コンテクストアウェアなアプリケーション構築のためのフレームワーク

藤波香織[†] 中島達夫[††]

ユビキタスコンピューティング環境に関する研究領域の1つにコンテクストアウェアネス（Context-Awareness：状況依存性）があり，物理空間とサイバー空間に張り巡らされたセンサとアクチュエータの連携による新たな産業やサービスの創出が期待される．しかしながら，このような環境の特徴の1つに超多様性があげられるため，アプリケーション開発にあたってもこれを考慮する必要がある．本論文では，コンテクストに関する情報をその本来の情報であるベースコンテクスト情報とこれを処理するためのメタコンテクスト情報に分離し，メタコンテクスト情報の取扱いを提案するフレームワーク中で論じる．特にメタコンテクスト情報としてコンテクストの時間的な不変性を取り扱い，実装および基礎性能測定を実施し，その可能性と課題について考察したので報告する．

A Framework for Developing Context-aware Applications

Kaori Fujinami[†] and Tatsuo Nakajima[††]

Context-awareness is one of the exciting research topics in a ubiquitous computing environment, which is expected to create some new businesses and/or services in conjunction with sensors and actuators embedded in physical spaces and cyber spaces. However, because ultra-heterogeneity is one of its unique characteristics, developers of such a context-aware application should take into account of this. In this paper, we propose an application framework and discuss a scheme for handling meta-context information, which represents context information itself, in our framework, especially an update rate of context as meta-context information. Also, we report its implementation, basic performance evaluation, and discussions.

1. はじめに

無線通信技術の発達，計算機の小型高性能化にともない，あらゆるものがネットワークに直接的，あるいは間接的に接続され日常生活空間内に自然な形で計算機能を埋め込むことが可能になりつつある．このような計算機環境はユビキタスコンピューティング環境（以下，ユビキタス環境とする）[1)]と呼ばれ，1990年代初頭より研究されてきた．

このような環境下では，空間に張り巡らされたセンサやアクチュエータの連携により，人間の明示的な操作指示なしに状況に依存した空間やデバイス機能の再構成といったことが可能になり，ユビキタス環境は，新たな人間とコンピュータとのかかわり方を示している．

† 早稲田大学大学院理工学研究科情報科学専攻
Graduate School of Science and Engineering, Waseda University
†† 早稲田大学理工学部コンピュータ・ネットワーク工学科
School of Science and Engineering, Waseda University

ユビキタス環境の特徴として，1）機能・性能・プラットフォーム・データ・データ取得方法・ユーザニーズなど，あらゆる多様性を内包していること（超多様性），2）WWW（World Wide Web）に代表されるサイバー空間とセンサやアクチュエータが埋め込まれている物理空間とのシームレスな融合，といったことがあげられる．特に2）については，サイバー空間が持つ物理法則非支配性（サイズ，時間，位置に依存しない性質）と物理空間が持つ物理法則支配性といった双方の特徴を適切に組み合わせることで，物理空間をサイバー空間で拡張したり，その逆を行うことができ，新たなサービスや産業の創出も期待される．

ユビキタス環境に関する研究領域の1つとして，コンテクストアウェアネス（Context-Awareness）がある．これは「状況依存性」を意味し，対象が置かれている状況（コンテクスト）をセンサなどから得られた情報を基に認識し，それを他者に提供したり，それにあわせて，その後の振舞いを変化させるアプリケーション（以下，コンテクストアウェアなアプリケーション，または単にアプリケーションとする）が考えられる[2)]．

本論文では，このようなユビキタス環境下でのアプリケーションフレームワークを提案し，コンテクストに関する情報（以下，コンテクスト情報とする）を本来の情報であるベースコンテクスト情報とこれを処理するためのメタコンテクスト情報に分離し，フレームワーク中でのメタコンテクスト情報の扱いについて論じる☆．ベースコンテクスト情報の扱い，およびメタコンテクスト情報の必要性については様々な研究で発表されているが[2)~8)]，メタコンテクスト情報の具体的な扱いに関してはほとんど言及されておらず，アプリケーションはアドホックに開発されている．メタコンテクスト情報は使用するセンサやコンテクスト抽出アルゴリズムに大きく影響を受けるものであり，変更のたびにアプリケーションの修正を行うことは，アプリケーション開発者，さらには一般消費者にとっても望ましくない．このため，メタコンテクスト情報をフレームワーク内にて扱うことで，Precision や Accuracy といったコンテクストに関する品質（Quality of Context：QoC）を明示的に扱うことが可能となり，ユビキタス環境の超多様性に対応することが可能となる．本論文では，QoC のうち，センサのデータ更新間隔と関連が深い，時間的な不変性を表す指標（TimePrecision）を扱う．提案するフレームワークにより，アプリケーション開発者が更新間隔の多様性を意識することなく，アプリケーションロジックの設計および実装に専念できることが期待される．

以下では，まず 2 章でフレームワークの基本設計について説明したのち，3 章にて例題シナリオをあげ，センサのデータ更新間隔に関わる問題提起とアプローチの概略説明を行い，4 章で段階的な抽象化コンテクストの抽出方法についてメタコンテクスト情報の扱いを交えて述べる．そして，5 章でそれを踏まえた実装と評価，6 章で考察と課題について述べたのち，7 章で関連研究，最後に 8 章で結論を述べる．

2. 設　　計

以下では，要求条件定義の後，実現方針および構成要素の説明を行う．

2.1 要求条件

次の要求条件をフレームワークは満たす必要がある．

(1) 同一アプリケーションでも様々なデバイスやサービスで構成されるため，それらに対しアプリケーションはポータブルであること．

(2) アプリケーション開発者の稼動を削減するため，センサデータやコンテクスト情報の品質を隠蔽化すること．

(3) ユーザニーズは多様で変化も早いため，アプリケーションの新規開発が容易であること．

(4) 通信ミドルウェア部分に関しても様々な製品を選択可能であるべきなため，フレームワーク自体がポータブルであること．

(5) 同一空間に複数のアプリケーションが存在し，システムを構成するデバイスやサービス，ならびに流通するデータが膨大なため，システムはスケーラブルであること．

(6) ユーザのプライバシーに直結する情報を扱うため，これを保護すること．

2.2 実現方針

上記の要求条件に対して，(1) および (2) については後述するように，コンテクスト情報をアプリケーションが本来の動作のために利用する情報（ベースコンテクスト情報）とそれを説明する様々な属性情報（メタコンテクスト情報）で構成し，フレームワークはメタコンテクスト情報を用いて実行環境に自己適応できるようにする．(3) については，コンテクストの抽象化度合いを細粒度化し段階的に抽出することで，新たに追加するアプリケーションのために新規にコンテクスト抽出ルールを記述することなく，既存の必要な粒度（抽象度）のものを利用可能とする．(4) については，通信ミドルウェアを抽象化するレイヤを設けることで，アプリケーション部分と分離する．(5) については，抽出に関わる構成要素を分散オブジェクト化し分散環境に配備可能とすることで，スケーラビリティを確保することを基本方針とするが，具体化については今後の課題とする．また，(6) についても今後の課題とする．

