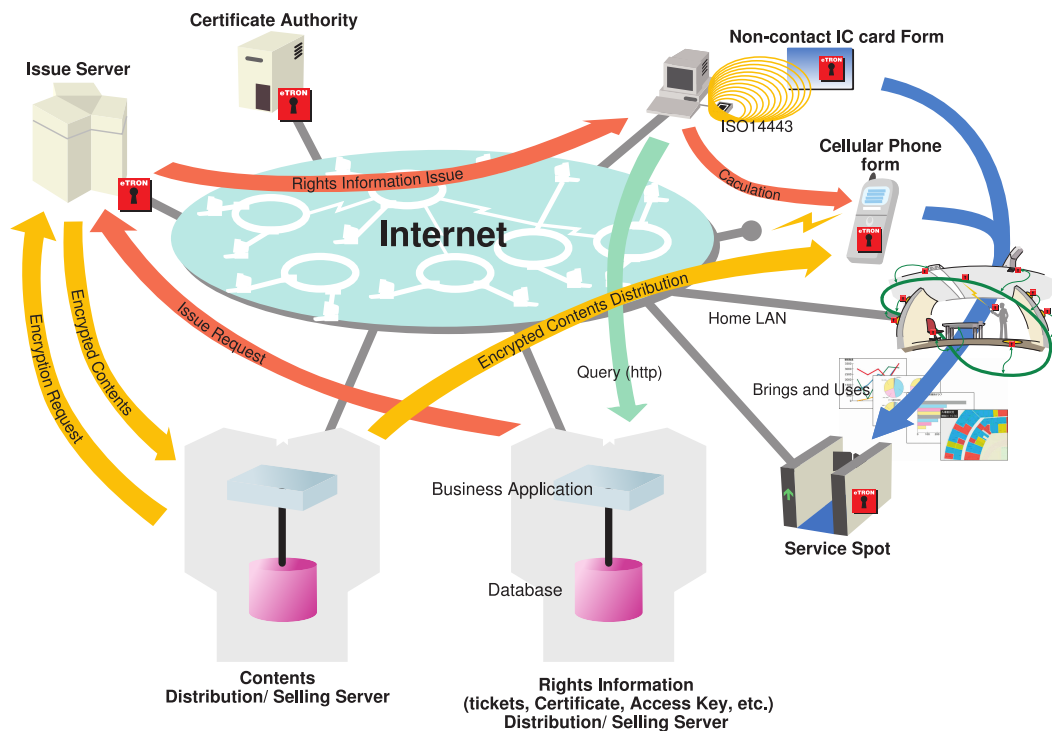


図 3.9: eTRON アーキテクチャ

マートハウスサービスからの制御を考えると、その実現のために権利の部分的な委譲処理も必要となる。

このように、ユビキタスコンピューティングにおいては、モノや情報に割り当てられた権利を移動・交換したり、それに基づくアクセス制御処理を行うための仕組みが必要となる。このような、ユビキタスコンピューティングにおけるセキュリティのためのアーキテクチャとして T-Engine フォーラムにより提案されているのが、「eTRON アーキテクチャ」[55, 45]である。eTRON アーキテクチャの全体像は図 3.9 に示す構成となっており、インターネットの上にも実現される価値や権利などの情報を交換・流通させるための基盤的アーキテクチャとなっている。eTRON アーキテクチャにおいては、eTRON チップと呼ばれる耐タンパ性の IC チップが価値情報のセキュアな格納および交換のために必要な機能を提供し、その機能を用いて価値や権利に関する情報をセキュアに流通させることができる。

この eTRON アーキテクチャの設計コンセプトに基づき、本研究ではその実装で

ある「eTNet アーキテクチャ」を設計し、実際に実装を行った。eTNet アーキテクチャは、インターネット上のオーバーレイネットワークとして実現され、そのネットワーク識別子として ucode を用いる。図 3.10 は、eTNet におけるセッション通信のフローを示したものであり、リレーサーバを介した最も単純なオーバーレイネットワークの構築はこの手順により実行される。この仕組みにより、実ネットワークのネットワークトポロジやそれに基づき割り当てられるネットワークアドレスによらない、ucode で一意に識別されるエンティティ間での価値および権利の交換を自然に実現できた。

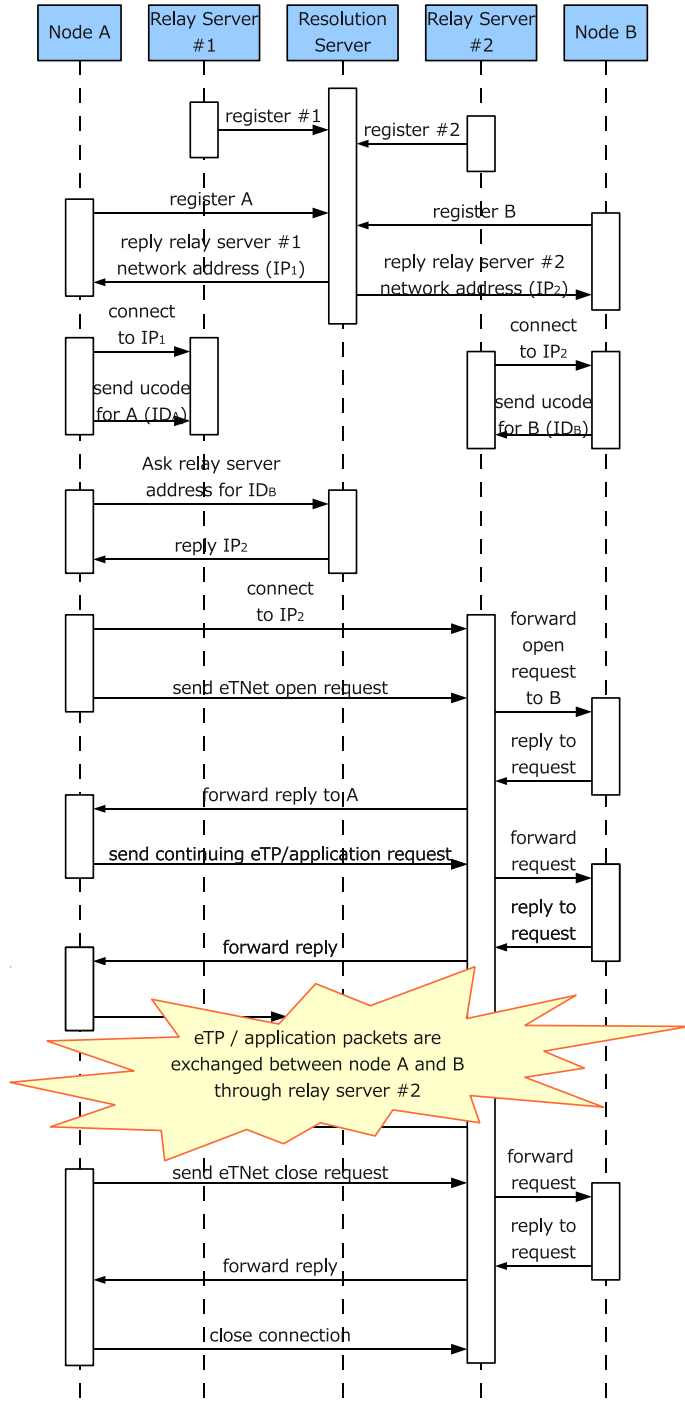


図 3.10: eTNet におけるセッション通信のフロー

第 4 章

UC フレームワーク

本章では、本研究の中核である UC (ubiquitous communicator) フレームワークの論理的な構成について説明する。

4.1 解決すべき課題

本研究の最大の目的は、ユビキタスコンピューティングにおけるユーザインタフェース端末上でのサービスの実現を容易にする仕組みの提供である。そのためには、以下の要件が同時に満たす仕組みを実現しなければならない。

- 自動的な動作
- 高い開発効率
- システム全体としての最適動作

以下、具体的にこれらについて説明する。

4.1.1 自動的な動作

自動的な動作の実現はユビキタスコンピューティングの本質であることから、避けて通ることのできない重要な課題である。ユビキタスコンピューティングの基本的な考え方として、コンピュータが現実世界の情報を認識し、それに基づく最適動作を実現するというものがある。たとえば空調を例に上げると、従来のコンピュータシステムにおいてはユーザの操作によってのみ設定が行われていたところを、環

境中に分布するセンサの値・人の位置・窓の開閉状態・電熱機器の状態等に応じて最適な制御を自動で行う、というのがユビキタスコンピューティングの方向性である。もちろん近年の高機能化した家電においては、機器内での最適制御はある程度実現しているものもあるが、ユビキタスコンピューティングでは単一機器を超えた最適制御を行うという点に違いがある。

これを実現するにあたって課題となるのが、どのように状況判断を行うか、その上でどのように最適制御を行うかである。前述の空調の例にも示されるとおり、どのような制御を行うかは現実世界の多数の状況に依存するため単純とはいえない。また、不適切な動作はユーザにとっての使用感を損ねることとなるため、ユーザの考えや意図をくんだ動作を実現することが望ましい。

特に、ユーザインタフェース端末においては自動的に情報を提供することは「プッシュ型」の動作と言われ、ユーザ操作による「プル型」の操作と区別される。プッシュ型の動作はユビキタスコンピューティングにおけるユーザインタフェース端末には本質的な機能の一つであるが、同様に適切な状況判断に基づく最適な動作が求められる。

4.1.2 高い開発効率

第1章で示したように、ユビキタスコンピューティングシステムは一般的な従来型のソフトウェアよりも複雑度が高いことから、開発者の負担を減らし開発効率を向上させる仕組みが必要である。

ここで重要となるのが、アプリケーションソフトウェアのモジュール化である。一般に、大規模なシステムの開発を行う上ではモジュール化の考えが有効であることが知られており、様々なアプローチによりモジュール化が進められてきた。よく知られた例としては、オブジェクト指向プログラミングやソフトウェアのコンポーネント化技術 (CORBA [51] など) が挙げられるだろう。モジュール化は上記に示す開発効率の観点はもちろん、複数の組織にまたがったトータルシステムの実現を行うためには必須となる事項である。

ここで例として、ユビキタスコンピューティングのサービスインタフェースを提供する端末において家庭内の家電の制御機能を提供することを考える。家庭内には、冷蔵庫・電子レンジ・炊飯器・テレビ、といった多数の機器が存在しているが、それらの制御を単一のアプリケーションで実現することは現実的でない。仮に、冷蔵庫

のメーカーが A 社であり、電子レンジのメーカーが B 社であるとする、その制御が単一のアプリケーションで実現されていた場合に誰がソフトウェアを開発するかという問題が発生する。したがって、アプリケーションソフトウェアのモジュール化による、機能単位での独立性の実現は必須である。

4.1.3 システム全体としての最適動作

前節で挙げたモジュール化の重要性については、一般のソフトウェア開発においても共通した話であるが、その障害となるのがここで説明するシステム全体としての最適動作の実現である。

一般のソフトウェアシステムにおいては、アプリケーションの動作はそれぞれ独立していることがほとんどである。例えば、デスクトップシステムにおいて文章作成ソフトウェアと音楽プレイヤーを同時に動作させた場合、それらは深く関連しあうことなく基本的には独立に動作している。一方、ユビキタス情報端末においては自動動作の実現の観点からそこまで話は単純ではない。例として、ココシル銀座 [50] にみられるようなプッシュ型のユビキタス場所情報システムにおいて、地図アプリケーション・地物情報表示アプリケーション (ブラウザ) を分割することを考える。このようなプッシュ型のシステムでは、地図アプリケーションが現在位置を表示している最中にも、ユーザの興味のある地物に対する情報が後者のアプリケーションによってポップアップ通知される。しかしながら、地図アプリケーションで道案内を行っている最中にユーザの興味の無い情報を通知してしまうと、使用感を損ねる結果となってしまう可能性がある。このようなケースにおいてもシステム全体としての最適動作を実現するためには、地図アプリケーションの状態に応じて地物情報表示アプリケーションが挙動を変える必要がある。

4.2 設計方針

前節では UC フレームワークが解決すべき課題を 3 つあげたが、これらは独立した話ではない点に注意が必要である。たとえば、システム全体の最適動作を実現することはソフトウェア間の依存性を高めるということにつながるため、アプリケーションシステムのモジュール性を損なうことにつながり、結果的に開発効率を低下させてしまう。従って、高い開発効率を実現しながらも、自動的な状況判断とそれに基づく制御を全体としての最適な形で実現するための仕組みが求めら

れる。

本研究では、これを実現するため、以下の3点を基本とするシステム設計を行った。

- メカニズムとポリシーの分離
- アプリケーション間調停機構の導入
- コンテキストの抽象化

以下、順に説明を行う。

4.2.1 メカニズムとポリシーの分離

ユビキタスコンピューティングにおける状況の判断や操作のためのロジックは複雑なものであり、アプリケーション開発者がこのすべてのメカニズムを実装することは困難である。したがって、本研究ではシステムの本質的な動作をポリシー記述として外部記述可能となるようにする。これにより、複雑なメカニズムをアプリケーション開発者から隠蔽し、簡潔なポリシー記述により状況依存の処理を実現する。

4.2.2 アプリケーション間調停機構の導入

ユーザの入力によらない自発的な最適動作はユビキタスコンピューティングの本質であるが、これを各アプリケーションがシステム全体とは無関係に勝手に行うという方式ではうまくいかない。たとえばプッシュ型のユビキタス場所情報システムにおいて、地図アプリケーション・地物情報表示アプリケーション(ブラウザ)が独立にポップアップを行う仕組みでは片方の通知がすぐに別のアプリケーションの通知で上書きされてしまうなどの問題が生じる。そこで、本研究では、各アプリケーションがシステムを通じた間接的な操作を通してシステム制御を行う方針とする。

前述の例においては、場所情報システムにおけるプッシュ通知において「ウィンドウをトップに移動する」というような直接的なAPIではなく、「このウィンドウはユーザの興味対象をプッシュ通知するものである」というウィンドウの表示判断に必要なメタ属性をウィンドウに付加し、そのような属性を元にシステムがどのウィンドウをトップ表示するかを決定する。これにより、アプリケーションはシステム全体としての挙動を気にすることなく、システム全体として最も適した操作を間接的に実現することが出来る。

4.2.3 コンテキストの抽象化

従来のプログラミングモデルでは、状況を取得するための API が明確に定められているのが一般的である。たとえば、Microsoft Windows 7 および 8 においてはセンサ値を利用する API として Sensor API [31] が規定されており、各センサの種別に応じた API が明確に規定されている。これらによってデバイスドライバごとの特殊性によらない統一したインタフェースを通してセンサ値等の情報が簡単に利用可能になる。

一方で、ユビキタスコンピューティングにおける制御の決定は、センサ等の値そのものよりは意味のある状況 (コンテキスト) に依存しているという点にある。例えば空調の制御の例であれば、部屋が暑い場合に冷房を ON にするという制御を考えた時、まず第一に「暑い」という判断をどのように行うかが問題となる。温度センサのみで判断する方法も考えられるが、湿度センサや風速等を用いてより人間の感覚に近い制御を行うことも考えられる。さらには、環境中にばらまかれた温度センサの分布を利用することも考えられる。それに加えて暑いか否かのみで空調制御を行うのでは明らかに不十分な場合がほとんどであり、実際のところはさらに多くのコンテキストに依存した処理が行われることとなる。たとえば、人がそこにいるか、空いている窓があるか、といったコンテキストに基づく判断が考えられる。従来のシステムにおいては、アプリケーションごとに環境変数の値やセンサ等の値をシステムから取得し、それらをアプリ内で計算して判断に用いるコンテキストを計算し、それに応じた動作を行うような作りがなされている。しかしながら、これはアプリケーション開発者の負担となるほか、このような判断ロジックを少し修正するたびにプログラムを書き換えなければならないこととなる。

これを解決するため、本研究ではセンサ値のような低次のコンテキストから、「暑い」といった低次のコンテキストから推論される高次のコンテキストに変換するメカニズムを提供し、アプリケーションに低次のコンテキスト・高次のコンテキストの両方を共通のインタフェースで提供するための仕組みを提供する。これにより、アプリケーション開発者は高次のコンテキストを用いて、より直感的かつ簡潔な状況判断とそれに基づく制御を実現できる。

4.3 全体構成

ここでは、本研究で提案する UC フレームワークのシステム構成について説明する。UC フレームワークはコンテキスト推論フレームワーク、アプリケーション間調停フレームワーク、省電力制御フレームワークによって構成されるフレームワークであり、そのメカニズムの概要は図 4.1 に示す通りである。

各フレームワークにはそれぞれの役割に応じたポリシー記述言語が割り当てられている。コンテキスト推論フレームワークに対しては、コンテキストポリシー記述と呼ばれるルール記述により、システム設定値やセンサ値等に基づくより高次元のコンテキストの定義が行えるようになっている。アプリケーション間調停フレームワークに与えられるアプリケーション間調停ポリシー記述、省電力制御フレームワークに与えられる電源管理ポリシー記述では、コンテキスト推論フレームワークから得られるコンテキスト群が示す状況に基づき、それぞれアプリケーション間の調停、省電力の制御処理を決めるためのルールが示される。

UC フレームワークの処理フローは、センサ値の変化などを含むコンテキストの変化を起点とするものと、アプリケーションからの要求に基づくものがある。前者については以下のような流れで処理が行われる。

1. コンテキスト推論フレームワークは、システム設定値やデータベース内のエントリの变化、あるいはセンサ API を通して得られる物理量の変化の通知を受け、それを元に高次コンテキスト群をコンテキストポリシー記述に従って再計算する。
2. 再計算の結果、変化のみられた高次コンテキストと通知のあった低次コンテキストすべてに対し、その変化を他のフレームワークや各アプリケーションシステムに通知する。
3. 通知されたコンテキストの変化に対し、各アプリケーションはそれに応じた処理を行うとともに、UC フレームワーク内では各フレームワークが以下の処理を実施する。

アプリケーション間調停フレームワーク

コンテキストの変化はアプリケーション間の調停を行う条件の変化を意味するため、アプリケーションの調停条件を再計算し、結果に応じた調

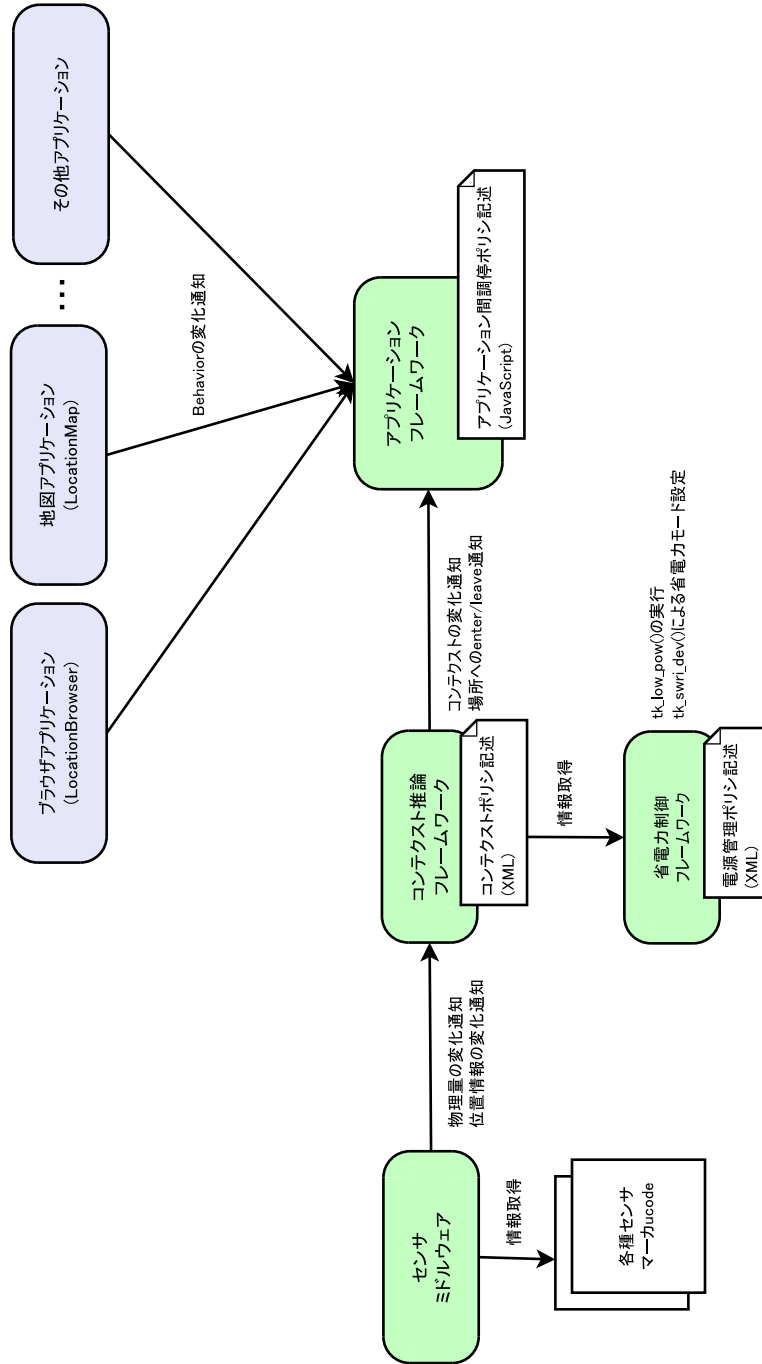


図 4.1: UC フレームワークの概要

停処理を行う。

省電力制御フレームワーク

コンテキストの変化に伴い、省電力モードを切り替える。ユーザインタフェース端末によって最も有用な例をあげると、利用者が画面を見ていない時にスクリーンのバックライトを切る、といった処理が考えられる。

後者のアプリケーションからの要求については、以下のような処理フローとなる。ここではあるアプリケーションが通知のための音声を出力したい場合の処理を想定する。

1. アプリケーションは、アプリケーションフレームワークに対し目的の制御に関する要求を送信する。このケースでは、音声を流したいという要求そのものに加え、流したい音声に関するメタ情報(重要度、内容、再生を割り込まれて構わないか、など)を付加したリクエストを送信する。
2. リクエストを受けたアプリケーションフレームワークは、制御対象に関する全アプリケーションおよびサービスからの要求をまとめ、それに基づきアプリケーションの調停条件を再計算し、その結果としてシステム全体としてのアクションを決定する。
3. 決定したアクションに基づき、必要な制御を行う。なおここで示している音声制御の例においても、必ずしも元々の要求がそのまま実行されるとは限らない点に注意が必要である。場合によってはより優先度の高い音声優先されるかもしれないし、リスニング試験中といったコンテキストを認識して全く音声を再生しないかキューイングする、といった処理を行うことも考えられる。

以降の各節では、各フレームワークの詳細について議論する。

4.4 コンテキスト推論フレームワーク

4.4.1 概要

コンテキスト推論フレームワークは、UCフレームワーク全体における様々なシステムの設定値や状態、センサデータなどの外部環境情報を統合し、アプリケーションにとって意義のある高次コンテキストに変換して、アプリケーションや各システムサービスに配信するための基盤システムである。

コンテキスト推論フレームワークが扱うコンテキストは、大まかには以下の2種類に分けることができる。

プライマリコンテキスト

コンテキスト推論フレームワークに入力される入力そのものであり、加工されていない生のデータに相当する。具体的には、デバイスドライバやセンサ API を経由して得られるセンサ値、環境変数の値、システム状態等がプライマリコンテキストにあたる。

合成コンテキスト

プライマリコンテキストや他の合成コンテキストの演算によって定義される、合成されたコンテキスト。具体的な例としては、ユーザが暑いと感じているか否か、という情報は合成コンテキストにあたり、温度センサや湿度センサ、ユーザの属性情報や行動などの情報によって推論されるコンテキストである。

コンテキスト推論フレームワークではプライマリコンテキストに対する監視を行い、それらに変化があったときに合成コンテキストを再計算する。すべての再計算が完了した時、コンテキスト推論フレームワークはすべてのアプリケーションやシステムコンポーネントに対して、変化したコンテキストの情報を通知する。この通知に基づき、アプリケーションやシステムコンポーネントは通知をイベントとして受信し、必要な処理を行うことでユビキタスコンピューティングシステムにおける自動制御を実現する。スマートハウスにおける空調制御に例を挙げると、受信したコンテキスト変化通知に基づき例えば以下のような処理を行うことができる。

ユーザが暑いと感じている場合

空調の温度設定を下げる。

ユーザが寒いと感じている場合

空調の温度設定を上げる。

外気温と室内気温差が小さい場合

空調機の電源を切り、かわりに窓を開けることで換気を行う。

このように、アプリケーションからはプライマリコンテキストのみでなく合成コンテキストによる高次のコンテキストが得られることで、より直接的に制御のためのロジックにフォーカスした開発を行うことが可能となる。さらに、コンテキストレ

ベルでの処理記述になっていることから、より複雑な状況判断にシステムを対応させることも容易に可能となる。たとえば前述の例においても、外部の空気が光化学スモッグ等により汚染されている場合には窓をあけないなど、状況判断を直感的な記述により詳細化することが可能である。これが、デバイスドライバインタフェースやセンサ API 等による低次コンテキストのアクセス手段のみでは、このような処理に対応することでシステムの複雑度が増してしまうと想定される。

4.4.2 コンテキストポリシ記述

コンテキスト推論フレームワークでは、フレームワークの内部に具体的なコンテキストの定義を含める代わりに、コンテキスト推論のためのルールを外部化する方針をとった。その理由は以下に示すとおりである。

アプリケーションシステムにより用いるコンテキストが異なること

アプリケーションが必要とするコンテキストは多種多様なものが考えられ、あらゆる場面において必要十分なコンテキスト群をフレームワークから提供することは難しい。たとえば、「お腹が減った」というコンテキストはあくまで食事に関するサービスなどの一部のサービスのみに関連した強い一般性を持たないコンテキストだが、このような特殊なコンテキストを含めた処理を実現するために、ルールを外部化し、必要なコンテキストをアプリケーション開発者が自由に定義できる方針とした。

コンテキストの定義はシステムによって一定でないこと

たとえば、携帯端末上においてユーザが暑いと感じているかどうかについての推論を行う場合、温度センサを搭載している端末ではそれを元にした推論が、温度センサ・湿度センサの両方を持っている端末ではそれらから導き出される不快指数に基づいた推論とすることができる。さらには、環境中のセンサを用いてネットワーク越しにより正確な室内の温度分布情報を得ることもできる。このように、コンテキストをどのように推論するかについての具体的な方針は、利用可能なセンサや環境中のリソースによって最適なものが異なっており、絶対的な唯一解が存在するわけではない。このことから、コンテキストの推論規則に関する定義はコンテキスト推論フレームワーク本体では実現せず、同フレームワークに外部ルールを与えることでそれに基づく推論が行われるメカニズムとした。

コンテキスト定義は、他のコンテキスト群によって構成される式により表現される。例えば、携帯端末が首からぶら下がっている、というコンテキストは携帯端末の計上やストラップの接続位置、さらには搭載しているセンサに依存するが、以下のように定義することができる。

$$\begin{aligned} \text{首からぶら下がっている} ::= & (\text{携帯端末のピッチ角} < 45 \\ & \wedge \text{ボタンが押されていない}) \end{aligned} \quad (4.1)$$

画面を注視している、というコンテキストについても、例えば以下のような定義を行うことができる。

$$\begin{aligned} \text{画面を注視している} ::= & \text{ボタンが押されている} \\ & \vee (\text{携帯端末のピッチ角} < 45 \\ & \wedge \text{端末の移動速度} < 20\text{cm/s}) \end{aligned} \quad (4.2)$$

なお、上記で定義したコンテキスト例はいずれも真偽値によるコンテキストであるが、これ以外にも値を持った引数を許容する。例えば現在の気温と湿度によって導かれる不快指数はその定義に従って以下のように定義すれば良い。

$$\begin{aligned} \text{不快指数} ::= & 0.81 \times \text{現在の気温} \\ & + \text{現在の湿度} \times (0.99 \times \text{現在の気温} - 14.3) \\ & + 46.3 \end{aligned} \quad (4.3)$$

4.4.3 コンテキストの再計算

このようにして定義された合成コンテキストは、プライマリコンテキストの値に応じて変化するため再計算が必要となる。コンテキスト管理マネージャはコンテキストの再計算を以下のアルゴリズムにより行う。

1. まずはじめに、全コンテキスト (C) を以下の3種類の集合に分割する

- 再計算済であり、値が更新されたコンテキスト (C_{upd})
- 再計算済であり、値の更新が行われていないコンテキスト (C_{ident})
- 再計算が済んでいないコンテキスト ($C_{unknown}$)

初期状態として、 C_{upd} には変化のあったプライマリコンテキスト集合を、 C_{ident} には変化のなかったプライマリコンテキスト集合を、 $C_{unknown}$ にはすべての合成コンテキストを代入する。

2. 次に、 $C_{unknown}$ の集合の要素のうち、以下の条件を満たすものを一つ取り出す。(取り出した要素を c とする)

- c の定義式の右辺に含まれるコンテキストがすべて $C_{upd} \cup C_{ident}$ に属していること

この条件を満たす要素がない場合はアルゴリズムは終了となり、この時点ですべてのコンテキストの再計算が完了している。

3. c の定義式の右辺に含まれる要素がすべて C_{ident} の要素であるか否かにしたがって、 $C_{upd} \cdot C_{ident} \cdot C_{ident}$ をそれぞれ以下のように更新する。

- すべて C_{ident} の要素: コンテキスト c の値の再計算は不要であり、 $C_{upd} := C_{upd} \cup \{c\}$, $C_{unknown} := C_{unknown} - \{c\}$ とする。
- それ以外の場合: コンテキスト c の値を再計算し、 $C_{ident} := C_{ident} \cup \{c\}$, $C_{unknown} := C_{unknown} - \{c\}$ とする。
- 2. に戻り $C_{unknown}$ が空となるまで処理を繰り返す。

このアルゴリズムによって得られたコンテキストの集合 C_{upd} が更新のあったコンテキスト全体であり、コンテキスト推論フレームワークはこの情報をコンテキスト変化通知として、各アプリケーションやシステムサービスに通知する。

4.5 アプリケーション間調停フレームワーク

4.5.1 概要

アプリケーション間調停フレームワークは、複数のアプリケーションの挙動を調停することで、ユビキタスコンピューティングシステムにおけるシステム全体の最適動作を実現するための仕組みである。例えば、ユビキタス情報サービス端末において音声での情報のプッシュ通知を行うときに、各アプリケーションシステムが勝手に音声再生を行ってしまうと多重音声状態となってしまう一般には聞き取りが困難になってしまう。これを避けるため、たとえばこの中から最も優先して再生すべき音声の一つを選んで再生し、他は再生するか再生を遅延させるかを判断するのが、アプリケーション間調停フレームワークの目的である。

ここで、複数のアプリケーションを全体最適となるように制御を行うためには、その制御対象に対する情報が必要となる。前述の例では、再生対象の音声は複数存在するとき、通知対象のコンテンツの重要度や緊急性、さらには音声の長さといったファクターがなくては最適な再生制御を行うことができない。これは音声再生の調停に限らず、画面上のコンテンツプッシュにおける制御においても全く同様である。

このことから、アプリケーション間調停フレームワークでは「ビヘイビア」(*behavior*)という概念を導入し、それをもととしたアプリケーションの制御を行う。ビヘイビア (*behavior*) とは、制御対象に対するアプリケーションの振る舞いに関するメタ情報をひとまとめにしたものとして定義する。たとえば、前述の音声通知の例においては通知対象のコンテンツの重要度や緊急性、さらには音声の長さといった情報をひとまとめにしたものがこれにあたる。UCフレームワークにおいては、アプリケーションは音声を直接再生する代わりに、再生したい音声データとそれに対するメタ情報(ビヘイビア)を組にしたものをアプリケーション間調停フレームワークに引き渡す。アプリケーション間調停フレームワークは、集められた音声データについて、それらのビヘイビアをもとにどの音声再生を優先すべきかを判断し、適切な音声を選択した上で再生制御を行う。

なお、ビヘイビアを用いたアプリケーション間の調停は、システムとしての全体動作を実現しながらもアプリケーション間の高い独立性を維持している点に特徴がある。前述の音声の例では、各アプリケーションは通知対象のコンテンツに対するメタデータのみを与えればよく、どのように調停を行うかはアプリケーション間調停フレームワークにより判断される。一般的な方式においては、アプリケーション間でのメッセージ通信を用いて各アプリケーション間でコンフリクトが生じないような動作を行う方式となっているが、このような方式ではアプリケーションの新規追加や新規変更に対する柔軟性が低いという問題がある。本研究で提案するビヘイビアに基づく調停はこの点で一般的な方式より優れていると考えられる。

なお、ここではビヘイビアによる制御例として音声を例に挙げたが、ビヘイビアによる調停は他の制御対象に対しても有効である。具体的には、以下のような制御対象をビヘイビアを用いることでシステム全体として最適な操作を実現できる。

表示のポップアップ

ユーザインタフェース端末上で複数のアプリケーションがプッシュ通知による情報のポップアップ表示を行う際に、複数のアプリケーションが勝手にポップ

アップ処理を行うと、あるアプリケーションが前面に表示されてからすぐに他のアプリケーションのポップアップが生じるなど、全体動作としては不適切なものとなる。ビヘイビアによる調停メカニズムを用いることで、全体として最適な表示のポップアップを実現できる。

バイブレーションの制御

携帯端末におけるバイブレーションによる通知の際に、似たタイミングで発生したバイブレーション通知を統合してまとめるといった処理に、システムレベルでの調停メカニズムが有用である。

4.5.2 アプリケーション間調停ポリシー記述

前節で示したように、アプリケーション間調停フレームワークでは、各アプリケーションから提供されるビヘイビアをもとに制御内容を決定する必要がある。しかしながら、この制御内容の決定は常に一定であるとはいえず、アプリケーションの構成やビヘイビアに含める情報の内容に応じて変わってくるのが想定される。たとえば前述の音声再生の例において、災害緊急速報の通知をシステムの機能として新たに追加したと仮定すると、ビヘイビアには新たにコンテンツの属性として災害の緊急通知情報であるという属性を加える必要が生じる。それによって、ビヘイビアに基づくアプリケーション間調停の処理自体の内容も変わってくる。

このように、ビヘイビアの内容もそれに基づく調停処理も一定ではないことから、アプリケーション間調停フレームワークではこれらの多様性や変化に柔軟に対応できる仕組みが求められる。このことから、アプリケーション間調停フレームワークでは、アプリケーション間の調停に関する処理を外部記述として分離することとした。この外部記述は、アプリケーション間調停ポリシー記述と呼ばれる。

各制御対象に対するアプリケーション間調停ポリシーは、ビヘイビアの集合からアクションへの関数として定義される。そのため、アプリケーション間調停ポリシー記述はプログラミング言語による関数定義を制御対象ごとに列挙したものとなる。関数定義中にはコンテキスト値を参照する機能が提供されており、アプリケーション間調停においてコンテキスト情報を活かすことも可能となっている。

4.5.3 アプリケーション間調停の実行

前節で示したポリシー記述に基づくアプリケーション間調停が実行されるタイミングは、調停結果の再計算が必要になった時である。これはすなわち、以下のいずれかの条件に相当する。

- アプリケーションからの本フレームワークへのビヘイビアの更新通知が届いた
- 調停ポリシー記述に含まれるコンテキストが変化した

上記のいずれかの条件が成立した時に、アプリケーション間調停フレームワークはアプリケーション間調停ポリシー記述で定義された関数を呼び出し、システム全体としてとるべきアクションを決定する。この調停結果に基づき、アプリケーション間調停フレームワークは制御対象に対する制御処理を行う。

4.6 省電力制御フレームワーク

4.6.1 概要

UCフレームワークにおいては、デバイスの省電力制御を行うための仕組みを省電力フレームワークにより実現する。

省電力制御フレームワークでは、ユビキタスコンピューティングにおけるユーザインタフェース端末上での省電力制御を、現在のコンテキストに基づき自動的に行うためのメカニズムである。ユーザインタフェースに搭載される入出力デバイスは必ずしも一定ではなく、コンテキストポリシー記述によって定義されるコンテキストについても常に一定の内容とは限らない。このことから、省電力制御フレームワークにおいても外部から与える電源管理ポリシー記述により、電力管理に関する実際の挙動を開発者が自由に決められるようにした。

4.6.2 電源管理ポリシー記述

省電力制御フレームワークにおける電源管理ポリシー記述は、コンテキストにより記述される条件式と、それに対する省電力制御の組み合わせのリストによって指定される。LCD画面の輝度の制御を例に挙げると、以下のようなルール群によりコンテキストに応じた電源管理を定義することができる。

$$\begin{aligned} & \neg(\text{画面を注視している}) \Rightarrow \text{輝度 } 0\% \\ & (\text{屋内環境である}) \wedge (\text{照明がある}) \Rightarrow \text{輝度 } 60\% \\ & (\text{それ以外の環境}) \Rightarrow \text{輝度 } 100\% \end{aligned} \tag{4.4}$$

このようなルールは各省電力制御対象について独立して定義される。たとえば温度センサのサンプリング頻度、WiFiのスキャン周期など別の省電力項目については別のリストによってルール化される。これにより、電力制御を各機能ごとに独立した小さな単位で簡潔に行うことができる。

4.6.3 省電力制御禁止機能

省電力制御フレームワークでは基本的には前節で示したとおり、コンテキストの変化に応じて電力制御処理が行われるが、場合によってはこれによりアプリケーションでの処理に支障をきたす可能性がある。コンテキスト推論フレームワークでは、システム全体の状態や実世界における環境情報等のコンテキストなど、一般にはアプリケーションに依存しないグローバルな意味を持ったコンテキストを管理するため、コンテキストにのみ基づく電力制御ではアプリケーションの状態を考慮することができず、アプリケーション側の要求を満たせなくなるケースが生じる可能性が考えられる。例えば、コンテキストにあわせてWiFiモジュールを省電力モードに移行させた結果、アプリケーションが通信を行う際に長時間待たされる結果となりうる。

したがって本フレームワークでは、アプリケーションが電源管理マネージャに対し省電力化制御を禁止する機能を提供するものとする。具体的には、アプリケーションがその時点で最低必要なモジュールの機能レベル、すなわちサービスのクオリティ(QoS)を伝達し、それに基づき省電力制御フレームワークがその機能以上を実現するために必要な水準を下回った省電力モードへの遷移を禁止する方針とした。例えば、2つのアプリケーションがそれぞれ「ネットワーク通信機能を使用する」、「バックライトの明るさが70%以上」という要求を本フレームワークに送信した場合、電力管理ポリシー記述により決定される省電力アクションはそれを満たす範囲で緩和され、ネットワーク通信機能が利用不能となったり、バックライトの明るさが70%未満となることが内容に制御が行われる。

4.6.4 省電力制御処理の流れ

省電力制御フレームワークにおいては、コンテキストの変化またはアプリケーションからの省電力制御禁止要求があった時に、電源管理ポリシー記述の再評価を行い、それに応じた省電力制御処理を行う。具体的な流れは以下のとおりである。