2.3 コンテクスト情報モデル

実現方針にて述べたように，コンテクスト情報はベースコンテクスト情報とメタコンテクスト情報の 2 層で表現する．

前者は，アプリケーションが必要とする抽象化度合いを「いつ，どこで，何が，どうである」という情報で表現するもので，主格（Subject）とその主格のある瞬間の状態（State）で表現し，さらに主格は対象（Target），時間（Time），場所（Place）で構成する．

一方後者は，主格および状態の各構成要素が用いる単位やフォーマット，QoC といった属性で構成される．QoC には，時間および場所の分解能（TimeResolution，PlaceResolution），状態の時間および空間的な不変性（TimePrecision，Place-

☆ これらの語の定義については 2.2 節および 2.3 節を参照のこと．

Precision），現実の状態を正しく抽出できる能力（Accuracy）といった属性がある．本論文では，メタコンテクスト情報についてはこの TimePrecision について扱うこととし，3 章にて例題シナリオを用いてこのような QoC の必要性について述べる．

なお，主格はある瞬間に 2 つ以上の状態の中のいずれかをとり，段階的な抽出過程においてより抽象度が低いコンテクスト（以下，下位コンテクストとする）の変化により別の状態を表す条件を満足するようになった場合にはその状態に遷移し，「コンテクストが変化した」，あるいは「コンテクストが遷移した」と呼ぶことにする．そしてこのコンテクストの変化は，より抽象度が高いコンテクスト（以下，上位コンテクストとする）の抽出，またはアクチュエータでの利用のためにイベントとして外部に発行される．

2.4 段階的コンテクスト抽出

段階的なコンテクスト抽出のための構成要素は，**図 1** に示すような，センサ（Sensor）・アクチュエータ（Actuator）・コンテクスト抽出器（Context Event Generator：CEG）の 3 つに大別される．センサはコンテクストの抽出に必要なデータをシステムに提供するものであり，物理空間の物理量を測定するものだけでなく，サイバー空間上の情報提供サービスから情報を取得するものや，計算機やネットワークの状態を監視するものも含む．アクチュエータは，抽出されたコンテクストに応じて物理空間やサイバー空間上のサービスに対して作用する．

CEG は抽象度の低いコンテクスト情報を用いて，より高い抽象度のコンテクストを抽出する処理ノード（以下，単にノードとする）であり，アプリケーション開発者により指定された抽出条件を充足した場合に，コンテクストの変化をイベントとして発行する．図 1 において，実線矢印はこのイベントのフローを表しており，A_2 と A_3 は CEG_5 が抽出したコンテクスト情報を共有するのに対し，A_1 は CEG_6 が CEG_1，CEG_2，および CEG_5 から得られたコンテクスト情報を用いて抽出した，さらに抽象度の高いコンテクスト情報を用いている．このように，段階的に高抽象度のコンテクストを抽出することにより，アクチュエータ側での自身が必要とするコンテクスト情報への加工処理が減少し，容易に新たなアクチュエータを追加しアプリケーションを開発可能となる．また新たに，より高抽象度のコンテクスト情報を必要とするアクチュエータを追加する場合にも，途中段階のコンテクスト抽出には既存の CEG を共有可能となり，新規開発部分を最小限に抑えることができる．CEG はコンテクストの抽出に際して中心的な役割を担うノードであり，詳細は 4 章にて述べる．なお，アプリケーションとは図 1 に示される要素で構成されるシステムを指す．

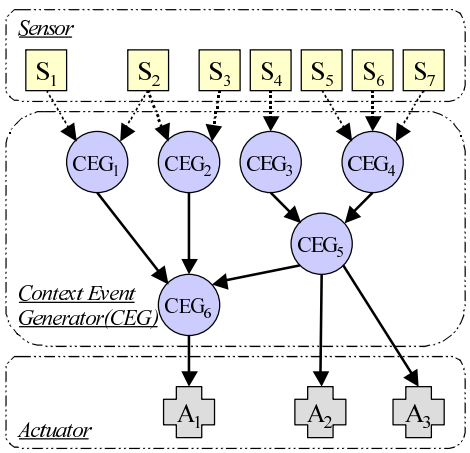

図 1 システム概念図
Fig. 1 Conceptual diagram of proposed system.

2.5 レイヤ構成

CEG は**図 2** に示す 6 つのレイヤで構成され，2.3 節で説明したベースコンテクスト情報とメタコンテクスト情報を利用して，ポータブルなアプリケーションの一部として振る舞う．

アプリケーションロジックレイヤは，コンテクスト抽出のための情報をノードに与えるレイヤであり，ノード機能レイヤが提供するアプリケーションロジック定義 API を利用する．なお，これがアプリケーション開発者の意識するレイヤとなる．

ノード機能レイヤには，4.2 節で説明する CEG の基本的な振舞いが定義されており，上位レイヤで定義されたロジックに従い振る舞うことで，下位コンテクストの変化に応じた新たなコンテクストへの遷移を可能とする．これら 2 つのレイヤは，ベースコンテクスト情報を利用する．また，ノード適応レイヤでは，メタコンテクスト情報を利用してデータのフォーマット変換や，4.3 節で述べるような QoC に合わせた CEG

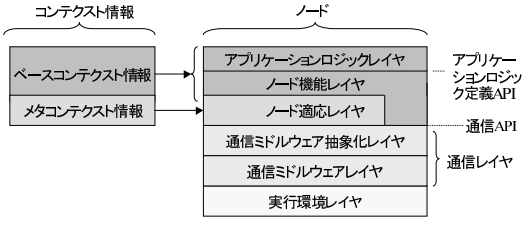

図 2 コンテクスト情報およびノードのレイヤ構成
Fig. 2 Layered architecture of context information and node.

自身の振舞いの最適化を行う．このように，コンテクスト情報にメタ情報を持たせ，それをフレームワーク側で処理することで，アプリケーションは末端のセンサデバイスやコンテクスト抽出過程の他のノードの規格やQoCに影響を受けずに，本来の処理ロジックのみを実行すればよく，ポータブルになる．よって，アプリケーション開発者が処理ロジックの開発に専念することができるようになる．さらに，一般消費者にとっても，ソフトウェアの改変なしに様々なデバイスを利用可能となる．

通信は，各ノードをオブジェクトと見なし，任意の計算機上に配備した分散オブジェクト通信として実現する．通信に関するレイヤでは，オブジェクトの名前管理，遠隔メソッド呼び出し，分散イベント通信といった機能を提供する．このような機構は分散オブジェクト指向のミドルウェアでは一般的な機構であり，Jini[9]やCORBA[10]などでも提供されている．2.2節にて述べたように，フレームワークをミドルウェアに対してポータブルにするために，通信ミドルウェアレイヤと通信ミドルウェア抽象化レイヤに分け，後者のレイヤでは，前者が持っている固有のAPIを隠蔽化して，上位レイヤに対して汎用的な通信APIを提供する．最後に，実行環境レイヤはOperating System（OS）やVirtual Machine（VM）などで構成される．

次章では，データ更新間隔に起因して発生する課題について例をあげて説明し，TimePrecisionについて改めて説明したのち，データ更新間隔の多様性の隠蔽化アプローチの概略を述べる．

3. データ更新間隔の多様性の隠蔽化

3.1 例題シナリオ

以下のシナリオについて考える．

“Fujiさんは大阪で午後2時から会議があるが，現在は東京にいる．そして，大阪では午後からにわか雨があると天気予報ではいっている．Fujiさんは忙しく，そのような天気予報を見る時間がないまま傘を持たずに出かけようとしている．ドアを開けて外に出ようとすると，所持している携帯電話にメールが入り，傘を持っていったほうがよいと教えてくれた．”

このシナリオでは，3種類のセンサ（サイバー空間上の天候情報提供サービス，およびスケジュール情報管理サービスの情報を定期的に監視するプログラム，ドアに取り付けられた加速度センサ），2種類の無線（Radio Frequency：RF）タグ[11]付けされた物理オブジェクト（傘・携帯電話），1種類のアクチュエータ（携帯電話）が登場する．携帯電話については，常時保持されており，Fujiさん自身を表すと考える．なお，天候情報提供サービスおよびスケジュール情報管理サービスについては，天候の変化や行動パターンといった将来起こりうる事象を表していると考え，そこから情報取得するプログラムはセンサの一種と考えた．

3.2 課　　題

この例のように，アプリケーションがなんらかの状態変化に反応した動作をとるべくその末端には必ず離散時間でデータを更新するセンサが存在し，多くは複数のセンサデータ，あるいは下位コンテクストの変化を契機として，上位コンテクストを抽出するが，ここでタイムリーな抽出を実現するためには，アプリケーション開発時にデータ更新間隔を意識する必要がある．つまり，あるコンテクストの変化を検知した際に，過去に受信していた残りのコンテクスト変化の情報を利用すると，誤った情報に基づいて抽出が行われる可能性がある．これは，過去に受信していたコンテクストの抽出に関わった末端のセンサデータの更新間隔が，受信したばかりのものと比較して長い場合に，末端のセンシング対象はすでに変化しており，それにともない別のコンテクストであるべきにもかかわらず検出されないことから発生する．

上述の例題シナリオで，天気情報提供サービスの監視プログラム，および加速度センサのデータ更新間隔がそれぞれ約6時間と0.5秒であるとする．そして最後に天気情報提供サービスに対してデータ取得をした際に「天候は良好」であり，その後5時間が経過している状態で，加速度センサのデータを基に「ドアが開いた」というコンテクストの変化を検出した場合には，アプリケーションは傘の携帯は不要と判断し何も警告はしないが，実際にはこの5時間の間に「天候が悪化する」という状況に変化している場合もある．

したがって，センサが任意の契機でのポーリングをサポートしていれば，一方のコンテクスト変化の検知時に，残りのコンテクストについての最新状態を取得することで，このような不整合が発生する可能性を低減することはできる．しかし，既存研究例では，このような最新状態の取得機構のみをAPIとして提供するのみで，利用はアプリケーション開発者に任されている[3),4)]．たとえば，文献3)では，`updateAndPollWidget`と呼ばれるメソッドをアプリケーション開発者に対して提供している．このメソッドは，引数で指定した情報を基にWidgetと呼ばれるセンサを抽象化したプログラムを検索したのち，最新状態の取得を行ったうえで結果を返すものであるが，Widgetの選択と取得した最新状態に関する処理の実装はアプリケーション開発者

が行う必要がある．このためアプリケーションごとにアーキテクチャについての設計も必要となるうえ，同一事象の測定であっても使用する手法や製品によっても更新間隔が異なることからポータビリティに欠け，アプリケーション開発者がアプリケーションロジックの設計に専念できない．

3.3 課題解決アプローチの概略

図 3 を用いて TimePrecision の概念を改めて説明する．一般に Precision とは，値のばらつきの程度や測定の再現性の程度を表す指標であり[12]，センサ自体の測定能力や，測定する事象の時間的な変化や空間的な偏りなどにも影響を受ける．特に，測定事象の時間的な変化により影響を受ける Precision の属性を TimePrecision と定義する．得られた情報の TimePrecision が高い，すなわち良好ということは，次のデータ更新でも同じ結果を得る可能性が高いということを意味し，更新間隔の間に事象が変化する可能性（時間的に不変の可能性）が低いことを意味する．これは直接センサが監視している個々の事象だけでなく，センサデータから抽出されるコンテクストに関しても同様である．

図 3 では，(a) は間隔 T_H でデータ更新が行われ，(b) は間隔 T_L（ただし，$T_H < T_L$）でデータ更新が行われるとする．(a) のほうが (b) より更新間隔が短いために，1 回の更新あたりの前回更新時と異なる結果を得る可能性について考えた場合，(a) は t_{H1}〜t_{H7} の 7 回に 2 回（t_{H4}，t_{H7}）の検出であるのに対し，(b) は t_{L1}〜t_{L3} の 4 回に 2 回（t_{L3}，t_{L4}）と (a) のほうが低い．このことから，(a) のほうが TimePrecision が高いといえる．

前節においてあげた課題に対するシステマティックな解決策の基本アイデアは，4.3 節で述べるように，コンテクストの変化の検知時にすべてのセンサに対して最新状態を問い合わせるのではなく，TimePrecision が低いもののみ問い合わせるということである．

また，TimePrecision はデータ更新間隔のみならず，事象の変化を検出してからの経過時間によっても決定づけられると考えられる．たとえば (a) について，t_{H4} で S_2 という状態に変化したことを検出し，その後の測定時 t_{H5} より t_{H6} のほうが時間が経過しているために，次の測定で異なる結果（S_3）を得る可能性が高くなっている．このように，前回変化を検出してからの経過時間が長くなるほど変化する可能性が高くなる（再現性が低くなる）と考えることができるため，TimePrecision の一属性と考えられる．