1. 新たなコンテキスト値に基づき、電源管理ポリシー記述におけるルールの条件式部分を上から順に評価し、その評価結果が真となるものを見つける。
2. 見つかったルールに定義されたアクションを実行したときに、省電力制御禁止による指定が満たされないとき、アクションの省電力モードを同指定が満たされるように緩和する。たとえば、省電力制御禁止機能によってバックライトの輝度が70%を下回らないように設定されていたとすると、バックライトの輝度を50%に落とすというアクションが得られたときにもアクションはバックライトの輝度は50%までは下げられず、代わりに70%に設定される。
3. 得られたアクションに基づき、ターゲットデバイスに対する省電力制御処理を実行する。

第 5 章

システム実装

UC フレームワークはアーキテクチャに関する提案であり、様々な実装形態が想定できる。本研究においては、UC フレームワークを組み込みリアルタイム OS 上に構築し、「UC 場所情報システム」と呼ばれる観光ガイドアプリケーションまでを含めたトータルなユビキタス情報システムを実現した。

本章では、UC フレームワークに基づくこのシステム全体の実装について説明する。

5.1 システムアーキテクチャ

本論の提案する UC 場所情報システムのアーキテクチャ全体を図 5.1 に示す。図に示される通り、本研究では組み込み向けリアルタイムオペレーティングシステムのデファクトスタンダードの一つである T-Kernel [48] を採用し、その上にシステムを構築している。赤い枠で囲まれた部分が UC フレームワークであり、本研究における提案の中核となる部分である。

以降の各節では、各コンポーネントの詳細を示す。

5.2 ハードウェア

UC 場所情報システムでは、ハードウェアとして「UC110」(図 5.2) を利用した。UC110 は YRP ユビキタスネットワークワーキング研究所によって開発された機器であり、多数のセンサデバイスを搭載している点に特徴がある。表 5.1 に主な仕様を示す。

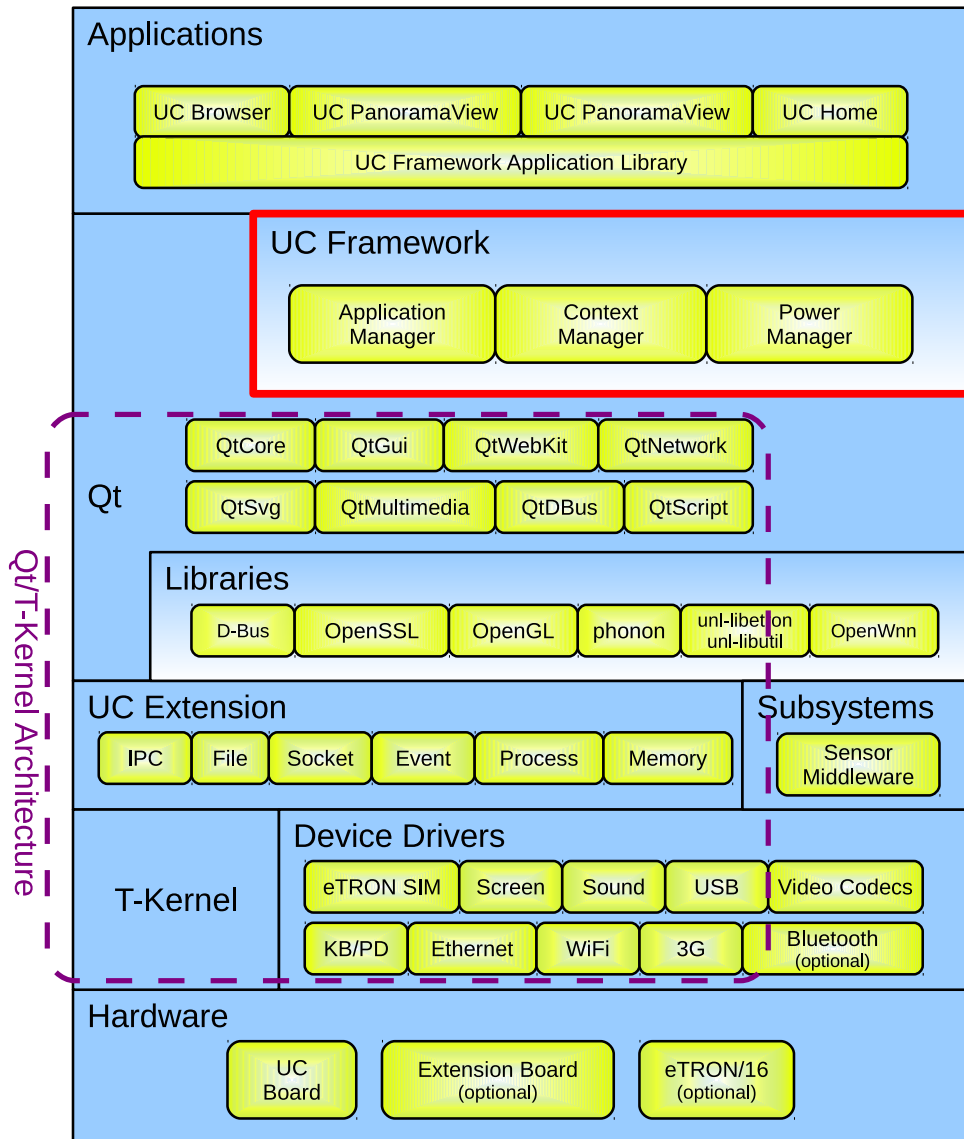


図 5.1: ユビキタス情報システムのアーキテクチャ

表 5.1: UC110 のハードウェア仕様 (概要)

項目	各項目の仕様
CPU	S5PC110 (ARM Cortex-A8) 1 GHz
ディスプレイ	4 inch TFT-LCD WVGA (800×480)
タッチパネル	Capacitive Multi-Touch Screen
筐体寸法	155 × 78 × 15.7 [mm]
ボード寸法	151 × 67 [mm]
RAM	512MB (mDDR)
内部ストレージ	OneNAND 512MB + eMMC 4GB
外部ストレージ	micro SD Slot(SDHC), mini USB
無線通信	WiFi IEEE 802.11b/g/n Bluetooth 2.1 + EDR, SPP, A2DP W-CDMA Module
カメラ	背面カメラ (5M pixel) 前面カメラ (0.3M pixel)
IrDA	2 個
GPS	あり (A-GPS をサポート)
RFID	13.56MHz, 400MHz, 900MHz, UWB, etc
センサ	3 軸ジャイロセンサ 3 軸加速度センサ + 3 軸地磁気センサ 気圧センサ 照度センサ 近接センサ
バイブレータ	on-board Vibrator
スイッチ	Power SW (slide type) Multi SW Hold SW Volume(+/-) SW Home SW Application SW Reset SW (背面)
音声出力	スピーカ (オンボード), 3.5 Φミニジャック (ステレオ)
クレードル接続	AV 出力 (HDMI+S/PDIF, 合成ステレオ), Ethernet
電源	リチウムポリマー充電電池 (1,550mAh), USB



図 5.2: UC110 の外観

5.3 Qt/T-Kernel アーキテクチャ

本節では、提案する UC 場所情報システムのうち、T-Kernel・UC Extension・Qt を中心として構成される部分アーキテクチャである「Qt/T-Kernel」アーキテクチャについて説明する。図 5.1 のうち、紫色の点線部分が本節で説明する Qt/T-Kernel アーキテクチャに相当している。

UC110 ハードウェア上に搭載するオペレーティングシステムとして、我々は組込みリアルタイムオペレーティングシステムにおけるデファクトスタンダードの一つである T-Kernel [48] を用いることとした。T-Kernel は、ハードリアルタイム性の必要とされる組込み機器のために設計・実装されたオペレーティングシステムであり、高い応答性と実時間処理の保証を特徴としている。我々の UC 場所情報システムにおいては多数のセンサからの入力に対するシステムの応答性能が重要となることから、T-Kernel を選択した。

一方、T-Kernel は基本的なリアルタイムオペレーティング機能を実現するコア機能のみによって構成されたものであって、情報システムにおける機能を提供する上

では、ファイル管理機能やネットワーク通信機能、GUI やマルチメディアの制御などの高度な機能についても必要となる。

そこで本研究では UC 場所情報システム用に新たな OS 拡張機能「UC Extension」を開発した。UC Extension のアーキテクチャ設計における最大の目的は、組み込みシステム特有の要求 (例：リアルタイム性) を満たしながらも、近年の組み込み製品の高度化に対応できるようなプラットフォームを提供することにある。そこで、基本的な方針として、我々のアーキテクチャでは基本的なコンセプトとして以下を掲げ、設計を行うこととした：

- 組み込み向けリアルタイム OS をベースとする。組み込みデバイスの限られたリソースの中で、例えばデバイス割り込みに対する処理を時間制約を満たすためには、OS カーネルとしてリアルタイム OS を選択する以外の選択はなく、これは前提といっても良い要件である。
- GUI アプリケーションレベルのインタフェースを、利用例の多い GUI プラットフォームと共通化する。一般的に、GUI を用いた組み込みシステムにおいて、組み込みシステム特有の処理部分と GUI 等の高度な処理部分の割合については、システムの高度化が進むにつれて後者の比率が前者を圧倒していく傾向にあるといっている。前者については、外部デバイスの接続を許さない限りは機器に搭載されるデバイスの個数分に相当する記述を行う程度で絞られるが、後者については「GUI をリッチにしたい」、「ブラウザを搭載したい」といった高度化の要求に応じてそのたびに爆発的に複雑度が増していくことが考えられる。したがって、独自の API による組み込み向けの GUI のプラットフォームを提供する代わりに、少なくともインタフェースレベルでは利用例の多いデスクトップ向けの GUI プラットフォームと共通化を行っておくのが、将来的な拡張を考えるとより現実的な解であると考えられる。

なお、人間が GUI を操作する際の時間制約は非常に緩やかなものであり、例えば 100 ミリ秒の表示の遅延があったとしてもほとんど利用者の目からは遅延としては認識できないため、GUI を実現する機能についてはさほど高いリアルタイム性は要求されない。このため、リアルタイム性の実現はデバイス制御に関する部分に注力して行えば十分である。

- 可能な限りコンパクトで軽量の構成とすること。一般的に、組み込みシステム

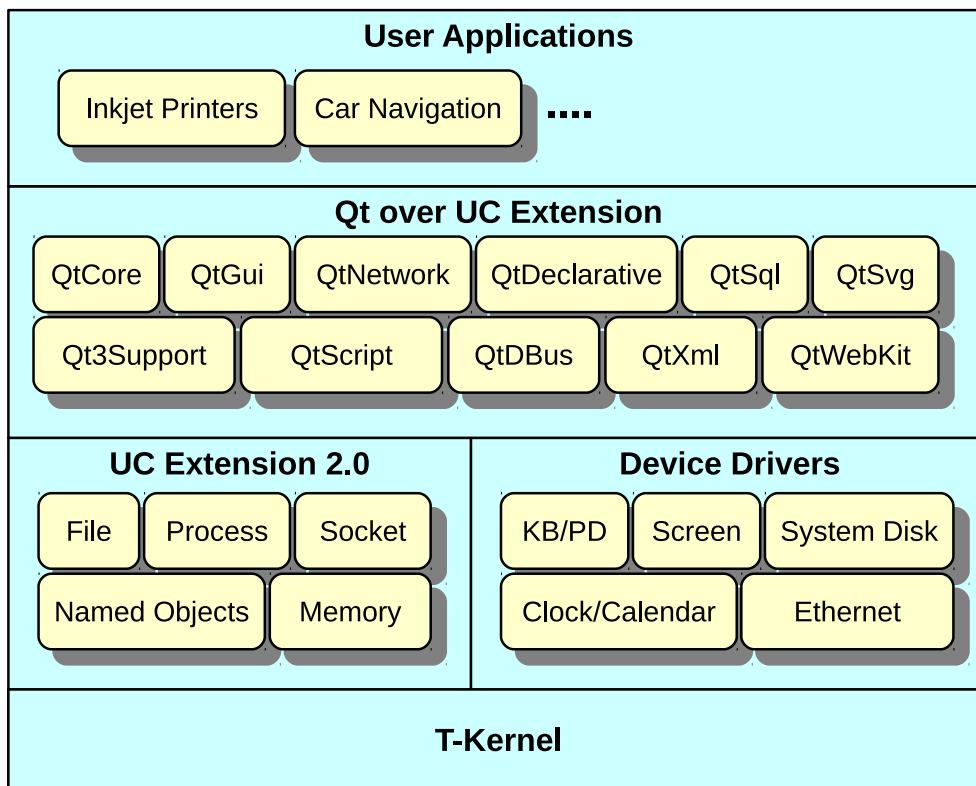


図 5.3: Qt/T-Kernel アーキテクチャ

においてはメモリ容量やプロセッサの能力、ストレージの容量などに制限があることが多い。したがって、PC や Android のような汎用システムとして多数の機能を詰め込む方向性ではなく、組み込み GUI システムに最低限必要な機能を取捨選択して実現する方針とし、それ以外の機能については必要に応じて拡張として追加されるような仕組みとする。

上記のコンセプトに従い、我々は組み込み向け GUI プラットフォーム「Qt/T-Kernel」の設計を行った。Qt/T-Kernel のアーキテクチャは図 5.3 に示される通りである。本プラットフォームでは、組み込みリアルタイム OS として T-Kernel を、GUI のプラットフォームとして Qt [39] をそれぞれ採用した。それらの間をつなぐ OS 拡張機能が、今回新たに設計した UC Extension である。

UC Extension の詳細な設計については [57] にゆずることとし、詳細な設計については本論では割愛する。

5.4 UC フレームワーク

本節では、UC 場所情報システムにおける UC フレームワークの具体的な実装に関する説明を行う。

UC フレームワークの実現においては、各フレームワークとアプリケーションとの間の相互通信のための仕組みが重要となるが、これを実現するための仕組みとして D-Bus [36] を用いた。D-Bus の特徴としては、バイナリデータによる効率的な通信を実現しながらも、通信に型シグネチャを含めることで型安全な通信が実現できること、ブロードキャスト型の通知やリモートプロシージャコール (RPC) を始めとする多様な通信モデルをサポートしている点にある。

本研究における提案システムでは、この D-Bus による通信機能を用いて UC フレームワークの機能をデーモンプロセス (システムサービス) として実装した。UC フレームワークにおけるシステムサービスは以下の 2 つによって構成され、これらのシステムサービスがアプリケーションサービスと D-Bus によるメッセージ通信を行うことで全体としてのシステム動作を実現する。

コンテキストマネージャ

UC フレームワークにおけるコンテキスト推論フレームワークの機能を実現する。

アプリケーションマネージャ

UC フレームワークにおけるアプリケーション間調停機能を搭載したウィンドウマネージャであり、ビヘイビアに基づくアプリケーション制御を行いシステム全体の振る舞いを制御するシステムサービスである。この他省電力管理フレームワークの機能もこのサービスに含まれる。

以下の各節では、UC フレームワークにおけるコンテキストマネージャおよびアプリケーションマネージャの実装について説明する。

5.4.1 コンテキストマネージャ

コンテキストマネージャは、UC フレームワークにおけるコンテキスト推論フレームワークの機能を実現するシステムサービスである。基本的な機能についてはすでに 4.4 節で示したとおりであるが、実際のアプリケーションシステムにおける要件の

実現に加え開発効率とパフォーマンスを効率化するため、以下のような実装上の工夫が加えられている。

永続コンテキスト値のサポート

UC 場所情報システムにおいては、バッテリー切れの際にバッテリーを交換することで再び電源を入れなおした際に、UC 場所情報システムをバッテリー切れの前と同じ状態に復帰させることが求められる。たとえばスタンプラリーのようなアプリケーションを実現するためには、電源が切れた際にも状態を保持できるよう、ディスクなどの不揮発性の記憶に状態を保持しなければならない。

このことから、本コンテキストマネージャではプライマリコンテキストを永続コンテキストと非永続コンテキストの2種類に分類し、前者の指定を行ったコンテキスト値のディスクへの保存・起動時のコンテキスト値の復元を自動的に行うようにした。非永続コンテキストについては再起動の度に値が初期化されることとなるが、その代わりに値更新時のディスクアクセスによるオーバーヘッドを避けることができる。

合成コンテキストの再計算処理の最適化

一つ以上のプライマリコンテキストに変化が生じた時、4.4.3 節に示す合成コンテキストの再計算が必要になる。この処理を効率化するために、コンテキストマネージャはコンテキスト間の依存関係を解析し、どの順序でコンテキストの再計算を行えば良いかをシステム起動時にあらかじめ決定している。具体的には、以下のアルゴリズムを用いてコンテキスト全体の再計算がコンテキスト数に対して線形時間で完了するようにした。

コンテキスト c_2 の値がコンテキスト c_1 の値に依存していることを $c_1 \triangleleft c_2$ と表現すると、 \triangleleft は半順序関係をなしている。ここで、この半順序関係に基づくトポロジカルソート [23] を行いコンテキストを整列する。得られた列の順にコンテキスト値の計算を行うことで、コンテキスト集合から条件に合うものを検索する処理を省略することができる。4.4.3 節で示した方法をそのまま実装した場合、各コンテキスト式の計算時間が定数時間であるとしコンテキストの数を n としたときの最悪の計算量は $O(n \log n)$ となるが、本方式を用いた場合には $O(n)$ となる。

コンテキスト名の ID 化

```

<?xml version="1.0" encoding="utf-8"?>
<contexts xmlns="http://www.ubin.jp/ucsoft/manager/context">
  <context name="sensor.temperature" type="primary"
    persistent="false"/>
  <context name="sensor.dir.pitch" type="primary"
    persistent="false"/>
  ...
  <context name="screen.isWatched" type="composite">
    <value><![CDATA[
      sensor.dir.pitch > 0
      && time.now - tpnl.lastTouched < 1000
    ]]></value>
  </context>
  <context name="user.isWalking" type="composite">
    <value><![CDATA[
      norm(sensor.velocity) > 1.0
      && norm(sensor.velocity) < 6.0
    ]]></value>
  </context>
  ...
</contexts>

```

図 5.4: コンテキスト推論ルールの定義例

アプリケーションシステム開発者は、コンテキストを名前による識別子で定義し参照するが、内部的な処理過程においてはこれを整数型の識別子に変換して処理を行っている。これにより、コンテキストの再計算の際の変数参照の計算効率を向上させている。

コンテキストマネージャに与える外部ポリシー記述としては、XML 形式 [8] によるコンテキスト推論ルールの文法を規定した。コンテキスト推論ルールの記述例は 5.4 に示すとおりとなっている。この例では、以下の 4 つのコンテキストが定義されて

いる。

- プライマリコンテキスト

sensor.temperature

温度センサから得られる温度データを示すコンテキスト。

sensor.dir.pitch

方位 (加速度・地磁気) センサから得られる、ユーザインタフェース端末のピッチ角を示すコンテキスト。

- 合成コンテキスト

screen.isWatched

利用者がスクリーンを注視しているか否かを示す合成コンテキスト。この例では、端末のピッチ角と画面操作の状況とを合わせたコンテキストの推論が行われる。

user.isWalking

利用者がスクリーンを歩いているか否かを示す合成コンテキスト。この例では、ユーザの移動速度による推論が行われる。

プライマリコンテキストに対するルールは、その値はシステムや外部アプリケーションから与えられるため定義に含まれておらず、コンテキストの利用宣言のみとなっている。合成コンテキストの値については、C 言語に似た文法で記述できる式定義が可能となっており、例に示されるような複雑な定義を行うことができる。

なお、コンテキストマネージャの機能を利用するためにはコンテキストマネージャに対する外部 API が必要となる。具体的には、コンテキストの値を設定・取得する API や、コンテキストの変化をイベントによって通知させるための API が他のアプリケーション等に対して提供されなければならない。このため、コンテキストマネージャは D-Bus のサービスを提供するデーモンとして作られており、D-Bus の通信インタフェースを通してコンテキストに関する操作 API を提供している。その具体的な仕様は、付録 A.1 に示すとおりである。