このような，複数の属性から問合せ先を決定する際に利用する総合的な大小値への多次元から一次元への

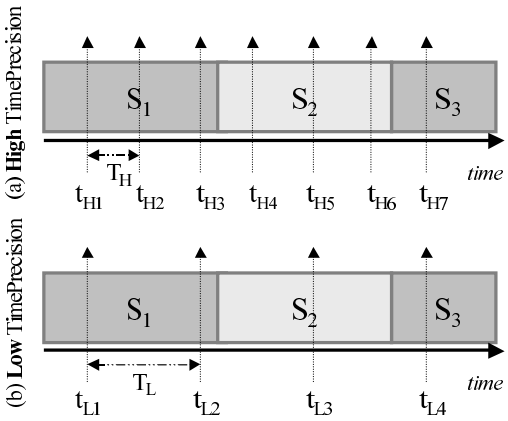

図 3 TimePrecision の概念図
Fig. 3 A conceptual diagram of TimePrecision.

```
抽出ルール ::= ルールID ⇐ 前提条件; 遷移候補
  ルールID          ::= 抽出コンテクストID
  前提条件          ::= 下位コンテクストID (論理演算子 下位コンテクストID)* |
                        前提条件 (論理演算子 前提条件)*
  遷移候補          ::= (遷移コンテクストID)+
  コンテクストID    ::= 主格 ⇐ 状態
  主格              ::= 対象 '/' 時間 '/' 場所
  論理演算子        ::= 論理和 | 論理積
```

図 4 コンテクスト抽出情報定義
Fig. 4 Definition of context extraction information.

写像を，フレームワーク開発者☆は別途定義する．本フレームワークにおいては，5.1.3 項にて述べるような抽象クラスを実装することで，利用側プログラムの変更なく写像を変更することが可能となる．

4. フレームワークの内部動作

4.1 コンテクスト抽出のための情報

コンテクスト抽出に関する情報を EBNF 記法（Extended Backus-Naur Form）[13] により定義すると，**図 4** のようになる．抽出コンテクストごとに抽出ルールが定義され，抽出にあたり充足する必要がある条件を下位コンテクスト ID の論理和（OR）および論理積（AND）の結合で表現した前提条件と，コンテクスト抽出後に遷移する可能性がある 1 つ以上のコンテクスト ID で構成される．なお，コンテクスト ID はベースコンテクスト情報である．これらの情報をアプリケーション開発者が定義し，2.5 節で説明したアプリケーションロジック定義 API を通してフレームワークに反映する．

4.2 CEG 基本動作

図 5 に CEG の内部ブロック図および外部との関係を示し，この図を基に動作を説明する．下位コンテク

☆ フレームワーク開発者とは本フレームワークを開発し，アプリケーション開発者に提供する役割を担う者である．

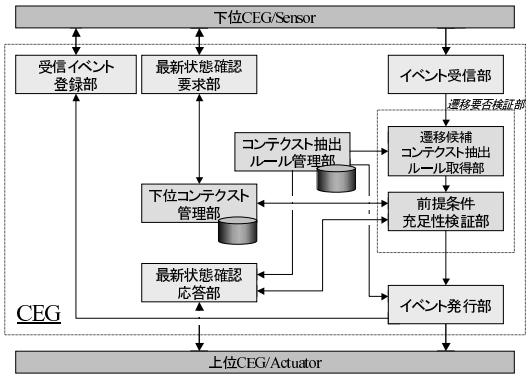

図 5 CEG 内部ブロック図および外部との関係
Fig. 5 Internal block-diagram of the CEG and relationship between external components.

スト変化を通知するイベント受信から新たなコンテクストへの遷移をイベントとして発行するまでには,(1) イベント受信,(2) 遷移候補コンテクスト抽出ルール取得,(3) 前提条件の評価,(4) イベント発行，の 4 段階を経る．以下で段階を追って説明する．

(1) イベント受信部が下位コンテクストの変化をイベントとして受信すると,(2) 遷移候補コンテクスト抽出ルール取得部がコンテクスト抽出ルール管理部より図 4 で表現されるルールを取得し,(3) そこに定義されている前提条件を前提条件充足性検証部が評価する．このとき，前提条件部を構成する他の下位コンテクスト情報については，図の下位コンテクスト管理部を通じて取得する．ここには過去に受信したコンテクスト遷移のイベントが格納されている．図 4 より，前提条件の論理式は論理積と論理和で構成されるので，受信したイベントを葉（Edge）の 1 つとする 2 分木で表すことができる．充足性の検証はこの 2 分木を葉から根（Root）に向かって受信したイベントを葉に持つ部分木から評価することで行われ，途中で不変性が確認されればそこで中断する．これにより 4.3 節で述べる最新状態確認の要否判断対象を狭めることができ，時には 1 回も下位コンテクスト管理部に問い合せる必要すらなくなる．これは，論理式がすべて論理和で構成される場合に当てはまる.(4) 前提条件を充足していれば，新たなコンテクストへの遷移がイベント発行部を通じてイベントとして外部に通知され，充足していなければ現在のコンテクストにとどまる．そして，イベント発行後には次のコンテクスト遷移に備え，遷移候補の抽出ルールを取得し，その前提条件部に記述されている下位コンテクストの変化をイベントとして受信すべく，受信イベント登録部を通じて登録を行う．

以上の 4 段階は 2.5 節で述べたノード機能レイヤにより実現されており，アプリケーションロジックレイヤで定義されている内容に従い，ベースコンテクスト情報を基に下位の抽象度が低いコンテクストの変化を契機として，より抽象度が高いコンテクストの抽出が可能となる．しかし，情報のタイムリーさの欠如により 3.2 節で述べたような不整合が発生することがある．以下で，3.3 節で述べた TimePrecision を用いた解決方法について述べる．

4.3 メタコンテクスト情報の利用

前節の段階（3）で述べたように，前提条件部が 2 つ以上の下位コンテクストで構成された場合には，残りの下位コンテクストを含めた充足性検証が必要になる場合がある．その際には，通信および計算リソースの効率利用を考慮して，必要なもののみその発行元に対して最新状態を確認し，その他は以前受信しているものを下位コンテクスト管理部から利用することとするため，確認要否を判断する機構が必要になる．ここで要否判断は，受信したイベントとその他すでに受信され保存されているものとの間の TimePrecision に関する相対的な関係に支配されると考えた．すなわち，下位コンテクスト管理部に保存されていたものが受信したものと同等かそれより高い TimePrecision を持つ場合には，それは少なくとも受信したものよりは信用できることになる．よってイベントを受信した CEG は，下位コンテクスト管理部に保存されている前提条件を構成する下位コンテクスト情報について，それらを受信した際に内包されてきた TimePrecision と新たに受信したものとを比較し，後者のほうが大きい（高い TimePrecision である）場合には，前者のベースコンテクストの変化の有無を最新状態確認要求部を通じてその抽出元に対して問い合せる．

一方，確認要求をかけられた下位コンテクスト抽出元の CEG（以下，下位 CEG とする）は，最新状態確認応答部を通じて自身の最新状態を評価する．その際には，現在のコンテクスト抽出ルールより前提条件を取得し，同様にこれを評価する．このように，芋づる式にデータ発生源であるセンサに向けてたどり，最新状態を取得し変化の有無を確認する．

また，CEG がイベント発行する際には，イベント受信時に内包されてきた TimePrecision を代表 TimePrecision として，イベントに内包するコンテクスト情報に持たせることで，最新状態の確認が取れている TimePrecision の上限を受信する CEG に伝える．

図 6 に，イベント発生契機とノード間に発生する最新状態確認通信の関係を示す．S はセンサを表し，カッコ内の数字は処理順を表す．細実線と太実線はそ

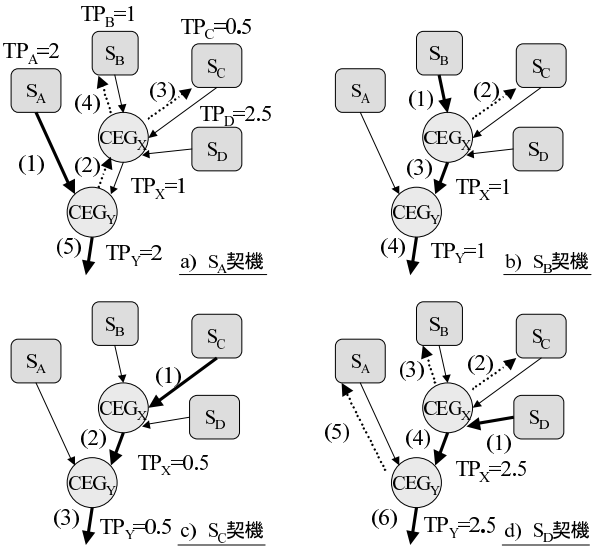

図 6 イベント発生契機と最新状態確認通信の関係
Fig. 6 Relationship between a trigger of event and state updating communication path.

れぞれ到着済のイベントと新たに発生したイベントを，また破線は最新状態確認のための通信を表す．TP は TimePrecision を表し，単に更新間隔の逆数（更新周波数）で表される静的なものとする．したがって，センサの TimePrecision は固定であるため，a）以外は省略した．

ここで a）の場合は，直前に b）の S_B 契機でのイベント発行がなされ，TP_X が S_B の TimePrecision である 1 となっている前提で話を進める．また，CEG_X および CEG_Y が利用するコンテクスト抽出ルールの前提条件部は，すべて論理積（AND）で構成されているとする☆．S_A からのデータ契機でイベントが発生すると，CEG_Y では，TP_X と TP_A を比較し，CEG_X への問合せを決定する．このノードで CEG_Y へのイベント発行に関係した下位コンテクスト情報は S_B，S_C，S_D から得られた 3 つであったが，このうち S_D からのものは TP_D=2.5 と，TP_X より大きいので問合せは行わない．最終的に CEG_Y から発行されるイベントには，受信イベントに内包されていた TP_A の値（2）が内包される．これにより，S_D 以外の最新状態の確認が取れたうえでのイベントであることが保証される．

一方，b）については S_B 契機であり，TP_B より低い TimePrecision を持つ S_C に問い合わせるのみ，c）については最も低い TimePrecision を持つ S_C 契機であるために，一度も問合せは発生しない．逆に d）ではすべてのノードに問い合わせることになる．

☆ 論理和（OR）が存在する場合は，4.2 節で説明したように，そもそも最新状態確認の要否判断をする対象とならない場合があるため，TimePrecision の利用方法を説明するという趣旨を踏まえて論理積に限定した．

5. 実装と評価

本章では現状の実装と評価を行う．なお，本システムでは，実行環境としてネットワーク指向・マルチプラットフォーム指向という点で Java を採用しているが，他言語（C++等）であっても実現可能である．

5.1 実　　装

5.1.1 通信レイヤ

現在の実装では，データおよびコンテクスト遷移イベントの送受信に JavaSpaces[14]，最新状態確認時の通信は RMI[15] を用いている．

JavaSpaces は，Linda システム[16] の Tuple の考え方を拡張し，Java 言語で実装したものであり，space と呼ばれる共有オブジェクトリポジトリに対し，通信主体は `write`（オブジェクトの書込み），`read`（オブジェクトのコピーの読込み）/`take`（オブジェクトの取出し），`notify`（指定オブジェクト書込み通知依頼の登録）の 4 種類の操作を行うことで，疎結合な分散システムを構築することができる．本実装では，JavaSpace をノード間でのイベントの流通機構として用いている．前述のように，ユビキタス環境ではこれらの情報は非同期的に発生し，受信者も動的に変化するために，通信相手を直接意識する必要がない Tuple ベースの通信基盤が適していると考え，その一実装として用いた．そして，JavaSpaces が提供する上記の 4 つ

の固有の API は，通信ミドルウェア抽象化レイヤにて通信 API にマッピングされることで上位レイヤからは隠蔽化される．

一方，下位 CEG またはセンサへの最新状態確認時の通信については，すでに相手を特定可能なので JavaSpaces を介さずに直接通信することとした．これについても，一実装として RMI を用いている．現在は，1 ノードが 1 リモートオブジェクトに対応しており，オブジェクトの名前管理についてはコンテクスト ID をリモートオブジェクト名としている．

5.1.2 アプリケーションロジックレイヤ

図 7 にあげるルールでコンテクストの抽出を行う場合の記述例を，**図 8** に示す．図 8 の 1〜3 行目がこの CEG が抽出するコンテクストのうちの 1 つである「傘が忘れられている」（C_St1）の定義で，主格（Subject）と状態（State）のペアである．なお，Subject と State は抽象クラスとして定義されており，その表現形式に応じて，これを継承した実装クラスを定義する．図では単に文字列で表現する簡易なものである（StringSubject および StringState）．4 行目は C_St1 を抽出するための前提条件を 2 つの下位コンテクスト（CL1_StL11，CL2_StL22）と論理積演算を表すオブジェクトにより表現している．なお Context クラス，および Condition クラスは I_Condition インタフェースを実装しており，Condition クラスのコンストラクタの 1 番目と 3 番目の引数で I_Condition インタフェースを実装したオブジェクトを引数にとるため，4.2 節で述べた前提条件の論理式の 2 分木を構成することができる．5〜6 行目では C_St1 の遷移候補コンテクスト（C_St2）をセットしているが，候補は複数ありうるので配列として表現する（この例では 1 件）．7 行目ではコンテクスト抽出ルールを定義し，8 行目ではこのルールの管理テーブルを初期コンテクスト（C_St1）を引数として与えることで定義している．そして，9 行目ではこのルールを登録している．これらがコンテクスト抽出ルール，すなわちノードの機能定義情報となる．

また，10 行目でこのノードの基本情報を定義しており，主格，ノードのタイプ（この場合は CEG）をセットしている．そして，11 行目ではこのノード基本情報（NodeInfo）と前述の機能定義情報（RuleTable）をセットすることで，ContextEventGenerator オブジェクトを生成し，最後に起動している（12 行目）．