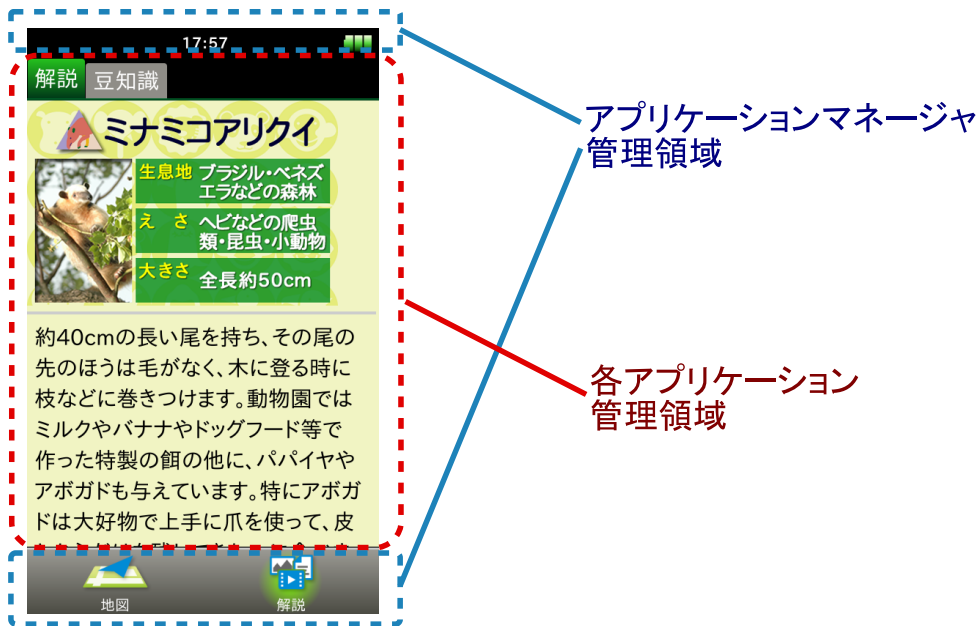


図 5.5: アプリケーションマネージャとアプリケーションの表示管理領域区分

5.4.2 アプリケーションマネージャ

アプリケーションマネージャは、UC フレームワークにおけるアプリケーション間調停機能と省電力制御機能を含んだ、ユビキタスコンピューティングのためのユーザインタフェース端末向けウィンドウマネージャである。一般的なデスクトップシステムにおけるウィンドウマネージャでは、タスクバーにウィンドウの一覧が表示され、その中からユーザが対象となるウィンドウを選択することでアクティブなウィンドウを切り替えるという機能が提供される。一方、本研究で実現するアプリケーションマネージャは、プッシュ型の情報サービスの実現を意図した設計となっており、ユーザの選択操作のみでなく、アプリケーションからのコンテンツプッシュ要求やその他のシステムの状態も考慮に入れたウィンドウ切り替えを行う。

図 5.5 に、アプリケーションマネージャ上でのトップレベルウィンドウの表示例を示す。図中の青枠で囲まれた部分がアプリケーションマネージャで管理される領域であり、赤枠で囲まれている部分がアプリケーションウィンドウが表示される領域である。アプリケーションのウィンドウ表示領域上では、特別なウィンドウ属性を与えない限りはアプリケーションマネージャがウィンドウのフレームやタイトルを

```

function compFun(a, b) {
  // 緊急表示コンテンツがあれば、優先表示
  if (a.critical)
    return -1;
  if (b.critical)
    return 1;

  // 緊急表示コンテンツがなければ
  return b.pagerank - a.pagerank;
}

// 調停ルール本体
function arbitrate(bs) {
  if (bs.length <= 0)
    return new Object();

  bs.sort(compFun);

  // 最も表示優先度が高いと判断されたコンテンツを持つ
  // ウィンドウを前面に移動して表示する
  var o = new Object();
  o.type = "Single";
  o.window_id = bs[0].id;
  o.app = bs[0].app;
  return o;
}

```

図 5.6: 表示調停ルールの記述例

含まない全面表示が行われるように設定されており、携帯端末の GUI に適した表示が行われるようになっている。

アプリケーション表示管理領域に表示を行うアプリケーションは複数存在し、それらがプッシュ動作による通知を非同期に行うことを想定している。たとえば、地図アプリケーションを起動して地図を見ながら移動している時に災害が起きたときに、災害に関する情報がブラウザアプリケーションによりプッシュ通知されるようなシーンが想定される。このようなプッシュ通知を、アプリケーション間の独立性を保ちつつもシステム全体として理想的な動作を実現するために、アプリケーションマネージャでは 4.5 節で示したアプリケーション間調停フレームワークの考えに基づくウィンドウ表示の調停メカニズムを備えている。

アプリケーションマネージャがウィンドウ間の表示調停のために行うポリシーは、JavaScript によるウィンドウ表示調停ルールとして記述される。図 5.6 にその記述例

を示す。ウィンドウ調停ルールの本体は `arbitrate` 関数であり、ウィンドウに対するビヘイビアの変更を受信した場合など、アプリケーションマネージャにおいてウィンドウ表示の調停が必要になった時にこの関数が呼び出される。`arbitrate` 関数は、ビヘイビアの配列 `bs` を引数として受け取り、ハッシュ値として調停結果を返す関数として定義されている。図 5.6 の例では、調停結果のタイプとしては "Single" が指定されているが、この指定により本関数の戻り値は `o.app` で示されるアプリケーションの `o.window_id` で示されるウィンドウを前面に移動して表示するという調停結果として解釈される。この例では `compFun` で定義される比較関数によって最も表示優先度が高いと判断されたウィンドウを、前面に表示する対象として選択している。

なお、アプリケーションマネージャにおけるビヘイビアの型は、キーを文字列とする任意のハッシュ型として規定されており、アプリケーションから調停のために必要な新たな情報を追加することもできる。前述の例ではビヘイビア `b` に対して `b.pagerank`(ページランク [35] で示されるコンテンツの重要度) と `b.critical`(緊急通知であるか否か) の 2 つを与えているが、`b.config`(利用者がアプリケーションの設定を行っている) という属性を追加して、緊急事態でなければその処理を割りこませないというような拡張も容易に実現できる。

また、アプリケーションマネージャには省電力フレームワークの機能も含めている。これは、アプリケーションマネージャが管理するシステムインジケータの表示管理のため、一つのプロセスにまとめたほうが効率的であるという実装上の都合によるものである。省電力制御に関するポリシ記述は XML ベースの省電力制御ルールとして与えられるように作られており、その記述例は図 5.7 に示すとおりである。

アプリケーションマネージャもコンテキストマネージャと同様、その操作のための API を D-Bus インタフェースとして他のアプリケーションやシステムに対して公開している。アプリケーションはこの API を通してウィンドウのビヘイビアの更新通知などを行い、必要なプッシュ通知等の処理を実現することができる。具体的な API の定義については、付録 A.2 にその詳細を示す。

5.5 UC アプリケーションライブラリ

UC フレームワーク上でのアプリケーション開発を用意にするため、本研究ではクラスライブラリ「UC アプリケーションライブラリ」を実装した。UC アプリケーションライブラリは、ユビキタスコンピューティングにおけるユーザ端末上のアプリ

```

<?xml version="1.0" encoding="utf-8"?>
<power xmlns="http://www.ubin.jp/ucsoft/manager/power">
  <rule target="lcd.brightness">
    <pattern>
      <cond>!screen.isWatched</cond>
      <value>10</value>
    </pattern>
    ...
    <pattern>
      <cond>>true</cond>
      <value>100</value>
    </pattern>
  </rule>
  <rule target="position.sampling_frequency">
    ...
  </rule>
  ...
</power>

```

図 5.7: 省電力制御ルールの記述例

ケーション構築のためのライブラリであり、Qt クラスライブラリの自然な拡張として実現されている。このライブラリを用いることで、開発者は UC フレームワークにおけるアプリケーションマネージャ・コンテキストマネージャに対する API を直接用いる代わりに、より抽象度が高く使いやすい API を用いてユビキタスコンピューティングにおけるユーザインタフェース端末上のシステムを構築することができる。

UC アプリケーションライブラリは、以下の 5 つのクラス (内部クラスを除く) から成り立っている。

UcApplication クラス

UC フレームワーク上の基本アプリケーション機能を実現するためのクラス。

UcMainWindow クラス

アプリケーションマネージャで管理されるトップレベルウィンドウの機能を実現するためのクラス。

UcUcode クラス

ucode の値を表現するためのクラス。

UcLocation クラス

緯度・経度・高さ (フロア) によって位置座標を表現するためのクラス。

UcDirection クラス

ロール・ピッチ・ヨーの組により方位・方向を表現するためのクラス。

このうち、UcApplication クラス・UcMainWindow クラスが中核となる機能を提供するクラスであり、これらによってアプリケーションマネージャやコンテキストマネージャに対する操作が抽象化され、各マネージャが提供する生のインタフェースより使いやすい API を提供する。具体的な例としては、ビヘイビアに関する内部的な処理は UcMainWindow クラスの内部で自動的に処理されるため、アプリケーションコードからは単純にビヘイビアの値を変更するだけで十分である。また、現在の場所変化などの特に重要なコンテキストについては専用の API を設けることでアプリケーションの記述を容易にしている。

なお、具体的な API インタフェースのリストは、付録 B に示すとおりとなっている。

5.6 UC 場所情報システム

5.6.1 概要

本節では、UC フレームワーク上で実現されたアプリケーションシステムである UC 場所情報システムについて説明する。UC 場所情報システムは、別所らによる先行研究の成果である「ユビキタス空間識別基盤」[56, 6] の空間モデルに基づき、携帯端末上での情報サービスを行うシステムである。本研究では「東京ユビキタス計画」[49] の協力のもと、東京都内の観光地における UC 場所情報システムの構築を行った。本研究の成果は現在、上野動物園や浜離宮恩賜庭園を含む複数の東京都内の観光地において実運用されている。

UC 場所情報サービスの特徴は、ユーザに貸し出された携帯端末が現在いる場所に基づく最適な情報を自動的に利用者に提示するという点にある。たとえば上野動物園のシステムにおいては、アジアゾウの檻の前に移動した時に自動的にその動物の情報の説明を行うといったサービスが行われる。このように、ユビキタス場所情報システムでは「プッシュ型」の動作を基本として情報サービスを行うが、これは観光地の説明員(ガイド)が観光地内の見どころの情報をその場で説明する動作に近く、端末操作を行う煩わしさを感じることなく情報を入手できることから有用性が高い。その一方で、人間のガイドでも共通の話となるが、利用者が不要としている情報や状況に合わない情報を提供してしまうと、利用者にはストレスを感じさせる結果となる。例えば、歩行者ナビゲーション機能を用いてトイレへに急いでいるときに、アジアゾウのコンテンツがプッシュ通知されるような事象は、一般的には利用者には歓迎される挙動ではない。このため、UC 場所情報システムにおいてはセンサ値などから得られる物理的な状況、そこから推定される利用者の状況、さらにアプリケーションやシステム内部の状態を含めた制御を行う必要があり、前の章で示した UC フレームワークのケーススタディとして適切なアプリケーションシステムであると考えられる。

さらに、UC 場所情報システムにおいては開発効率やシステムの柔軟性も重要な課題となる。利用者からのフィードバックに基づき、動作の継続的な改善は頻繁に行われるものであるから、これを効率的に行う仕組みが求められる。ユビキタス場所情報サービスの運営者は、その観光地に応じたカスタマイズを希望する場合も多いが、さらには独自の機能を提供することで差別化を行うことも多く行われている。図 5.8 は、上野動物園のユビキタス場所情報サービスに 2013 年 2 月追加されたクイズ機能の画面例である。上野動物園では特に家族連れの観光客が多いことから、子供により動物に親しんでもらうため、クイズを通して動物について学ぶ機能の追加を希望された。このような機能追加への要求に対して、開発効率の改善は重要な課題であることから、UC 場所情報システムでは UC フレームワークを用いることとし、それに基づく開発効率の高い柔軟なシステム構築を行うこととした。

5.6.2 アプリケーション構成

UC 場所情報システムでは、単に観光地の見どころ情報を表示するのみではなく、地図によるナビゲーション機能や拡張現実 (AR) の技術に基づくパノラマビュー機能、



図 5.8: 上野動物園場所情報サービスに追加されたクイズ機能

コース選択、履歴表示機能などを持つ統合的な観光ガイド機能を有している。UC 場所情報システムにおいては、これらは各機能を実現する個別のアプリケーションによって実現する。その上で、全てのアプリケーションは UC フレームワークによって管理され、アプリケーション間の調停処理によりそれらをシステムとして最適な全体動作を実現できるようまとめ上げることで、システム全体として求められる挙動を実現している。以下は、UC 場所情報システム上に実装されたアプリケーションである。

UC ブラウザ

その場所に応じた見どころコンテンツを、現在の状況に応じてプッシュ型通知に基づき表示するアプリケーション

UC マップ

地図上に現在位置を表示するほか、ガイドコースに応じて地図上にルートガイドを表示するアプリケーション

UC パノラマビュー

現在位置から見える風景に対し、地物の情報を重ねあわせて表示するアプリケーション

UC ホーム

各種環境設定 (言語設定など) や、ガイドルート (コース) 等の選択を行うためのアプリケーション

UC 履歴管理

過去にプッシュされた見どころ情報を、後から選択して閲覧するためのアプリケーション

以下、順に詳細を示す。

5.6.3 UC ブラウザ

UC ブラウザは現在いる場所に紐付けられたコンテンツを場所情報データベースから検索し、プッシュ型の情報サービスとして自動的に表示・再生を行うアプリケーションである。動作画面例は図 5.9 に示すとおりであり、その大まかな処理の流れは以下のとおりである。

- ユーザが設定画面 (UC ホーム) により設定したガイドコースの設定に従い、利用者が新たな場所領域に入った情報をコンテキストとして受信し、その場所に対する地物情報コンテンツの検索を行う
- マッチしたコンテンツがあったとき、そのコンテンツを自動的に表示する

ガイドコースの設定によって、UC 場所情報システムにより提示されるコンテンツや、ガイドの所要時間を切り替えることができる。例えば、上野動物園のシステムにおいては「自由散策」モード、「絶滅危惧種テーマ散策」モード、「動物クイズ」モードがそれぞれ用意されており、1つ目を選んだ場合には全ての動物の情報がその場所で表示されるが、残りのモードにおいては特定の動物の前で自由散策モードとは異なったより詳細なコンテンツや、クイズなどの付加価値のついたコンテンツを楽しむ事ができる。

5.6.4 UC マップ

UC マップは、現在利用者がいる位置を地図上に表示するほか、地図上での現在のガイドコースに従った順路案内や地物検索 (トイレ検索など) を実現するアプリケーションである。動作画面例は図 5.10 に示すとおりである。

現在位置の変化をコンテキストとして受信し、現在位置を地図上に表示する処理を繰り返すのが UC マップの基本動作であるが、その他にも他のアプリケーションからの要求に基づきナビゲーションの表示を行うことや、コースに応じた順路を表示する機能も実現している。

5.6.5 UC パノラマビュー

UC パノラマビューは、現在利用者がいる場所から見える風景に対し、地物の情報を重ねあわせて表示するアプリケーションである。動作画面例は図 5.11 に示すとおりであり、左側が風景に情報をオーバーレイ表示させた画面となっており、右側がオーバーレイ表示されたアイコンをタップして詳細な情報表示の画面を呼び出した際の画面である。

現在位置に基づくパノラマ画像を検索・取得し、そこに地物情報をオーバーレイ表示させるのが基本動作であるが、向いている方向に応じて画像の表示を回転させるため端末の方位についてもコンテキストとして受け付けを行う。



図 5.9: UC ブラウザの画面



図 5.10: UC マップの画面



図 5.11: UC パノラマビューの画面



図 5.12: UC ホームの画面

5.6.6 UC ホーム

UC ホームは、UC 場所情報システムのホーム画面として各種設定機能やアプリケーション切り替え等の機能を提供するプログラムであり、システムコンテキストの設定の観点から非常に重要なコンポーネントである。動作画面例は図 5.12 に示すとおりである。

初期起動時に UC ホームアプリケーションは常にトップに表示され、ユーザはその画面を通して言語設定およびガイドコース設定を行うこととなる。その間、初期設定が完了するまでの間はアプリケーションマネージャによる調停処理により、UC ブラウザや UC マップを含む他のアプリケーションに制御が渡ることはない。

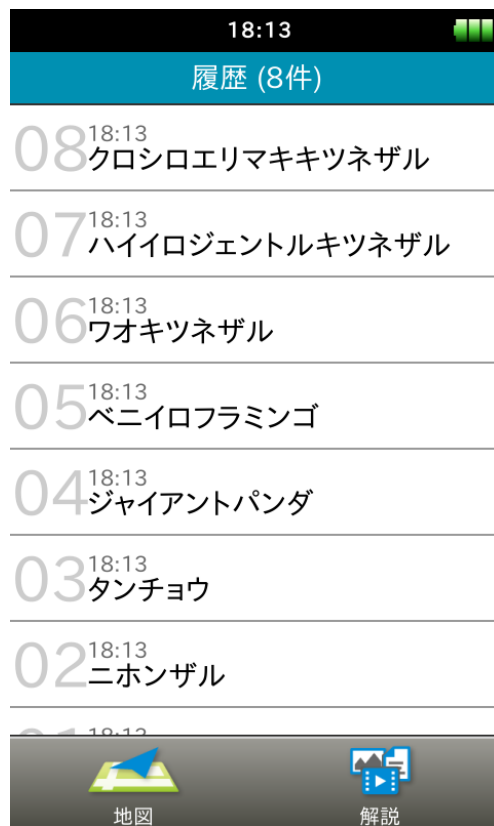


図 5.13: UC 履歴管理の画面

5.6.7 UC 履歴管理

UC 履歴管理は、これまでに利用者が見てきた観光地の見どころコンテンツの履歴を表示し、過去のコンテンツの表示を行うためのアプリケーションである。画面動作例は図 5.13 に示すとおりであり、上野動物園においては UC 場所情報システムを持って訪れた動物の一覧が表示される。ここから、動物の名前のエントリを指でタップすることで、過去に見た動物の情報を再び閲覧することができる。

第 6 章

評価

本章では、UC フレームワークを複数の観点から評価し、本研究の有用性についての検証を行う。

6.1 開発効率

本節では UC フレームワーク上でのアプリケーションの開発効率について、第 5 章で示した UC 場所情報システムを例にあげ、アプリケーションシステム開発における高い開発効率が実現できるかについての検証を行う。

6.1.1 コード量

まず第一に、外部ルールや設定ファイルを含まないアプリケーションシステムのコード量について評価する。表 6.1 に、UC フレームワーク上に実現された各アプリケーションソフトウェアのサイズを示す。コードサイズは空行・コメント行を含めたものである。

表 6.1 の結果に示されるとおり、UC フレームワーク上の実用的なアプリケーションシステムはいずれも数千行程度の C++ のコードで実現されており、十分に開発可能かつ十分にメンテナンスを維持できるコード量に抑えられていることが分かる。

次節以降では、実際に使用したアプリケーションプログラムのコードや外部記述を参照し、より詳細な観点にもとづき開発効率についての議論を深める。

表 6.1: UC フレームワーク上のアプリケーションサイズ

アプリケーション	コードサイズ (行数)	オブジェクトサイズ (KB)
UC ブラウザ	1305	151
UC マップ	5196	1922
UC パノラマ	2082	134
UC ホーム	893	95
UC 履歴管理	1015	107

6.1.2 アプリケーション記述

本研究では、UC フレームワークをアプリケーションから使いやすくする仕組みとして UC アプリケーションライブラリを提供している。これにより、アプリケーションマネージャやコンテキストマネージャとの通信処理の詳細を気にすることなく、開発者はアプリケーション内部の処理を効率的に記述することが出来る。

図 6.1 に、UC ブラウザにおけるコンテンツプッシュ処理の実際のコード例を示す。このメソッドは UC アプリケーションライブラリで定義される以下のシグナル (イベント) に対するスロット (イベントハンドラ) として定義されており、利用者の場所変化があった際の処理を記述したコードとなっている。

```
void UcApplication::enteredPlace(UcUcodeSet ucodeSet);
```

この記述例に関する処理の概要は以下の通りである。

1. 新たに利用者が入った場所集合全体に対して、ガイドコースおよび現在の状態に応じて関連付けられた観光地の見どころコンテンツ (タブセット) を SQL クエリにより検索する。
2. コンテンツが見つかった場合には、ガイドコースの状態遷移が必要ならばそれを行い、コンテンツ (タブセット) の内容を更新する。さらに、ビヘイビアの変更をアプリケーションマネージャに通知し、システム全体としての新しいコンテンツのプッシュ通知を要求する。

なお、図 6.1 中のコード例において、UC フレームワーク固有の記述は以下の 2 文のみである。この部分ではアプリケーションのビヘイビアの更新とそのアプリケー

ションマネージャへの通知を行っており、UC フレームワークがない場合はここで他のアプリケーション等の通信を行い、システムとして正しいアクションを決定するメカニズムがこの部分に必要となる。

```
behavior()["push_time"] =
    QDateTime::currentDateTime().toMsecsSinceEpoch();
ucApp->notifyBehavior(this, (int)-1);
```

逆に言えば、これ以外の部分については ucode を扱っている点を除けば一般的な GUI アプリケーションのプログラムと何ら変わりのない処理となっている。すなわち、UC フレームワークとアプリケーションライブラリの導入により、ユビキタスコンピューティングのアプリケーション開発におけるコード記述が一般的な GUI アプリケーションとほぼ変わらない水準の複雑度に押し下げることが出来たことを示している。

ここまでの評価により、アプリケーション開発におけるコード記述が、質・量の両観点から一般的な GUI アプリケーションと同程度の水準まで効率化出来たことが示された。しかしながら、UC フレームワークにおいては一般的な GUI アプリケーションの開発では存在しない、フレームワークに与える外部ポリシ記述が必要であり、開発コストの一部はコンテキスト推論ルール・アプリケーション間表示調停ルールの記述作業に引き継がれたと考えられる。そこで、本節に続く 6.1.3 節・6.1.4 節において、コンテキスト推論ルール・アプリケーション間表示調停ルールに関する記述効率に関する検証を行う。

6.1.3 コンテキスト推論ルール

コンテキスト推論ルールの記述例を図 6.2 に示す。このルールは、浜離宮恩賜庭園でのユビキタスガイド端末のために記述されたルールを一部省略して抜き出したものであり、トイプログラムにおける記述例ではなく、実運用されているシステムにおける記述を示したものである。

UC フレームワークにおいてはコンテキスト推論ルールの導入により、アプリケーションやシステムにおけるセンサ値等の状況における状況判断を簡潔に行うことができる。たとえば、アプリケーションマネージャではバッテリー残量が減った際にシステムを強制シャットダウンする必要があるが、この条件を推論するルールとして、図 6.2 では以下のような規則が定義されている。このルールにより、アプリケーショ

```

bool MainWindow::onEnteredPlace(const UcUcodeSet& ucodeSet)
{
    if (ucodeSet.isEmpty() || ucApp->getCourse().isEmpty())
        return false;

    UcUcodeSet::const_iterator iend = ucodeSet.end();
    UcUcodeSet::const_iterator ibegin = ucodeSet.begin();
    QString condStr;
    for (UcUcodeSet::const_iterator i = ibegin; i != iend; ++i) {
        if (i != ibegin)
            condStr += " OR ";
        condStr += "P.unicode=\'" + i->toString() + "\'";
    }

    // 新たに入った場所 (ucode) に関連付けられたタブセットを検索する
    QSqlQuery query(m_db);
    query.prepare(
        "SELECT R.tabset, P.unicode, R.next_state "
        "FROM relationbetweentabsetandplace as R "
        "INNER JOIN place as P ON P.unicode = R.place "
        "WHERE (" + condStr + ") "
        "AND P.placeclass = \'group\' "
        "AND (R.course = ? OR R.course IS NULL) "
        "AND (R.current_state = ? OR R.current_state IS NULL)");
    query.bindValue(0, ucApp->getCourse());
    query.bindValue(1, m_tourState);
    if (query.exec() && query.next()) {
        // 最初の一つを使って状態更新・表示コンテンツの更新を行う
        int tabset = query.value(0).toInt();
        UcUcode place(query.value(1).toString());
        bool has_next_state;
        int next_state = query.value(2).toInt(&has_next_state);

        if (has_next_state && next_state >= 0)
            setTourState(next_state);

        // Behavior を変更して通知する
        behavior()["push_time"] =
            QDateTime::currentDateTime().toMSecsSinceEpoch();
        ucApp->notifyBehavior(this, (int)-1);

        setTabSet(tabset, ucApp->getLanguage());
        return true;
    }
    return false;
}

```

図 6.1: 利用者の位置変化に対するコンテンツプッシュ処理記述


```

<?xml version="1.0" encoding="utf-8"?>
<contexts xmlns="http://www.ubin.jp/ucsoft/manager/context">
  <!-- 現在地に相当する場所集合 -->
  <context name="sensor.location.place" type="primary"
    persistent="false">
    <type>ucodeSet</type>
    <init></init></context>

  <!-- システム言語設定 (案件によって、persistent か否かは変わってくる) -->
  <context name="config.lang" type="primary" persistent="true">
    <type>string</type>
    <init>jpn</init></context>

  ....(中略)....

  <!-- ユーザが端末を注視している (composite) -->
  <context name="user.watchingScreen" type="composite">
    <type>bool</type>
    <value><![CDATA[sensor.dir.pitch > 0]]></value></context>

  <!-- バッテリ残量低下状態 (composite) -->
  <context name="power.battery.low" type="composite">
    <type>bool</type>
    <value><![CDATA[power.battery.remain < 10
      && !power.externalSupplyConnected]]></value>
  </context>

  <!-- 現在のコース名 (primary) -->
  <context name="ubinavi.tour.course" type="primary"
    persistent="true">
    <type>string</type>
    <init></init></context>

  <!-- ツアーの実行状態 (primary) -->
  <context name="ubinavi.tour.status" type="primary"
    persistent="true">
    <type>int</type>
    <init>1</init></context>

  <!-- 貸出場所の名称 (primary) -->
  <context name="ubinavi.tour.rentalplace" type="primary"
    persistent="true">
    <type>string</type>
    <init>oote</init></context>
</contexts>

```

図 6.2: 浜離宮恩賜庭園向けシステムのコンテキスト推論ルール (一部省略)

ンマネージャ側では、コンテキスト `power.battery.low` が `true` となった時にシステムのシャットダウン処理を開始すれば良い。

```
<!-- バッテリ残量低下状態 (composite) -->
<context name="power.battery.low" type="composite">
  <type>bool</type>
  <value><![CDATA[power.battery.remain < 10
                    && !power.externalSupplyConnected]]></value>
</context>
```

コンテキスト推論ルールの記述性については、上記の例からもわかるように簡潔である。この例では、バッテリー残量が10%を下回っており、外部電源 (AC 電源) の接続がない特にバッテリー残量が低下していると推論しているが、右辺値をコンテキストの式で表現することによって簡潔かつ分かりやすい記述を実現できている。ここで、バッテリー残量の判断をたとえば現在の消費電力実績に基づいて判断するロジックを実現したいと考えた時にも、現在の消費電力コンテキストを用いた式に書き換えを行うことで、このような変更に対応することが出来る。

6.1.4 アプリケーション間表示調停ルール

アプリケーション間の表示調停ルールの記述例を図 6.3 に示す。調停のためのルールは、関数 `arbitrate` で定義されるとおりであり、その処理は比較関数 `compFun` で示される条件に基づき最も優先して表示を行うべきビヘイビアを選択し、そのビヘイビアに対して単一の全面ウィンドウでの表示を行うというものである。これによりウィンドウが自動的に切り替わったとき、利用者にとってはプッシュ通知が発生したように見える。

図 6.3 において、比較関数 `compFun` で示される優先ウィンドウの決定過程は以下の通りとなっている。

- 初期設定が完了していない時、UC ホームアプリケーションを常に優先表示する。
- それ以外の場合で緊急表示コンテンツがある場合には、その表示を常に優先する。

```

// gui_activated_time に関して降順ソートするための比較関数
function compFun(a, b) {
  // 初期設定が未完であれば、必ず優先的に表示
  if (a.app == "home" && !a.initial_config_finished)
    return -1;
  if (b.app == "home" && !b.initial_config_finished)
    return 1;
  // 初期設定が完了した状態で緊急表示コンテンツがあれば、優先表示
  if (a.critical)
    return -1;
  if (b.critical)
    return 1;

  var time_a = a.gui_activated_time;
  var time_b = b.gui_activated_time;
  if (a.hasOwnProperty("push_time") && a.push_time > time_a)
    time_a = a.push_time;
  if (b.hasOwnProperty("push_time") && b.push_time > time_b)
    time_b = b.push_time;

  // 最後にプッシュされたコンテンツを先頭 (bs[0]) に移動
  return time_b - time_a;
}

// 調停ルール本体
function arbitrate(bs) {
  if (bs.length <= 0)
    return new Object();
  bs.sort(compFun);
  var o = new Object();
  o.type = "Single";
  o.window_id = bs[0].id;
  o.app = bs[0].app;
  o.disable_tool_button = (bs[0].critical || (bs[0].app == "home"
    && !bs[0].initial_config_finished))

  return o;
}

```

図 6.3: 上野動物園向けシステムの表示調停ルール

- それ以外の場合、最後に GUI 操作またはコンテンツプッシュが行われたウィンドウを優先して表示する。

この例に示されるように、UC 場所情報システムにおけるアプリケーションの表示調停ルールは実用システムにおいても 60 行程度の簡単な記述によって実現することができる優れた記述性を持っており、アプリケーションシステム開発者は少ない開発コストでシステム全体としての最適動作を実現することができる。ここで仮に、新たなアプリケーションが追加され特殊な制御を行う必要が生じた場合にも、他のアプリケーションを書き換える必要性を生じることなく、調停ルールの書き換えのみですべての作業が完結するため、システム構成の変更に対する柔軟性も高い。

6.1.5 UC12 システムとの比較

UC 場所情報システムと同様のサービスを実現するためのシステムとして、YRP ユビキタス・ネットワークング研究所によって開発された「UC12 システム」が挙げられる。本節では、本研究で提案するシステムと UC12 システムにおける実現メカニズムの比較を簡潔に行い、それらの利害得失について議論を行う。

UC12 システムにおけるシステムの全体アーキテクチャは図 6.4 に示すとおりとなっている。アプリケーションはプラグインマネージャにより管理され、それを通して ucode の受信や GUI 操作を含む各種のイベントを受け取ることができるようになっている。プラグインマネージャは、利用者の現在位置の変化に対応する場所を受信すると、その ucode をもととしたコンテンツの検索をシステムレベルで実行し、アプリケーションシステム全体としての動作を決定する。その大まかな流れは以下のとおりとなっている。

1. 与えられた場所 (ucode) に対し、特殊な条件処理を行うためのスクリプト言語による ucode の変換処理を行う。変換処理が適用されない ucode については、与えられた場所の ucode がそのまま返される。
2. 変換された ucode をキーとして、場所コンテンツの関連付けデータベースに対する検索を行い、コンテンツの URI およびそのコンテンツに対応するアプリケーションタイプの組のリストを得る。
3. 得られたリストそれぞれの項目を一つのタブウィンドウに対応付け、各タブ内でそれぞれのコンテンツをアプリケーションタイプに対応するアプリケーション

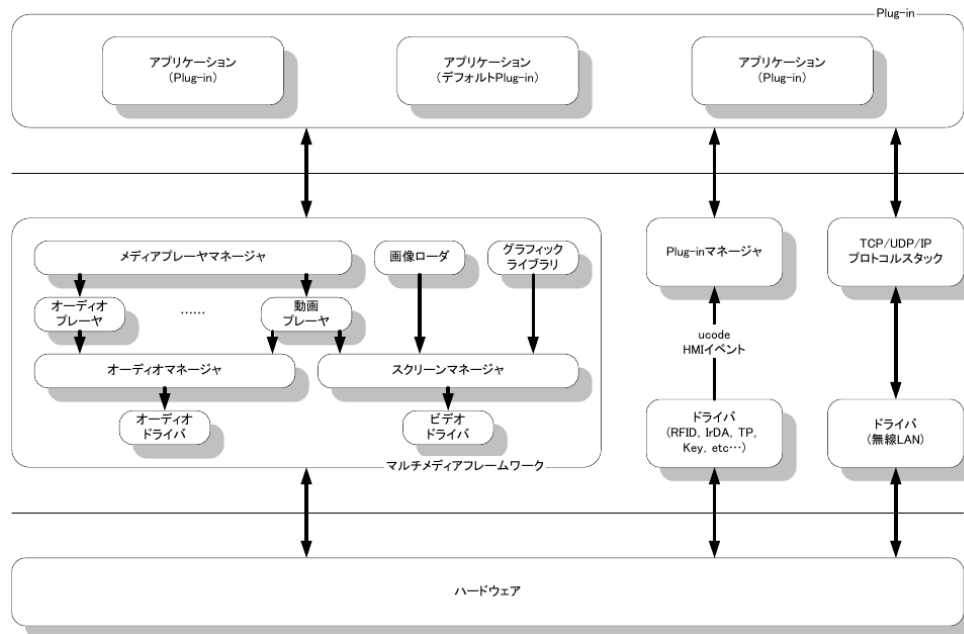


図 6.4: UC12 システムの全体アーキテクチャ

ンを用いて開き、コンテンツ URI で示されるコンテンツの表示を行う。

すなわち、UC12 システムにおいてはシステム全体としての最適動作を実現するために、アプリケーション機能の一部をシステム側であるプラグインマネージャに持たせ、そこで必要な処理を行うという仕組みになっている。たとえば、ガイドコースや言語設定の完了していない時点ではブラウザによるコンテンツのプッシュ通知を行わないといった動作の実現が必要となるが、コンテンツの検索処理がブラウザ側でなくシステム側のデータベースによって実現されていることからシステムレベルでの判断を行うことが出来る。すなわち、本研究での提案システムと同様にシステム全体の動作を実現することが出来るように構成されている。

なお、この方式ではアプリケーション機能の多くがシステム側で実現されることとなるため、システム側で実現する情報処理を柔軟かつ容易に行えるようにする仕組みが必要となる。そこで、UC12 システムにおいては、1. の段階において特殊な条件処理を「ucode ルール」というスクリプト記述によりカスタマイズできるようにし、システム側で実現される情報処理を柔軟に行えるようにしている。図 6.5 に、ucode ルールの実際の記述例を示す。この例は、浜離宮恩賜庭園における ucode ルールの

表 6.2: UC12 システムと提案システムのシステム動作記述のサイズ

システム	行数	サイズ (バイト)
UC12 システム (上野動物園)	385	15985
提案システム (上野動物園)	60	2531
UC12 システム (浜離宮)	1064	35214
提案システム (浜離宮)	59	2421

記述例の一部を抜き出して示したものであり、貸出用ユビキタス端末の貸出・返却地点における特殊な処理の記述例を示している。現在のガイドコース設定とガイドツアーの往路・復路のいずれかにあるかによって、再生すべきコンテンツを切り替えているというのが主な処理内容である。

ここまでの説明から分かるように、本研究での提案方式と UC12 システムにおける処理の流れは大幅に異なっている。UC12 システムにおいてはツアー機能などのアプリケーション側機能の一部をシステム側で実現し、システム全体のルールを記述する方式を導入することでシステム全体の動作を柔軟に行えるようにしている。一方、本研究での提案システムにおいては、アプリケーション側でそれぞれの判断に基づいた制御を実現した上で、それらの調停により全体としてのシステム動作を実現している。UC12 システムの方式では、アプリケーションシステムが複雑化するにつれシステム側で行わなければならない処理が増加するが、本研究における提案方式ではシステム側の処理が調停のみに限定されることから、システム側の処理の複雑化が抑えられシステム構築におけるスケーラビリティの向上が可能になると考えられる。

これを評価するため、UC12 システムと本研究における提案手法のシステム動作の記述に関する複雑度の比較を行った。システム動作の記述は、前者においては ucode ルール、後者においてはアプリケーション間表示調停によって行われる。結果は表 6.2 に示すとおりであり、本研究における提案方式ではシステム側の処理の複雑度が大幅に抑えられ、システム構築におけるスケーラビリティの向上が可能となることが確認された。

```

....

def PLACE_H001 "00001C00000000000002000000162CE2"
....

PLACE_H001:
  if (useState == USESTAT_BOOTED)
    useState = USESTAT_SETTING
    @location_appli = LOCATION_BROWSER
    @rental_place = "oote"
    @guide_course = "free"
    @guide_route = ROUTE_FREE
    logf "START=oote"
    @dis_func_button = 1
    $$ = SET_PROFILE_HTML
  elseif (useState == USESTAT_SETTING)
    $$ = ""
  elseif (route == ROUTE_LITE_OOTE2 || route == ROUTE_ADVANCE_OOTE2)
    ## 大手門スタートでゴールした場合、終了動画を見せる
    $tabset = ""
    $tabset = "00001C000000000000020000001647F9,"
    @dis_func_button = 1
    $$ = FINAL_HTML
    logf "END"
  elseif (route == ROUTE_LITE_NAKA2 || (route == ROUTE_ADVANCE_NAKA))
    ## 中の御門スタートでゴールした場合、大手門動画 → 終了動画を見せる
    $tabset = ""
    $tabset = "00001C0000000000000200000016483C,"
    $$ = LAUNCH_SIGHTS_MOVIE
    sights_content = FINAL_HTML
    logf "END"
  endif
  cmap = CMAP_H001
  panorama = PANOV_H001

....