アプリケーション開発者が，1 つの CEG に関して記述することは以上である．このコードには，イベント到着時に他の下位コンテクストの最新状態を確認するロジックは含まれていないことに注意されたい．

```
抽出コンテクスト:
 C:(現在)家にFujiの傘が忘れられているか否か
     Subject(Sb)=傘/現在/家
     State=忘れられている(St1), いない(St2)
     状態遷移 St1->St2, St2->St1

下位コンテクスト:
 CL1:大阪のPM2時の天気予報が雨か否か
     Subject(SbL1)=天気予報/PM2時/大阪
     State=雨(StL11), 雨以外(StL12)
 CL2:(現在)玄関でFujiの傘が帯同されているか否か
     Subject(SbL2)=Fujiの傘/現在/玄関
     State=帯同されている(StL21), いない(StL22)

コンテクスト抽出ルール:
C_St1= CL1_StL11 AND CL2_StL22 ; C_St2
C_St2= CL1_StL12 OR  CL2_StL21 ; C_St1

■ただし, C*_xは状態xのコンテクストIDとする.
```

図 7 コンテクスト抽出ルールの例
Fig. 7 Example of context extraction rule.

```
1 : Subject Sb = new StringSubject(
                        "Fuji's Umbrella",
                        "NOW",
                        "home" );
2 : State St1 = new StringState("forgotten");
3 : Context C_St1  new Context(Sb, St1);
//- 同様にC_St2, CL1_StL11, CL2_StL22も定義 ----
4 : Condition cond1 = new Condition(
                                CL1_StL11,
                                new AND(),
                                CL2_StL22 );
5 : Context[] t1 = new Context[1];
6 : t1[0] = C_St2;
7 : Rule r1 = new Rule(C_St1, cond1, t1);
          //---- 同様にr2も定義 ----
8 : RuleTable table = new RuleTable(C_St1);
9 : table.putRule(r1);
10: NodeInfo node = new NodeInfo(Sb, Node.CEG);
11: ContextEventGenerator ceg
    = new ContextEventGenerator(node,table);
12: ceg.start();
```

図 8 記述例
Fig. 8 Sample code.

```
1 : public abstract class TimePrecision{
2 :   public abstract double calcTotalTimePrecision();
3 :   public boolean morePreciseThan(TimePrecision p){
4 :     if( calcTotalTimePrecision() >=
          p.calcTotalTimePrecision() ) return true;
5 :     else return false;
6 :   }
7 : }
```

図 9 抽象 TimePrecision クラス
Fig. 9 Abstract TimePrecision class.

5.1.3 抽象 TimePrecision クラス

3.3 節で述べた TimePrecision は，**図 9** に示すように抽象クラスとして表現され，複数の属性から総合的な大小値（総合値）への写像を定義する抽象メソッド（calcTotalTimePrecision）と総合値算出結果を基に大小比較を行うメソッド（morePreciseThan）が提

表 1 試験環境
Table 1 Experimental environment.

マシン	CPU: Pentium4 1.9 GHz
スペック	メモリ：512 MB, HDD:40 GB
OS	Windows2000 Professional SP2
実行環境	Java2 SE 1.4.0_01
通信ミドル	Jini Software Kit 1.2.1
ウェア	Java RMI

供され，前者についてはフレームワーク開発者が実装する．TimePrecision を使用した最新状態確認対象の選択アルゴリズムは，この抽象クラスを用いて実装されているために，後からフレームワーク開発者が写像を交換することは容易である．

5.2 基礎性能評価

表 1 に示す試験環境のもと，処理単位ごとの処理時間を把握するために，同一マシン上に構成要素（CEG（3つ），通信ミドルウェア（JavaSpaces，rmiregistry））を 1 プロセスずつ計 5 プロセス配備し，本システムの基本処理性能を測定した．性能測定にあたっては，Java により提供されているメソッド（`java.lang.System` クラスの `currentTimeMillis` メソッド）を利用して測定区間の先頭と末尾にタイムスタンプを埋め込み，差分を計算した．**図 10** に測定区間を示す．図中の濃い色の 3 つの CEG が，本測定で使用しているものに相当する．t_{prop} は下位 CEG が，イベント発行を通信ミドルウェアに依頼してから，上位 CEG がこのイベント受信を通信ミドルウェアから通知されるまでの時間を表し，実質的なイベントの伝播時間と見ることができる．t_{tr} は，通知を受けて次コンテクストへの遷移処理を行い，さらに上位 CEG に向けてイベント発行を通信ミドルウェアに依頼するまでの時間を表し，最新状態確認要否判定時間（t_{dec}）とその通信時間（t_{comm}），下位 CEG での最新状態確認時間（t_{rp}），遷移コンテクスト計算処理（t_{cal}）で構成される．また，t_{rp} はさらに下位 CEG への最新状態確認に関する処理時間で構成される．なお，num_{dec_n} と num_{req_n} は，それぞれ $n+1$ 段目の CEG が最新状態確認要否判定をする下位 CEG の数と，判定後に実際に最新状態確認要求をする下位 CEG の数を表す．測定にあたり，最低限 1 回の最新状態確認処理をともなうコンテクスト遷移処理時間を測定すべく，下位 CEG は 2 つとることとし，前提条件部をこれら 2 つの CEG からのコンテクストの論理積の結合として表現した．抽象 `TimePrecision` クラスの実装にあたっては，コンストラクタで単位時間あたりの更新回数（更新周波数）を渡し，これをそのまま総合値として返すよう，図 9 の `calcTotalPrecision` メソッドを実装している．また，最新状態確認を下位 CEG に対して行うことができるよう，総合値についてはイベントとしてやってくるものより他方が低いとした．そして，最新状態確認要求をされた CEG では，本来ならさらに下位 CEG に対する最新状態確認要否判定を行い，その結果に応じて最新状態確認要求を行うが，ここではつねに「変更なし」とし最新状態確認は不要としている．

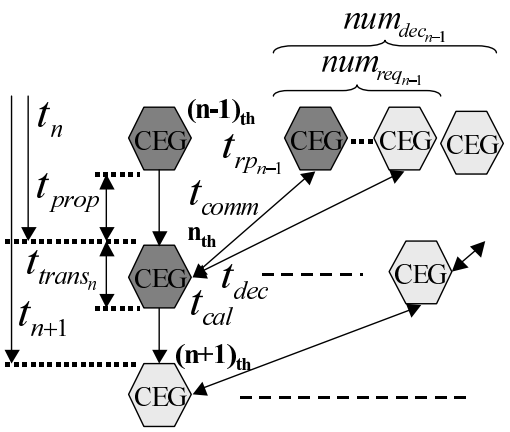

図 10 測定区間
Fig. 10 Measurement section.