```

図 6.5: ucode ルールの記述例

6.2 最適動作の実現性

6.2.1 評価方法

本節では実運用事例をもとに、UC フレームワークがシステム全体最適動作を実現し、利用者の観点からみた利用感の向上に寄与するかどうかについて評価を行う。ユビキタス場所情報システムの納入先の組織¹の一つにご協力をいただき、利用者に対して行っているアンケートの結果を提供していただいたので、この結果を元に評価を行う。なお、アンケートの実施は都内観光地の管理主体が一般の観光客に対して行っているものであり、システム開発者の名目で行っているわけではない。このため、アンケート回答者の母集団は本システムに対して中立性の高い人々によって構成され、結果の信頼性については高いことが期待できる。

ここで評価を行ったユビキタス場所情報システムの納入先では、以下のような運用がなされていた。

- 2010年度から2011年度まで、前節で挙げたUC12システムに基づく場所情報サービスを提供していた。
- 2012年度に、本研究の成果に基づくUC場所情報システムへの置換えを行い、少なくとも2013年12月現在に至るまで同システムの運用を継続している。

なお、両システム間には提供するコンテンツを含むサービスの本質的な内容自体には大きな違いはない。このため、プッシュ動作を中心とするシステム動作の最適性の実現こそが、ユーザからの利用感を左右する特に重要な要素であるとみなすことができる。

行われたアンケートは自由記述の項目を含む多くの項目からなるが、本節におけるユーザにとっての利用感に関する項目から自由記述の項目を抜き出すと、以下の5つの項目に絞られる。いずれの項目についても、UC12システム・提案システムの両方について少なくとも100件以上の有効回答を得ている。

- Q1. 機器の操作はいかがでしたか？
- Q2. ユビキタスガイドを使うことによって、より観光を楽しむことができましたか？

¹匿名希望のため名前は伏せる。

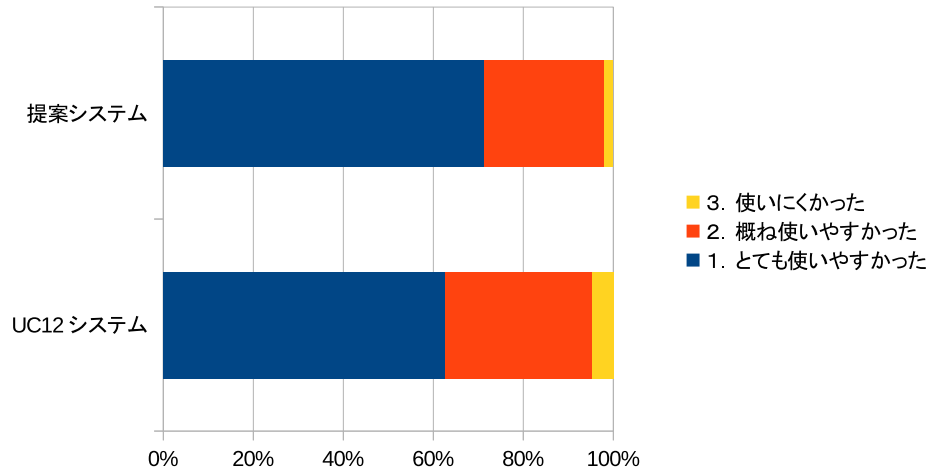


図 6.6: 機器の操作性についての感想

- Q3. 再び訪れたときに、この携帯端末ガイドをお使いになりますか？
- Q4. この携帯端末ガイドが有料化された時には、いくらまでなら利用しますか？
- Q5. 将来、ユビキタスガイドを使える環境が、道路や街なかに展開されることを望みますか？

以降の各節では、それぞれのアンケート項目についての回答の集計結果とそれに対する考察を示す。全体としては、すべての設問において利用者にとっての利用感が向上を読み取ることができ、UCフレームワークがシステムの最適動作を実現するのに有意義であると結論づけられる。

6.2.2 機器の操作性に関して

まず、機器の操作性に関する設問「Q1. 機器の操作はいかがでしたか？」についてのアンケート集計結果は、図 6.6 に示されるとおりとなった。「1. とても使いやすかった」という回答の割合が 62%から 71%へと大幅に増えているほか、「3. 使いにくかった」の割合が 4%から 2%へと減少しており、提案システムにおいて利用感の有意な改善を見てとることができる。

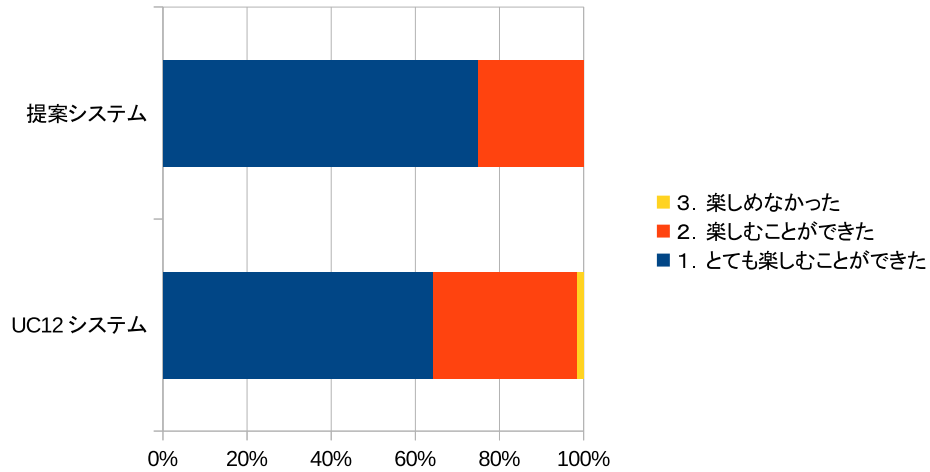


図 6.7: サービスの利用の楽しさについての感想

6.2.3 利用時の楽しさに関して

観光時に機器を利用した際の楽しさに関する設問「Q2. ユビキタスガイドを使うことによって、より観光を楽しむことができましたか?」の設問についての結果は、図 6.7 に示されるとおりとなった。「1. とても楽しむことができた」の回答割合が 64%から 75%へと大幅に増えているほか、「3. 使いにくかった」の割合が 2%から 0%へと減少しており、提案システムにおいて改善がみられる。

6.2.4 再利用の希望に関して

同じ観光地に再訪した時、もう一度ユビキタスガイドを利用したいかについての設問「Q3. 再び訪れたときに、この携帯端末ガイドをお使いになりますか?」の設問についての結果は、図 6.8 に示されるとおりとなった。「1. とても楽しむことができた」の回答割合が 46%から 52%へと増えている。「3. 特に使いたいとは思わない」の割合が 8%から 7%にわずかながら減少している。「3. 特に使いたいとは思わない」と回答した人の多くは、一度システムを利用して観光を行ったので、二度目はシステムなしで観光してみようと考えた可能性が高いと考えられる。

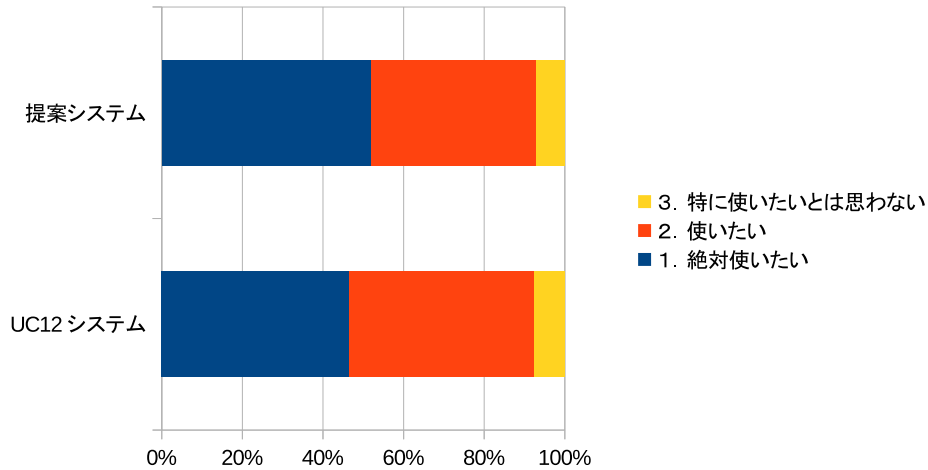


図 6.8: 再度利用したいか否かについて

6.2.5 有料化の際の支払い金額の上限に関して

ユビキタス場所情報サービスは現在無料貸出しによる形態で運用されているが、これが仮に有料化されたときに、いくらまでを上限として金銭を支払うかについての設問「Q4. この携帯端末ガイドが有料化された時には、いくらまでなら利用しますか?」の設問についての結果は、図 6.9 に示されるとおりとなった。大幅な増加とはいえないものの、デフレの経済環境の中においても全体的には金銭価値が向上していることがわかる。

6.2.6 道路や街なかへの展開に関して

最後の設問「Q5. ユビキタスガイドを使える環境が、道路や街なかへ展開されることを望みますか?」に対する回答結果は、図 6.7 に示されるとおりとなった。「1. 庭園の外でも利用環境が整ってほしい」の回答割合が 43%から 48%へと大幅に増えている一方で、「3. 展開は望まない」の割合が 9%から 8%へとわずかに減少しており、提案システムにおいて改善がみられる。

本項目については「3. 展開は望まない」の割合が 8%となっているなど、道路や街なかでの展開に関して懐疑的な意見が残っていることが分かる。これは、道路で

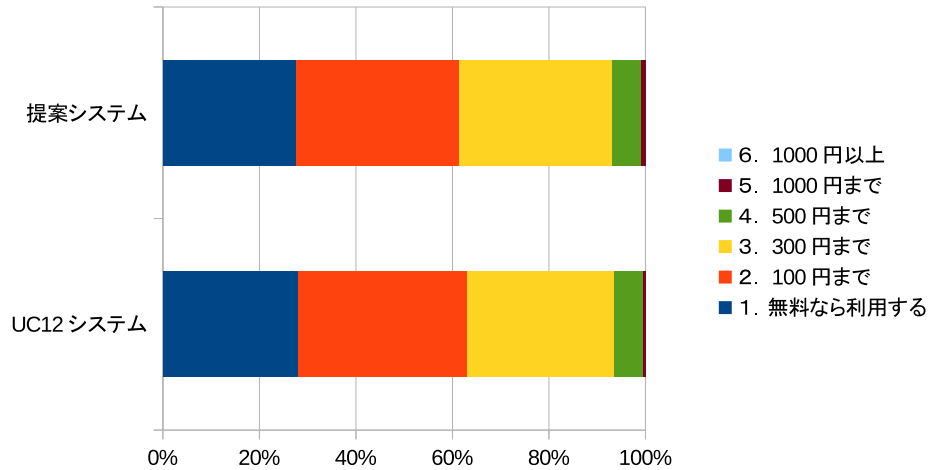


図 6.9: 有料サービスとなった場合の支払い可能金額について

ユーザ端末のスクリーンに注視したまま歩くのは危ないといった考えによるものであると推測され、道路や街なかでは音声を用いるなどの別のサービス形態が望ましいと考えられる。

6.3 パフォーマンス

UC 場所情報システムにおいて、現在位置の変化に対して UC ブラウザがプッシュ通知を行うまでの処理の流れは以下のとおりとなっている。

1. 位置座標に関するセンサ値の変化情報がコンテキストマネージャに入力される。
具体的には、方位センサの値・GPS 座標の変化・受信した RFID ビーコンの ID がコンテキストマネージャに入力される。
2. コンテキストマネージャに入力されたセンサ値の変化に基づき、その値に依存した全てのコンテキスト値の再計算を行う。
3. コンテキストマネージャは変化したコンテキストを全てのアプリケーションに通知する。
4. UC ブラウザはコンテキストの変化通知を受信する。

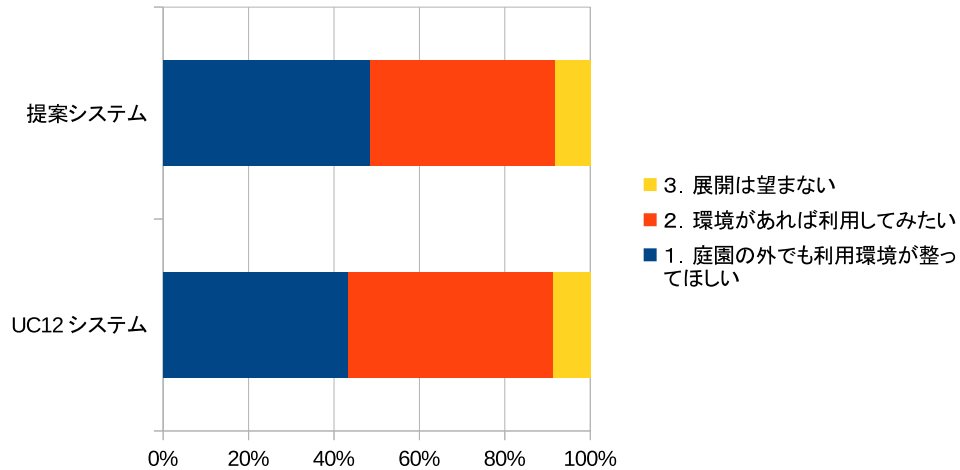


図 6.10: 本システムの道路や街なかへの展開について

5. 受信したコンテキストの変化通知の中に、現在いる場所領域の変化通知が含まれていた時には、新たに入った場所情報を用いて地物情報データベースからのプッシュ通知すべきコンテンツの検索を行う。
6. UC ブラウザは検索の結果得られた表示すべきコンテンツをロードし、表示する。
7. UC ブラウザは自アプリケーションのビヘイビアを更新し、アプリケーションマネージャにプッシュ通知の要求を発行する。
8. アプリケーションマネージャはプッシュ通知の要求を受け取り、アプリケーション間の調停を行った上で UC ブラウザへの画面切り替えを行う。

この処理において、1. から 3. にかけてがコンテキストマネージャによる処理時間、4. から 5. にかけてが各アプリケーションの固有処理に要する時間、6. から 7. がアプリケーションマネージャによるウィンドウ表示調停に要する時間に相当している。

これらの処理に消費する実時間を、複数の位置座標においてそれぞれ計測した時間の平均は図 6.11 のとおりとなった。結果から分かるように、ほとんどの時間はブラウザ内でのコンテンツの検索と、その表示によって費やされていることが確認で

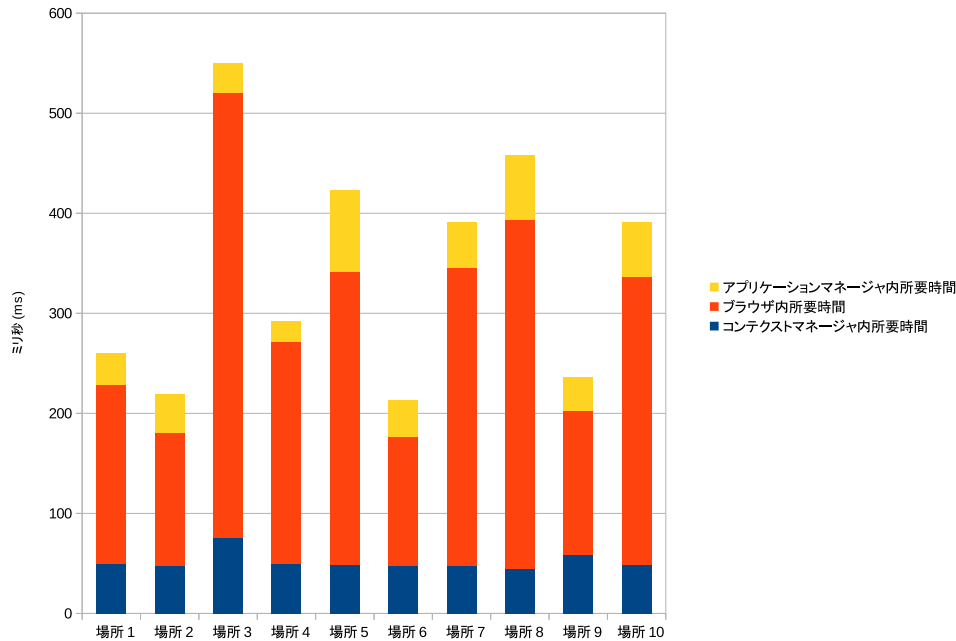


図 6.11: 各地点における現在位置変化に対するシステム処理時間 (実時間)

きる。特筆すべきは、プロセス間通信により処理を行うコンテキストマネージャ・アプリケーションマネージャにおいて、その処理時間が両者を合わせても平均で 100 ミリ秒で収まっていることである。これは、UC Extension におけるプロセス間通信機能の実装の効率と、コンテキストマネージャにおけるコンテキストの計算戦略の最適化によるところが大きいと考えられる。結果として、システムのパフォーマンスを大きく損なうことなく UC 場所情報システムが実現されており、実用に十分なパフォーマンスを得られていることが確認できた。

なお、利用者からみたパフォーマンスについても十分な利用感を実現するのに十分な性能を実現できていることはアンケート調査により確認できている。その具体的な内容は、第 6.2 節で示すとおりである。

6.4 電力消費

本研究では、アプリケーションシステムから直接的な指示を行わずに、UC フレームワークにおける電源管理フレームワークを用いて自動的に低消費電力化処理を行っている。この方式と UC フレームワークの有効性を検証するため、実際にユビキタスガイドによるツアーを行った際の消費電力量を計測し、評価した。

6.4.1 実験内容

電力消費の評価のための実験として、UC 場所情報システムの端末を実際に被験者に使ってもらい、その間の消費電力量を計測し、他の手法を用いた場合の消費電力量と比較することで評価を行った。

具体的な手順としては、被験者に特定のガイドルートにしたがってユビキタス場所情報サービスを歩いて使ってもらい、その際の消費電力量を計測した。今回設定したルートは銀座内を約 30 分ほど歩くルートとなっており、具体的な順路は以下に示すとおりとなっており、その位置関係は図 6.12 に示すとおりである。

1. 日本のフルーツパーラ発祥の地「銀座千疋屋」
2. 歩行者天国発祥の地「銀座 4 丁目交差点」
3. あんみつ発祥の店「銀座若松 (銀座コア 1 階)」
4. 木村屋総本店
5. ポークカツレツ発祥の店「煉瓦亭」

本実験において、各利用者はユビキタス場所情報端末を持って目的地を徒歩で順に回っていく。各チェックポイントにおいては、端末が近くに配置されたアクティブタグの電波を受信することで場所認識を行い、場所に応じたコンテンツのプッシュ通知を行う。また、利用者はプッシュ通知される情報の閲覧以外の操作を行うことも許容される。たとえば、順路にしたがって進む際にユーザは地図画面の呼び出しを行い、現在位置を確認することも可能である。

ここまでで説明した実験手順を、同一のシステムに別々の省電力メカニズムを搭載したユビキタス場所情報端末を持って体験してもらい、各メカニズム間の消費電力量を比較した。具体的には以下の省電力メカニズムを実装し、比較を行った。



図 6.12: 電力消費量評価実験のためのガイドルート

提案手法

コンテキスト情報に基づき省電力制御フレームワークによる自動的な省電力制御を実現する、本研究における提案手法。

Android 端末を模した省電力手法

Android 端末に搭載されているものと同様の省電力手法を実装したもの。具体的な想定としては、以下のような省電力メカニズムを想定している。

低次コンテキストのみを用いる省電力手法

2.2.1 節で紹介した Nishihara らの提案 [34] に基づく、低次コンテキストのみを用いる省電力メカニズムを実装したもの。

何も省電力手法を使わない

OS・デバイスドライバ内部での省電力制御処理のみを行い、それ以上の省電力メカニズムを実装しないもの。

6.4.2 利用したコンテキスト

本実験における提案手法で用いる具体的なコンテキストとして、以下を定義し利用した。

watchingScreen

ユーザが端末の画面を中止している状態

inDarkness

夜の屋外など、暗いところで端末を利用している状態

underLight

室内灯のもとで端末を利用している状態

outdoor

ユーザが屋外に出ている状態

inGinzaCore

銀座コア内で端末を利用している状態

inGinzaMitsukoshi

銀座三越内で端末を利用している状態

inKimuraya

木村屋総本店内で端末を利用している状態

onMainStreet

アクティブタグの設置されているエリア (銀座実験エリア内の一部) で端末を使用している状態

inGinza

銀座エリア内で端末を使用している状態

running

利用者が走っている状態

walking

利用者が歩いている状態

stopped

利用者が停止している状態

keepingDirection

利用者が同じ方向に向かって停止しているか移動している状態

lookingAround

利用者がある地点にとどまって、あたりを見回している状態

6.4.3 電力制御ルール

前節で示したコンテキストを利用し、本実験では以下の3つの電力制御対象に対してそれぞれ電力制御ルールを定義した。

- LCD 画面の輝度 (0~100%)
- WiFi スキャンの頻度 (ミリ秒)
- RFID の受信モード (0: 受信しない, 1: 受信する)