表 2 測定結果［ミリ秒］
Table 2 Measurement results [msec].

t_{prop}	t_{dec}	t_{comm}	t_{cal}
197.0	23.3	80.3	36.7

表 2 に測定結果を示す．測定結果は各々の 3 回の取得結果の平均値で表している．この結果より，通信に関する部分（t_{prop} および t_{comm}）の割合が高いことが分かる．t_{prop} については Jini および JavaSpaces，t_{comm} については RMI の性能によるところが大きい．通信遅延の影響については次章で考察する．

6. 考察と課題

6.1 通信遅延の影響について

前節で得られた基礎性能数値にみられた通信遅延について，その影響を考察する．図 10 のモデルで，センサから得られたデータが n 段目の CEG にイベントとして到達するまでの所要時間を定式化すると次式のようになる．なお，t_{dec}，t_{cal}，t_{comm}，t_{prop} については等しいとする．また，$n=0$ はセンサを表す．

$$t_n = \begin{cases} t_{n-1} + t_{tr_{n-1}} + t_{prop} & (n \geq 1) \\ 0 & (n = 0) \end{cases} \quad (1)$$

where

$$t_{tr_n} = \begin{cases} t_{updt_{n-1}} \cdot num_{req_{n-1}} \\ +t_{dec} \cdot num_{dec_{n-1}} \\ +t_{cal} \qquad (n \geq 1) \\ 0 \qquad (n = 0) \end{cases} \tag{2}$$

$$t_{updt_n} = t_{rp_n} + t_{comm} \tag{3}$$

$$t_{rp_n} = t_{tr_n} \tag{4}$$

ここで，これらの式に表 2 の値を与え，さらに num_{req_n} については n によらず一定値（$num_{req} = 0$ or 4）を与える．そして，段数（n）を変数として通信遅延（t_n）を計算した．これを**図 11** に示す．$num_{req} = 4$ のときは極端な例であるが，つねに 4 つの下位 CEG に対して最新状態確認要求をする場合と考えることができる．たとえば，$n = 8$ で遅延は約 3,300,000 [msec] となり，“時間” のオーダとなってしまう．一方，$num_{req} = 0$ は 4.2 節および 4.3 節にて提案した手法が最大限機能して 1 つも問い合わせる必要がない場合と見なせ，$n = 10$ のときの遅延は 2,300 [msec] となる．しかし，このイベントのきっかけとなったセンサのデータ更新間隔がこれより短い場合には，イベントが 10 段目のノードに到達する前に次の下位コンテクストの変化をもたらすデータの変化が検出されて，再びこのノードに向かってイベント送信が始まる可能性がある．すると，つねに何世代前かのコンテクスト変化に追従することになり，利用者にとって意味のない振舞いをアプリケーションがすることになる．このことは，再利用性とスケーラビリティを考慮した実装を行ったときに，より顕著となることが予想される．すなわち再利用性を考慮し，抽出するコンテクストの粒度を細かくすると，必要な抽象度のコンテクストを得るまでのノード段数（n）は必然的に増加することになる．一方，ノードを複数のマシンに分散配置しスケーラブルにするとで，マシン間の通信が発生し，t_{prop} や t_{comm} がさらに増加することになる．このため，図 11 のグラフの増加率はさらに上昇し，10 段目でのイベント到達時間は先ほどの 2,300 [msec] を大きく上回ることになる．

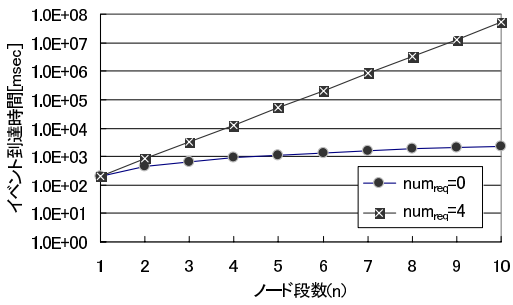

図 11 ノード段数とイベント到達時間の関係
Fig. 11 Relationship between number of node and arrival time of event.

よって，上記の要件を満足しつつ $n = 1$，$t_{prop} = 0$ および $t_{comm} = 0$，すなわち必要な抽象度のコンテクストを得るまでのノード段数とノード間の通信時間を最小化する必要があり，アプリケーション稼動時に他のアプリケーションの稼動状況を考慮してノードを分割・統合することが考えられる．さらに，分散環境下でのノードの配置に際しては，たとえばアプリケーションを構成しているコンテクスト抽出ルールに応じて，最もマシン間の通信が発生しないよう動的にノード配置を変更するといった案が考えられる．しかしこの場合でも，ノードの動的な分割・統合や再配置に関わるコストも鑑みて，総合的に振舞いを決定する必要がある．

6.2 フレームワークについて

本システムは，ユビキタス環境でのコンテクストアウェアなアプリケーション開発のためのフレームワークであり，ユビキタス環境でのアプリケーション開発に対する要求条件と，コンテクストアウェアなアプリケーション開発に対する要求条件を満たす必要がある．

前者については，物理空間とサイバー空間に存在するあらゆるオブジェクトが持つ多様性を吸収する機構があげられ，現在は TimePrecision がセンサから得られるデータ，および抽出されるコンテクストの時間的な不変性に関する多様性を吸収するのに用いられている．

表 3 に，提案フレームワークと既存研究例（Context Toolkit）[3] の開発工程における作業の比較を示す．5.1.2 節で見たように，本フレームワークではアプリケーション開発者が記述すべきプログラムコードに，上記の多様性を吸収する部分の記述は現れていないことから，各工程においてもそれを意識した検討や実装は発生しない．一方，Context Toolkit では 3.2 節で示したように，最新状態取得のためのメソッドが提供されるのみであるため，その使い方に関する作業が加わることになる．特に，開発後の維持工程では使用するセンサの変更などは容易に想定されることであり，アプリケーションのポータビリティという点で提案フレームワークの優位性が強調される．

後者については，コンテクスト情報やその抽出ルールの表現などがあげられるが，図 8 にみられるように 1 つの抽出ルールの表現に多くのコードを要するた

表 3 提案フレームワークと既存研究例[3] の各開発工程での作業比較
Table 3 Comparison of work between proposed framework and existing one [3] in each development process.