電力制御ルールは一つの XML ファイルで定義されるが、そこから各電力制御対象に対して定義されるルール部分をそれぞれ抜き出したものがそれぞれ図 6.13・図

6.14・図 6.15 である。それぞれによって定義される電力制御メカニズムは以下のとおりとなっている。

- LCD 画面の輝度制御においては、ユーザが画面を見ていない時はバックライトを完全に切り、輝度を 0% とすることで省電力化を図る。それ以外の場合には、環境の明るさに応じた輝度の設定を行うことで、余分な電力の消費を抑える。
- WiFi のスキャンは、本場所情報システムにおいて測位のために用いられるものである。このため、正確な位置情報が必要なときにはスキャン頻度を高く、そうでない場合にはスキャン頻度を抑えることで省電力化を実現できる。ユーザが一箇所に立ち止まってあたりを見回している場合、あるいはサービスエリア外に出てしまっている時には、利用者は迷っていると判断されることから、図 6.14 による定義においてはスキャン感覚を 0 ミリ秒とし、可能な限り正確かつリアルタイム地図上に位置情報をプロットする。そのほか、ユーザの移動速度や状況、どのような規模の建物内にいるか屋外にいるかによって、サービスの実現に必要な位置情報精度は異なるので、それに応じてスキャン頻度を適切に設定する。
- RFID ビーコン (Dice) からの場所情報の受信については、RFID ビーコンの設置されていないエリアにおいては受信待ちによる電力消費を抑えるため、受信をオフとする。

6.4.4 実験結果

本実験の結果、各手法の適用により実際に得られた消費電力量の削減率を図 6.16 に示す。結果に示されている通り、提案手法による WiFi の消費電力量の削減率が他の手法と比べて大きくなっている。WiFi のスキャン頻度の制御においては、高次コンテキストを利用することでスキャン頻度をきめ細かに制御することが出来たのがその結果であると考えられる。全体としては、提案手法により Android を模した省電力手法に対し 20.7%、低次コンテキストのみを用いる省電力手法に対して 14.0% の省電力化効率の向上を実現できた。

```
<rule target="lcd.brightness"><!-- LCD 輝度 -->
  <match>
    <condition>!context.watchingScreen</condition>
    <setval unit="percent">0</setval></match>
  <match>
    <condition>context.inDarkness</condition>
    <setval unit="percent">10</setval></match>
  <match>
    <condition>context.underLight</condition>
    <setval unit="percent">60</setval></match>
  <match>
    <condition>>true</condition>
    <setval unit="percent">100</setval></match>
</rule>
```

図 6.13: LCD 輝度制御に関する電力制御ルール

```

<rule target="wifi.scancycle"><!-- WiFi スキャン周期 -->
  <match>
    <condition><![CDATA[(context.stopped && !context.searching)
      || !context.inGinza]]></condition>
    <setval unit="msec">0</setval></match>
  <match>
    <condition><![CDATA[context.running
      && context.watchingScreen]]></condition>
    <setval unit="msec">1000</setval></match>
  <match>
    <condition><![CDATA[context.walking
      && context.watchingScreen]]></condition>
    <setval unit="msec">5000</setval></match>
  <match>
    <condition>!context.outdoor</condition>
    <setval unit="msec">5000</setval></match>
  <match>
    <condition>context.inGinzaCore</condition>
    <setval unit="msec">20000</setval></match>
  <match>
    <condition >context.inGinzaMitsukoshi</condition>
    <setval unit="msec">10000</setval></match>
  <match>
    <condition>context.inKimuraya</condition>
    <setval unit="msec">5000</setval></match>
  <match>
    <condition><![CDATA[context.walking
      && context.watchingScreen
      && context.onMainStreet]]></condition>
    <setval unit="msec">20000</setval></match>
  <match>
    <condition>>true</condition>
    <setval unit="msec">1</setval></match>
</rule>

```

図 6.14: WiFi スキャン頻度に関する電力制御ルール

```

<rule target="dice.power"><!-- RFID ビーコン (Dice) 受信 -->
  <match>
    <condition><![CDATA[[!(context.stopped && ! context.searching)
      || !context.inGinza
      || !context.onMainStreet]]></condition>
    <setval unit="binary">0</setval></match>
  <match>
    <condition>context.default</condition>
    <setval unit="binary">1</setval></match>
</rule>
</power>

```

図 6.15: Dice(RFID) 受信に関する電力制御ルール

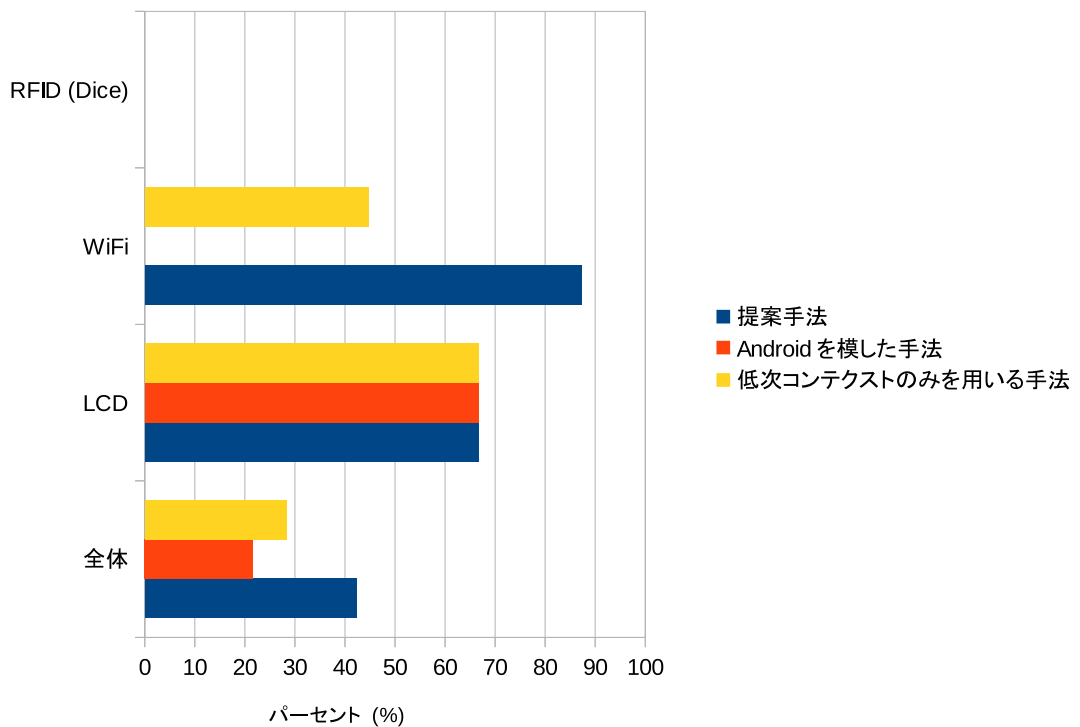


図 6.16: 省電力制御を行わない場合に対する消費電力量の削減率

第 7 章

おわりに

本章では以降、各章の概要をまとめ、それに基づく結論を示した上で、今後の展望を述べた上で本論を終える。

7.1 まとめ

本節では、本論文における各章の概要をまとめる。

7.1.1 関連研究

第 2 章では、ユビキタスコンピューティングのためのソフトウェアアーキテクチャおよび省電力制御技術の分野における関連研究・関連技術をまとめた。

ユビキタスコンピューティングのためのソフトウェアアーキテクチャは多数提案されており、それらの多くはコンテキストの抽象化やそれに基づく処理の実現を容易にするための機能を提供していることが紹介された。しかしながら、これらのいずれの研究においても複数のアプリケーションサービスを同時に動作させるための特別な仕組みは提供されておらず、様々なアプリケーションサービス群によって実現されるユビキタスコンピューティングシステムのプラットフォームとしては不十分であることが明らかとなった。

省電力制御技術の分野においては、コンテキストに基づく省電力化処理に関する研究を中心とした技術をまとめた。コンテキスト情報を用いた省電力化に関する提案は多数みられ、それらが電力効率を大幅に向上させるのに役立つことが示された一方で、既存の提案はいずれも特定の構成からなる機器を想定したものであり、ユビ

キタスコンピューティングにおけるデバイスの多様性や環境中で利用可能なリソースに基づく柔軟な電力制御を実現できる仕組みではないことが分かった。

7.1.2 ユビキタスネットワークアーキテクチャ

第3章では、本研究の前提となるユビキタスコンピューティングを実現するための全体アーキテクチャである、ユビキタスネットワークアーキテクチャについて説明した。ユビキタスネットワークアーキテクチャは、インターネット (IPv6) に基づくアーキテクチャであり、オープンなネットワーク上でスマートハウスなどのアプリケーションサービスが実現されるモデルであることを説明した。

その上で、ユビキタスネットワークアーキテクチャにおける各要素に関する取り組みについて説明した。ユビキタス ID アーキテクチャ 2.0 では、一意識別子である ucode を用いて現実世界におけるモノを一意に特定することと、ucode の割り当てられた事象に対する情報を統合するための仕組みとして ucode 解決サービスが提供されることを説明した。ユビキタス ID アーキテクチャ 2.0 における ucode 解決サービスは、システム間・組織間を横断するハイブリッドなデータベースシステムとして実現されており、ユビキタスコンピューティングに必要なコンテキストの集約・統合のための機能を提供していることが示された。

ユビキタスコンピューティングにおける組込み機器向けのフレームワークとしては uID-CoAP アーキテクチャを提案した。組込み機器を同アーキテクチャに統合するための仕組みとして実装された、組込み機器向け CoAP フレームワークについても説明を行った。また、ユビキタスコンピューティングのための権利および価値情報の流通のためのフレームワークとして、eTNet アーキテクチャを提案した。

7.1.3 UC フレームワーク

第4章では、ユビキタスネットワークアーキテクチャにおけるユーザインタフェース端末上でのサービスの実現を容易にするための仕組みとして、UC (ubiquitous communicator) フレームワークを提案した。UC フレームワークが満たすべき要件として、自発的な動作の実現・高い開発効率の実現・システム全体としての最適動作の3要件を挙げ、これらに基づく UC フレームワークの設計方針について説明した。

UC フレームワークのシステム構成は、コンテキスト推論フレームワーク・アプリケーション間調停フレームワーク・省電力フレームワークの3つからなっており、こ

れらが連携して UC フレームワークとしてのシステム動作を実現することと、それぞれの動作はアプリケーションシステム開発者が定義して与えるポリシー記述により定義されることを説明した。

コンテキスト推論フレームワークは、UC フレームワーク全体における様々なシステムの設定値や状態、センサデータなどの外部環境情報を統合し、アプリケーションにとって意義のある高次コンテキストに変換して、アプリケーションや各システムサービスに配信するための基盤システムである。これを実現するためには、プライマリコンテキストの変化に対し、合成コンテキストの再計算を行った上で各アプリケーションに伝達するための仕組みが必要となる。この再計算に必要なとなるアルゴリズムについても第4章で説明した。

UC フレームワークにおけるアプリケーション間調停フレームワークは、複数のアプリケーションの挙動を調停し、システム全体としての最適な動作を実現する仕組みである。これを実現する上で重要となるのが、ビヘイビアの概念の導入である。アプリケーションの挙動を、ビヘイビアと呼ばれるメタ情報を添付してアプリケーション間調停フレームワークに渡すことで、同フレームワークがアプリケーション間調停ポリシー記述に基づき最適な制御を実現することを示した。これにより、アプリケーション間での調停が必要な挙動を、アプリケーション間のモジュール独立性を維持したまま実現でき、高い開発効率とシステム全体としての最適さという2つの相反する要求を両立させることができる。

省電力制御フレームワークでは、コンテキスト推論フレームワークから得られるコンテキストに基づき、電源管理ポリシー記述により与えられた規則に基づく自動的な省電力制御が実現される。その一方で、アプリケーションの状態とは無関係にコンテキストに基づいた電源管理を行ってしまうと、不適切な省電力化処理が実行されてしまう可能性があることから、アプリケーションから省電力制御に関する制限をかけるための API を提供することを示した。

7.1.4 システム実装

本研究では UC フレームワークのモデルに基づき、「UC 場所情報システム」と呼ばれる観光ガイドアプリケーションまでを含めたトータルなユビキタス情報システムを実際に構築している。第5章では、この UC 場所情報システムに関する実装の詳細について説明した。

同システムにおける特に重要なコンポーネントが UC フレームワークであり、システム実装上は UC フレームワークはコンテキストマネージャ・アプリケーションマネージャの2つによって構成されている。コンテキストマネージャは前述のコンテキスト推論フレームワークに相当する機能を実現するが、効率的な処理のためにあらかじめ評価順序の決定を行っていることを示した。これによりコンテキストの再計算処理が、コンテキスト全体の数に対して高々線形時間で完了することとなる。アプリケーションマネージャはいわゆるウィンドウマネージャに相当する機能であり、アプリケーション間調停フレームワークの考え方に基づくウィンドウ表示のアプリケーション間調停機能を持っていることを説明した。

7.1.5 評価

第6章では、UC フレームワークの方式および実装に関する評価を、開発効率・最適動作の実現性・パフォーマンス・消費電力の4つの観点から行った。

開発効率については、UC 場所情報システムを少ないコード量と少ない定義で実現できているほかその記述自体も分かりやすいものとなっており、十分に高い開発効率を実現できることを確認した。さらに、システム動作やハードウェア仕様変更等にも容易に対応できることが確認され、柔軟性の高いシステムとなっていることも確認できた。さらに、本フレームワーク以前に利用されていた UC12 システムのシステムモデルとの比較を行い、システム全体に対する動作記述を行う UC12 のシステムモデルよりも、提案モデルのほうが開発効率・柔軟性の観点から優れていることを確認した。

最適動作の実現性については、ユーザから見た際の動作の適切性に関する評価という形でアンケートによる評価を行った。結果、いずれの項目においても本提案によるシステムが UC12 システムを超えていることが確認でき、本フレームワークによりシステム全体としての最適動作が実現されることを確認した。

パフォーマンスの評価は、実際の処理に対する時間を実機上で計測することで行った。結果、現在位置の変化に対するフレームワーク内の処理はいずれの場合にも 100 ミリ秒以内に完了しているほか、アプリケーション内での処理時間全体に対して十分に小さい時間割合となっており、本フレームワークの導入により大幅に処理時間が増加しないことを確認した。実際にシステムを動作した際にも利用感的な問題がないことを確認できており、十分に実用的な速度を実現していることを確認した。

消費電力に対する評価は、既存の手法による省電力化手法と Android における省電力化手法における消費電力量の実測値との比較を行うことで行った。結果として、何も省電力化処理を行わない場合に対する省電力化の割合として、Android を模した省電力化手法に対して 20.7%、低次コンテキストのみを用いる省電力化手法に対して 14.0%の省電力化効率の改善が行われることが確認できた。

7.2 結論

本研究では、ユビキタスコンピューティングサービスを提供する端末インタフェース端末向けのソフトウェアアーキテクチャ「UC (ubiquitous communicator) フレームワーク」を提案した。

UC フレームワークは、ユビキタスコンピューティングにおけるユーザインタフェース端末上でのサービスの実現を容易にすることを目的に設計されている。UC フレームワークは、コンテキスト推論フレームワーク、アプリケーション間調停フレームワーク・省電力制御フレームワークの 3 つのフレームワークによって構成され、コンテキストの推論を行うためのルール定義と、それに基づくシステム動作を決定するためのルール群によりその動作を柔軟に変更できる点が特徴である。

UC フレームワークに基づく具体的な実装として、本研究では実用的なユビキタスコンピューティングシステムである UC 場所情報システムを組み込みリアルタイム OS である T-Kernel 上に構築した。UC フレームワークの機能はコンテキストマネージャ・アプリケーションマネージャとして実現されており、D-Bus による通信インタフェースのもとで他のアプリケーションやシステムコンポーネントと連携して動作するように作られている。さらに、UC フレームワークの機能を容易にするための仕組みとして、UC アプリケーションライブラリと呼ばれるクラスライブラリを提供し、よりアプリケーションから使い勝手の良い API を提供した。この UC 場所情報システムはすでに実運用環境への導入が行われており、すでに上野動物園や浜離宮恩賜庭園等におけるユビキタスガイドシステムとして貸し出し運用が行われている。

UC フレームワークの方式および実装に関する評価は、開発効率・最適動作の実現性・パフォーマンス・消費電力の 4 つの観点から行った。結果、すべての観点から UC フレームワークに基づくシステムがユビキタスコンピューティングにおけるサービス構築のためのプラットフォームとして有用であることが示された。

参考文献

- [1] Chulbum Ahn and Yunmook Nah. Design of location-based web service framework for context-aware applications in ubiquitous environments. In *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*, pages 426–433, 2010.
- [2] A Arsanjani, G Booch, T Boubez, P Brown, D Chappell, J deVadoss, T Erl, N Josuttis, D Krafzig, M Little, et al. The SOA manifesto. *SOA Manifesto, Oct*, 2009.
- [3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, October 2010.
- [4] Guruduth Banavar and Abraham Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. *Commun. ACM*, 45(12):92–96, December 2002.
- [5] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. A mobile computing middleware for location- and context-aware internet data services. *ACM Trans. Internet Technol.*, 6(4):356–380, November 2006.
- [6] Masahiro Bessho, Shinsuke Kobayashi, Noboru Koshizuka, and Ken Sakamura. A space-identifying ubiquitous infrastructure and its application for tour-guiding service. In *Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08*, pages 1616–1621, New York, NY, USA, 2008. ACM.
- [7] Claudio Bettini, Dario Maggiorini, and Daniele Riboni. Distributed context monitoring for the adaptation of continuous services. *World Wide Web*, 10(4):503–528, 2007.

- [8] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language. *World Wide Web J.*, 2(4):29–66, November 1997.
- [9] E. Callaway, P. Gorday, L. Hester, J.A. Gutierrez, M. Naeve, B. Heile, and V. Bahl. Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks. *Communications Magazine, IEEE*, 40(8):70–77, 2002.
- [10] Renato Cerqueira, Christopher K Hess, Manuel Román, and Roy H Campbell. Gaia: A development infrastructure for active spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, volume 2, page 11, 2001.
- [11] Sravanthi Chalasani and James M Conrad. A survey of energy harvesting sources for embedded systems. In *Southeastcon, 2008. IEEE*, pages 442–447. IEEE, 2008.
- [12] M Chan, C Hariton, P Ringard, and E Campo. Smart house automation system for the elderly and the disabled. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 2, pages 1586–1589. IEEE, 1995.
- [13] A. Cupper, G. Treu, and C. Linnhoff-Popien. TraX: a device-centric middleware framework for location-based services. *Communications Magazine, IEEE*, 44(9):114–120, 2006.
- [14] Angela B. Dalton and Carla S. Ellis. Sensing user intention and context for energy management. In *Proceedings of the 9th conference on Hot Topics in Operating Systems*, volume 9, pages 26–26, 2003.
- [15] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks, POLICY '01*, pages 18–38, London, UK, UK, 2001. Springer-Verlag.

- [16] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.
- [17] C. Harris and V. Cahill. Exploiting user behaviour for context-aware power management. In *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, volume 4, pages 122 – 130 Vol. 4, aug. 2005.
- [18] Colin Harris and Vinny Cahill. An empirical study of the potential for context-aware power management. In *UbiComp 2007: Ubiquitous Computing*, pages 235–252. Springer, 2007.
- [19] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: a programmable pervasive space. *Computer*, 38(3):50–60, 2005.
- [20] Fritz Hohl, Uwe Kubach, Alexander Leonhardi, Kurt Rothermel, and Markus Schwehm. Next century challenges: Nexus—an open global infrastructure for spatial-aware applications. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '99*, pages 249–255, New York, NY, USA, 1999. ACM.
- [21] Chi-Hong Hwang and A.C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on*, pages 28 –32, nov 1997.
- [22] ITU-T. Multimedia information access triggered by tag-based identification, June 2012.
- [23] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.
- [24] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. Seemon: Scalable and energy-efficient

- context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 267–280, New York, NY, USA, 2008. ACM.
- [25] Mikkel Baun Kjærgaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjær. Entracked: Energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 221–234, New York, NY, USA, 2009. ACM.
- [26] M. Kovatsch, S. Duquennoy, and A. Dunkels. A low-power CoAp for Contiki. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 855–860, 2011.
- [27] Axel Küpper and Georg Treu. Efficient proximity and separation detection among mobile targets for supporting location-based community services. *SIG-MOBILE Mob. Comput. Commun. Rev.*, 10(3):1–12, July 2006.
- [28] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. TinyOS: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.
- [29] Marwa Mabrouk, T Bychowski, H Niedzwiadek, Y Bishr, JF Gaillet, N Crisp, W Wilbrink, M Horhammer, G Roy, and S Margoulis. OpenGIS location services (OpenLS): core services. *OGC Implementation Specification*, 5:016, 2005.
- [30] Delfina Malandrino, Francesca Mazzoni, Daniele Riboni, Claudio Bettini, Michele Colajanni, and Vittorio Scarano. Mimosa: context-aware adaptation for ubiquitous web access. *Personal and Ubiquitous Computing*, 14(4):301–320, 2010.
- [31] Microsoft Corporation. Windows sensor and location platform. <http://msdn.microsoft.com/en-us/library/windows/hardware/gg463473.aspx>.

- [32] Geoff Mulligan. The 6LoWPAN architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 78–82, New York, NY, USA, 2007. ACM.
- [33] Kazuya Murao, Tsutomu Terada, Yoshinari Takegawa, and Shojiro Nishio. A context-aware system that changes sensor combinations considering energy consumption. In *Pervasive Computing*, pages 197–212. Springer, 2008.
- [34] K. Nishihara, K. Ishizaka, and J. Sakai. Power saving in mobile devices using context-aware resource control. In *Networking and Computing (ICNC), 2010 First International Conference on*, pages 220–226, 2010.
- [35] Lawrence Page. Method for node ranking in a linked database, September 4 2001. US Patent 6,285,999.
- [36] Havoc Pennington. D-Bus specification revision 0.23. <http://dbus.freedesktop.org/doc/dbus-specification.html>, 2014.
- [37] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM.
- [38] Weijun Qin, Yuanchun Shi, and Yue Suo. Ontology-based context-aware middleware for smart spaces. *Tsinghua Science & Technology*, 12(6):707–713, 2007.
- [39] Qt Project. Qt framework documentation. <http://qt-project.org/doc/qt/index.html>.
- [40] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. MiddleWhere: A middleware for location awareness in ubiquitous computing applications. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '04, pages 397–416, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

- [41] Manuel Roman. *An application framework for active space applications*. PhD thesis, University of Illinois, 2003.
- [42] Manuel Roman and Roy H. Campbell. A model for ubiquitous applications. Technical report, Champaign, IL, USA, 2001.
- [43] Manuel Roman, Christopher K. Hess, Anand Ranganathan, Pradeep Madhavarapu, Bhaskar Borthakur, Prashant Viswanathan, Renato Cerqueira, Roy H. Campbell, and M. D. Mickunas. Gaiaos: An infrastructure for active spaces. Technical report, Champaign, IL, USA, 2001.
- [44] A. Roy, S.K. Das Bhaumik, A. Bhattacharya, K. Basu, D.J. Cook, and S.K. Das. Location aware resource management in smart homes. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 481 – 488, march 2003.
- [45] K. Sakamura and N. Koshizuka. The eTRON wide-area distributed-system architecture for e-commerce. *Micro, IEEE*, 21(6):7–12, 2001.
- [46] Ken Sakamura. The objectives of the TRON project. Technical Report 87-29, Department of Information Science, Faculty of Science, University of Tokyo, 1987. Proceeding of the Third TRON Project Symposium, Nov. 13, 1987, Tokyo.
- [47] Zach Shelby, Klaus Hartke, and Carsten Bormann. Constrained application protocol (CoAP). 2013.
- [48] T-Engine Forum. T-Kernel 2.0. <http://www.t-engine.org/>.
- [49] Tokyo Ward and Ministry of Land, Infrastructure and Transport, Japan. Tokyo ubiquitous technology project in Ginza. http://www.tokyo-ubinavi.jp/index_en.html.
- [50] UC Technology Corporation. Kokosil Ginza – Ubiquitous Spatial Information System. <http://home.ginza.kokosil.net/en/>.

- [51] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, 1997.
- [52] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.
- [53] Qing Wu, M. Pedram, and Xunwei Wu. Clock-gating and its application to low power design of sequential circuits. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 47(3):415–420, mar 2000.
- [54] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382, oct 1995.
- [55] YRP Ubiquitous Networking Laboratory. *Introduction: Ubiquitous Security Technology and Related Tools*. 2006.
- [56] 別所 正博. ユビキタス空間識別基盤に関する研究. PhD thesis, 東京大学大学院学際情報学府, 2008.
- [57] 矢代 武嗣, 別所 正博, 小林 真輔, 越塚 登, 坂村 健. Qt over T-Kernel Extension 2.0. In *第 3 回トロン技術研究会 予稿集*. T-Engine フォーラム, 2010.

付録 A

UCフレームワークのAPI仕様

本付録では、UCフレームワークの外部API仕様について示す。本節で示されるAPIはいずれもD-Busサービスとして定義される。通信仕様については理解を容易にするため、C++言語の関数プロトタイプ形式で示す。引数・戻り値等の送受信のシリアライズ方式についてはQtDBusの仕様に基づくものとする。

A.1 コンテキストマネージャAPI仕様

A.1.1 エンドポイント

コンテキストマネージャが提供するサービスのエンドポイントは以下のとおりである。

- D-Bus Service Name : `jp.ubin.ucsoft.ContextManager`
- D-Bus Object Path: `/Context`

コンテキストマネージャによって公開されるD-Busオブジェクトは`"/Context"`(コンテキストオブジェクト)のみであり、サービスと一対一対応している。したがって、サービスとオブジェクトを同一視して問題ない。

A.1.2 メソッド

`"/Context"`(コンテキストオブジェクト)に対する、D-Bus上のメソッド定義は以下のとおりである。

```
ID registerClient(ID pid);
ER unregisterClient(ID id);
```

コンテキストマネージャのクライアントとして、自プロセスを登録・登録解除する。`registerClient` の引数としては、コンテキストマネージャへの接続 ID が返される。

以降に述べる `limitSet`, `getSet`, `getSetDifference` 等の主に集合知に関する操作はほとんどクライアント単位で動作が分離されているため、これらの関数を使うためには `registerClient` を用いて自分を登録し、取得した接続 ID を上記操作関数に渡す必要がある。

```
ER create(QString key, QDBusVariant val, bool persistent);
ER remove(QString key);
```

名称 `key` を持つコンテキストを生成・削除する。`persistent` が真の時、端末の電源が切られた場合も値が保存される。(したがって、リッチなグローバル環境変数のように利用することも可能である。)

この機能はアプリケーション・デーモンからは直接利用しないことが望ましい。代わりに、コンテキストの定義はコンテキストマネージャが参照する記述ファイルにて定義され、コンテキストの生成・削除は同マネージャによって内部的に行われるのが一般的である。(記述ファイルの内容は、`create` の引数をそのまま列挙したものと合成コンテキストのルール記述に一致する。)

```
ER setValue(ID id, QString key, QDBusVariant val);
```

名称 `key` で示されるコンテキストの値を `val` に設定する。`id` には接続 ID を指定する。

コンテキスト `key` が未定義な場合は、エラーコード `E_NOEXS` (オブジェクトが存在しない) が返る。

```
QDBusVariant getValue(int id, QString key);
```

コンテキスト `key` の値を `val` に設定する。コンテキスト `key` が未定義な場合は、`invalid QVariant` が返る。

```
QSet<UcUcode> getSet(ID id, QString key);
```

集合値コンテキスト `key` の値を取得する。

得られる集合は、`limitSet` で指定された普遍集合の範囲内に限定される。

```
void limitSet(ID id, QSet<UcUcode> subset);
```

接続 ID `id` で示されるクライアントが扱う `ucode` の範囲を、`ucodeSet` で示される普遍集合に限定する。

この API 関数を用いて `ucode` 範囲を限定することで、それ以外の `ucode` を `getSet`、`getSetDifference` の対象から除外し、プロセス間通信のオーバーヘッドを減らすことができる。

```
UcUcodeDiff getDifference(int id, QString key);
```

集合値コンテキスト `key` に対して、前回の値取得時との差分 (新たに加わった要素集合と削除された要素集合のペア) を返す。得られる集合は、`limitSet` で指定された普遍集合の範囲内に限定される。

```
ER suspendSensors();
```

```
ER resumeSensors();
```

センサマネージャ(OS 拡張機能) をサスペンド/レジュームする。

A.1.3 シグナル

一方、`"/Context"` (コンテキストオブジェクト) に対するシグナル定義は以下のとおりである。

```
void notifyUpdate(QStringList names);
```

変化のあったコンテキストの名前の一覧をクライアント側に通知する。

A.2 アプリケーションマネージャAPI仕様

A.2.1 エンドポイント

アプリケーションマネージャが提供するサービスのエンドポイントは以下のとおりである。

- D-Bus Service Name : `jp.ubin.ucsoft.ApplicationManager`
- D-Bus Object Path: `/Application`

コンテキストマネージャによって公開される D-Bus オブジェクトは `"/Application"` (アプリケーションオブジェクト) のみであり、サービスと一対一対応している。したがって、サービスとオブジェクトを同一視して問題ない。

A.2.2 メソッド

`"/Application"` (アプリケーションオブジェクト) に対する、D-Bus 上のメソッドの仕様を示す。

```
ER registerProcess(ID pid, QString name);
ER unregisterProcess(ID pid);
```

プロセス ID `pid` を持つアプリケーションプロセスをアプリケーションマネージャに登録・登録解除する。`name` にはアプリケーションのグローバル識別名を与える。(pid は起動順などによって変化する可能性があるが、name は常に不変の参照名となる。)

```
ER activateProcess(QString name);
```

アプリケーション識別名 `name` で示されるプロセスを有効化する。

```
ER registerWindow(ID pid, ID wid);
ER unregisterWindow(ID wid);
```

トップレベルウィンドウをアプリケーションマネージャに登録・登録解除する。

本 API 関数による登録後、wid で示されるウィンドウはアプリケーションマネージャのウィンドウの調停の対象とされるようになり、ウィンドウの調停に関する処理を適切に行わなければならない。

```
void notifyBehavior(ID wid, ID rid, QVariantMap behavior);
```

ビヘイビアの変更 (あるいは変更がないこと) をアプリケーションマネージャに通知する。

behavior が空の場合、ビヘイビアに変更がないという意味に解釈されるものとする。rid はビヘイビアの変更の契機となったビヘイビア通知要求であり、シグナル requestBehavior に対する返答においてはかならずその要求に含まれる rid を用いなければならない。

なお、マネージャからの要求によらず、自アプリケーション内で状況の変化にもとづいて自発的にビヘイビアの変更の通知を行う場合には、rid としては負の値を指定する。rid が正の値であり、requestBehavior で指定された期限を超えて通知が行われた場合には、ビヘイビアの変更によるウィンドウの切り替えは行われないうちに延期されるものとする。

```
ER setProgress(QString name, bool startup, int percent);
```

アプリケーションの起動・終了処理の進捗状況をアプリケーションマネージャに通知する。

アプリケーションマネージャはこの情報を用いてアプリケーションの起動・終了の同期を行うほか、システムの起動状況や終了状況をプログレスバーに表示する。

```
ER shutdown(int code);
```

ユビキタス情報システム全体のシャットダウン処理を実行する。引数として、終了コード code に終了要因をとる。

A.2.3 シグナル

”/Application”(アプリケーションオブジェクト) に対する、D-Bus 上のシグナルの仕様を示す。

```
void requestBehavior(ID rid, int64_t deadline);
```

ビヘイビアの変更 (あるいは変更がないこと) の通知を要求する。
この通知を受け取ったトップレベルウィンドウは、可能なかぎり即座にマネージャへ `notifyBehavior` を発行し、新しいビヘイビアまたは変更がないことを通知しなければならない。最大の待ち時間は、`deadline` (ミリ秒) で示され、アプリケーションはその期限内に応答を返さなければならない。ただし、`deadline` の値に関係なく、アプリケーションは可能な限り即座にアプリケーションマネージャに対し `notifyBehavior` によるビヘイビアの通知を送信することが求められる。

```
void activateWindow(ID rid, QVariantMap align);
```

トップレベルウィンドウに対し、`align` で示される表示設定に基づくウィンドウの制御を要求する。

```
ER terminateProcess(ID pid, int code);
```

プロセス ID `pid` で示されるアプリケーションシステムおよびシステムコンポーネントのシャットダウンを要求する。

付録 B

UCアプリケーションライブラリのAPI仕様

本付録では、UCアプリケーションライブラリのインタフェース定義を示す。

B.1 UcApplication クラス

UcApplication クラスは、UC フレームワーク上のアプリケーションの基底クラスである。Qt で提供される QApplication の代わりに用いることで、UC フレームワーク上のアプリケーションを容易に実現することが可能となる。

```
class UcApplication : public QApplication
{
    Q_OBJECT
public:
    UcApplication(const QString& appName, int &argc, char** argv);
    UcApplication(const QString& appName, int &argc, char** argv,
                  bool GUIenabled);
    UcApplication(const QString& appName, int &argc, char** argv,
                  QApplication::Type type);
    virtual ~UcApplication();

    QVariant getContext(const QString& contextName);
    void setContext(const QString& contextName, QVariant value);
    const UcUcodeSet& getPlace() const;
    const UcLocation& getLocation() const;
    const UcDirection& getDirection() const;
    const QString& getLanguage() const;
    const QString& getCourse() const;

    const QSet<QString>& getInterests() const;
    void setInterests(const QSet<QString>& contexts);
    void addInterest(const QString& context);
    void removeInterest(const QString& context);
    bool hasInterest(const QString& context) const;
```

```

    int registerWindow(UcMainWindow* window);
    int unregisterWindow(UcMainWindow* window);
    void notifyBehavior(UcMainWindow* window, int rid);

    void setProgress(bool startup, int percent);

    template <class T> T* getApplicationInterface();

    const QString& getConfigPath() const;

signals:
    void enteredPlace(const UcUcodeSet& ucodeSet);
    void leftPlace(const UcUcodeSet& ucodeSet);
    void placeUpdated(const UcUcodeSet& ucodeSet);
    void markerUcodeRead(const UcUcode& ucode);
    void locationUpdated(UcLocation location);
    void directionUpdated(UcDirection direction);
    void languageUpdated(const QString& language);
    void courseUpdated(const QString& course);
    void contextUpdated(const QString& contextName);

protected slots:
    void onContextUpdateNotification(const QStringList& names);
    void onTerminateProcessRequest(int pid, int code);
    void onActivateWindow(int rid, const QVariantMap& align);
    void onServiceOwnerChanged(const QString&, const QString&, const QString&);
    void setTranslation(const QString& lang);

private:
    Q_DISABLE_COPY(UcApplication)
    Q_DECLARE_PRIVATE(UcApplication)
    QScopedPointer<UcApplicationPrivate> d_ptr;
};

#define ucApp (static_cast<UcApplication*>(QCoreApplication::instance()))

```

B.2 UcDirection クラス

UcDirection クラスは、方向を表現するためのクラスである。

```

class UcDirection
{
public:
    explicit UcDirection(double yaw = 0, double pitch = 0, double roll = 0)
        : m_yaw(yaw), m_pitch(pitch), m_roll(roll) {}

    friend QDBusArgument& operator<<(QDBusArgument& argument,
                                    const UcDirection& dir);
    friend const QDBusArgument& operator>>(const QDBusArgument& argument,
                                           UcDirection& dir);

```

```

double yaw() const { return m_yaw; }
double pitch() const { return m_pitch; }
double roll() const { return m_roll; }
void setYaw(double yaw) { m_yaw = yaw; }
void setPitch(double pitch) { m_pitch = pitch; }
void setRoll(double roll) { m_roll = roll; }

bool operator==(const UcDirection& other) const {
    return m_yaw == other.m_yaw
        && m_pitch == other.m_pitch
        && m_roll == other.m_roll;
}
bool operator!=(const UcDirection& other) const {
    return !operator==(other);
}

private:
    double m_yaw, m_pitch, m_roll;
};

```

B.3 UcLocation クラス

UcDirection クラスは、位置座標を表現するためのクラスである。

```

class UcLocation
{
public:
    explicit UcLocation(double lat = 0, double lng = 0,
                        double err = 0, double floor = 0)
        : m_latitude(lat), m_longitude(lng), m_error(err), m_floor(floor) {}

    friend QDBusArgument& operator<<(QDBusArgument& argument,
                                    const UcLocation& location);
    friend const QDBusArgument& operator>>(const QDBusArgument& argument,
                                           UcLocation& location);

    double latitude() const { return m_latitude; }
    double longitude() const { return m_longitude; }
    double error() const { return m_error; }
    double floor() const { return m_floor; }
    void setLatitude(double latitude) { m_latitude = latitude; }
    void setLongitude(double longitude) { m_longitude = longitude; }
    void setError(double error) { m_error = error; }
    void setFloor(double floor) { m_floor = floor; }

    bool operator==(const UcLocation& other) const {
        return m_latitude == other.m_latitude
            && m_longitude == other.m_longitude
            && m_error == other.m_error
            && m_floor == other.m_floor;
    }
};

```

```

    }
    bool operator!=(const UcLocation& other) const {
        return !operator==(other);
    }

private:
    double m_latitude, m_longitude;
    double m_error;
    double m_floor;
};

```

B.4 UcMainWindow クラス

UcMainWindow クラスは、ウィンドウ表示調停のためのビヘイビア処理機能を内包した、UC アプリケーションにおけるメインウィンドウのためのクラスである。

```

class UcMainWindow : public QWidget
{
    Q_OBJECT

public:
    explicit UcMainWindow(QWidget *parent = 0, Qt::WindowFlags flags = 0);
    ~UcMainWindow();

    QVariantMap currentBehavior(bool recalculate);

protected:
    void setVisible(bool visible);

    virtual bool recalcbBehavior();
    QVariantMap& behavior();

private:
    Q_DISABLE_COPY(UcMainWindow)
    Q_DECLARE_PRIVATE(UcMainWindow)
    QScopedPointer<UcMainWindowPrivate> d_ptr;
};

```

B.5 UcUcode クラス

UcUcode クラスは、unicode を表現するためのクラスである。

```

class UcUcode
{
public:
    explicit UcUcode(quint64 hi = 0, quint64 lo = 0);
    explicit UcUcode(const QString& unicode);
};

```

```

friend QDBusArgument& operator<<(QDBusArgument& argument,
                                const UcUcode& ucode);
friend const QDBusArgument& operator>>(const QDBusArgument& argument,
                                       UcUcode& ucode);

quint64 hi() const { return m_hi; }
quint64 lo() const { return m_lo; }

QString toString() const;

static UcUcode fromByteArray(const quint8 data[16]);
void toByteArray(quint8 data[16]);

bool operator==(const UcUcode& other) const {
    return m_lo == other.m_lo && m_hi == other.m_hi;
}

bool operator<(const UcUcode& other) const {
    return m_hi < other.m_hi || (m_lo < other.m_lo && m_hi == other.m_hi);
}

bool operator>(const UcUcode& other) const {
    return m_hi > other.m_hi || (m_lo > other.m_lo && m_hi == other.m_hi);
}

bool operator!=(const UcUcode& other) const {
    return !operator==(other);
}

bool operator>=(const UcUcode& other) const {
    return !operator<(other);
}

bool operator<=(const UcUcode& other) const {
    return !operator>(other);
}

private:
    quint64 m_hi, m_lo;
};

```