フェーズ	提案フレームワーク	Context Toolkit
設計	コンテクスト抽出ルールの検討	コンテクスト抽出ルールの検討 最新状態確認対象の選定
実装	コンテクスト抽出ルールの実装	コンテクスト抽出ルールの実装 最新状態確認処理の実装
維持	なし	最新状態確認対象の選定と実装

め，可読性は高いとはいえずメンテナンス性の点では課題が残るが，GUI（Graphical User Interface）を持ったルール記述ツールなるものを作成し，それを基にソースコードを自動生成することで改善が図れると考える．また，CEG をなんらかの名前管理機構で管理し，アプリケーションが必要とする最上位のコンテクスト情報を定義しただけで，自動的にその抽出に必要な下位 CEG を次々と導出する機構も生産性向上のうえで必要となると考える．また，TimePrecision の例で用いた CEG とコンテクスト情報双方の 2 層表現を他のメタコンテクスト情報へ応用することで，より高品質のコンテクストアウェアなアプリケーションを容易に開発可能となると考える．

6.3 最新状態確認の質について

5.2 節において述べたように，基礎性能測定で使用した抽象 `TimePrecision` クラスの実装クラスでは，単にデータの更新周波数のみを属性として持っており，更新周波数が低いセンサに対しては，たとえ直前にデータを受信していても無意味なポーリング（データ更新）を行ってしまうことになる．これに対しては，3.3 節で述べたように，前回のイベント到着（変化の検出）からの経過時間のような動的な属性もあわせ持つことで，よりその瞬間の状態にふさわしい，最新状態確認要否判断ができると考えられる．このような抽象クラス（`TimePrecision` クラス）の実装クラスに持たせる属性と総合値を算出するアルゴリズム（`calcTotalTimePrecision` メソッド）はフレームワーク開発者が決定すればよく，アプリケーション開発者が意識すべきコンテクスト抽出ルールから分離することが可能である．属性と実装アルゴリズムの検討は今後の課題である．

7. 関 連 研 究

開発環境の点からは，Solar [4] は Context Toolkit と同様に，コンテクストの段階的抽象化を行うための filter，transformer，merger，aggregator といった汎用的なオペレータを提供するものであるが，状態取得については One-time subscription と呼ぶイベントシステムのための publish/subscribe 機構が提供されており，これを用いて開発者がそのつど実装する必要がある．

QoC の点からは，文献 17) では，WWW サーチエンジンが効率的にサイト更新情報を獲得するためのアルゴリズムを考案するにあたり，「現在（current）」の概念を猶予期間（β）と猶予期間内の情報の確からしさ（α）で拡張し，α の確率で少なくとも現在より β だけ前に変化がおきていないことを “(β ;α)-current” と表現し，情報の鮮度を表現している．Precision に持たせる属性としてこのような指標も検討する必要があると考える．また，文献 5) ではコンテクストの形式表現について述べ，QoC や時間の概念を取り入れているが，具体的な開発フェーズへの利用については言及されていない．文献 6)〜8) では，ACCURACY，TIMELINESS，RESOLUTION，COVERAGE，FREQUENCY，CERTAINTY，REPEATABILITY，FRESHNESS，CONFIDENCE といった属性の集合で QoC を定義しており，今後の本フレームワークにおける QoC を検討するうえでのヒントとなる．さらに，文献 6) についてはメタコンテクストの概念を取り上げているが，実際のアプリケーションへの利用については言及されていない．

8. 結 論

本論文では，ユビキタス環境が持つ超多様性を意識せずに，アプリケーション開発者がアプリケーションロジックの開発に専念できる環境の提供を目的としたフレームワークを提案した．初めにフレームワークの基本設計について述べたのち，例題シナリオをあげ，センサのデータ更新間隔に関する多様性がアプリケーションの振舞いに不整合をもたらす可能性と，その問題に対する解決策の既存研究例でのアドホックさを指摘した．そしてコンテクスト情報をベースコンテクスト情報とメタコンテクスト情報とに分離し，後者に TimePrecision という指標を導入することで，アプリ

ケーション開発者の実装に依存せずにシステマティックにこの問題を解決する方法を提案した．そして，これに基づいた実装および基礎性能評価を実施し，その可能性と課題について考察した．今後は，課題を解決しつつ実際のセンサ/アクチュエータ/物理オブジェクトを用いた前出の例題シナリオの実装を通して，フレームワークとしての洗練を行う予定である．

参 考 文 献

1) Weiser, M.: The Computer for the Twenty-First Century, *Scientific American*, pp.94–104 (1991).
2) Chen, G. and Kotz, D.: A Survey of Context-Aware Mobile Computing Research, Technical Report TR2000-381, Department of Computer Science, Dartmouth College (2000).
3) Dey, A., Abowd, G. and Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, *HUMAN-COMPUTER INTERACTION*, Vol.16, No.2–4, pp.97–166 (2001).
4) Chen, G. and Kotz, D.: Context Aggregation and Dissemination in Ubiquitous Computing Systems, Technical Report TR2002-420, Department of Computer Science, Dartmouth College (2002).
5) Henricksen, K., Indulska, J. and Rakotonirainy, A.: Modeling Context Information in Pervasive Computing Systems, *Proc. International Conference on Pervasive Computing, Pervasive 2002*, pp.167–180 (2002).
6) Gray, P. and Salber, D.: Modelling and Using Sensed Context Information in the Design of Interactive Applications, *Proc. 8th IFIP Working Conference on Engineering for Human-Computer Interaction, EHCI'01*, pp.317–335 (2001).
7) Ebling, M., Hunt, G. and Lei, H.: Issues for Context Services for Pervasive Computing, *Proc. Workshop on Middleware for Mobile Computing in Middleware 2001* (2001).
8) Castro, P. and Muntz, R.: Managing Context data for Smart Spaces, *IEEE Personal Communications*, pp.44–46 (2000).
9) JiniCommunity: Jini.org. http://www.jini.org/
10) OMG: CORBA. http://www.corba.org/
11) AIM: RF-ID. http://www.aimglobal.org/technologies/rfid/
12) Brooks, R. and Iyengar, S.: *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Pearson Education (1997).
13) W3C: EBNF. http://www.w3.org/TR/REC-xml #sec-notation
14) Freeman, E., Hupfer, S. and Arnold, K.: *JavaSpaces^{TM} Principles, Patterns, and Practice*, Addison-Wesley (1999).
15) Sun Microsystems: *Java Remote Method Invocation Specification.* ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf
16) Carriero, N. and Gelernter, D.: Linda in Context, *Comm. ACM*, Vol.32, No.4, pp.444–458 (1989).
17) Brewington, B. and Cybenko, G.: Keeping up with the changing Web, *IEEE Computer*, Vol.33, No.5, pp.52–58 (2000).

(平成 14 年 12 月 21 日受付)
(平成 15 年 4 月 17 日採録)

藤波 香織

昭和 45 年生．平成 7 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年日本電信電話（株）入社．平成 9 年より NTT コミュニケーションウェア（現 NTT コムウェア）にてモバイルシステムの実用化研究に従事．平成 14 年早稲田大学大学院理工学研究科情報科学専攻博士課程入学．ユビキタスコンピューティングのミドルウェアに関する研究に従事．電子情報通信学会会員．

中島 達夫（正会員）

早稲田大学理工学部コンピュータ・ネットワーク工学科教授．ユビキタスコンピューティングや分散システムのシステムサポート，およびシステム開発技法に関する研究に従事．