モバイルコンピューティングと
ワイヤレス通信　23−19
高度交通システム　11−19
（2002．11．29）

ウェアラブル/移動情報端末におけるコンテキスト指向プロセス管理

久住憲嗣[†1]　中西恒夫[†2,†3,†4]
北須賀輝明[†2]　福田晃[†2,†4]

近年, ウエアラブルコンピュータの研究がさかんである. 常に利用者が装着するウエアラブルコンピュータは, 利用者が操作に専念できない状況が考えられ, センサ等の情報から推測される利用者の置かれている状況をもとに, できるだけ自動的に利用者の生活の補助をすることが望ましい. いわゆるコンテキストアウエアネスが必要である. そこで本稿では, コンテキストアウエアアプリケーションの実装を容易化するミドルウエアを提案する. 提案ミドルウエアにおいてコンテキストを統一的に扱うために, コンテキストモデルを集合論・関係代数を用いて形式的に定義する. また, コンテキストアウエアアプリケーション実装者の負担を軽減する, コンテキストに基づくプロセスの自動制御手法について述べる. さらに, コンテキストモデル及びコンテキストに基づくプロセス自動制御手法の実装方針について述べる.

キーワード: ウエアラブルコンピュータ, コンテキスト, アクティブデータベース

The Concept of Context Oriented Process Control on Wearable and Mobile Computing

Kenji Hisazumi,[†1] Tsuneo Nakanishi,[†2,†3,†4] Teruaki Kitasuka[†2]
and Akira Fukuda[†2,†4]

In recent years many researchers have been investigating wearable computers. Since users cannot always concentrate on operating wearable computers, applications running on them must be it context-aware, that is, help users according to situations the users face. In this paper we propose a middleware to ease implementation of context-aware applications. We define the context model formally with set theory and relational algebra to handle context on the middleware. We also propose a concept which relates process activity to context. Finally, we discuss implementation of the middleware.

Keywords: Wearable computer, context, active database

1. はじめに

昨今, ウエアラブルコンピュータの研究が盛んである[1]. 常に利用者が装着するウエアラブルコンピュータは, 利用者が操作に専念できない状況が考えられる. ウエアラブルコンピュータ上で動作するアプリケーションはセンサからの情報等から推測される利用者の置かれている状況をもとに, できるだけ自動的に利用者を生活の補助をすることが望ましい. いわゆる, コンテキストアウエアネスが必要となる. コンテキストとは, アプリケーションからみた環境の状態であり, 位置情報, その他のセンサからの情報, コンピュータ環境の状態, 記憶装置内の情報, また, これらを組み合わせることにより推測される情報がある.

コンテキストアウエアアプリケーションを通常のオペレーティングシステムのみを利用し実装する際には, プログラマは, コンテキストを取得しコンテキストに基づいて振る舞うように, コードを手続き的に記述しなければならない. しかし, これらの動作を漏れなく記述するのは難しい. しかも, コンテキストを取得する手続きは, すべてのコンテキストアウエアアプリケーションにおいて共通の手続きである.

そこで, 本研究では, コンテキストアウエアアプリケーションの構築を容易化するミドルウエアを提案する. 提案ミドルウエアはオペレーティングシステム上

†1 九州大学システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 九州大学システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University
†3 システム LSI 研究センター
System LSI Research Center, Kyushu University
†4 九州大学情報基盤センター
Computing and Communications Center, Kyushu University

で動作し, コンテキストをセンサからのデータ, 記憶装置内のデータなどから導出し, アプリケーションが利用可能な形に抽象化する. また, 提案ミドルウエアは, アプリケーションプログラマによってあらかじめ宣言的に定義された通りに, コンテキストに基づいたプロセス活動の自動制御を行う. 提案ミドルウエアは, ウエアラブルコンピュータ, 情報家電, PDA, カーナビゲーションシステムなどを主な応用対象とする.

以降,2 節では本稿で提案する概念を応用例を用いておおまかに説明する. 3 節では提案ミドルウエアが前提とするコンテキストモデルを集合論と関係代数を用いて定義し, コンテキストに基づくプロセス活動の自動制御手法について述べる. 4 節では 3 節で議論したモデルに基づいた, 提案ミドルウエアの実装方法について議論する. 最後に,5 節で本研究のまとめと今後の課題を述べる.

2. 応　用　例

本節では, 本稿で提案する概念を, 応用例を通じておおまかに説明する.

TEA[2] では, 携帯電話をコンテキストに基づいて振る舞わせる手法が示されている.TEA で扱う携帯電話には様々なセンサが搭載されており, そのセンサの情報から周辺環境を推定する. 推定された周辺環境に基づいて, 着信音量やバイブレーション機能の入, 切を制御する. 例えば, 室内に携帯電話の持ち主がいるときには着信音量を小さくし, 室外であれば着信音量を大きくする. また, 会社にいるときには着信音を切り, バイブレーション機能を入れる.

提案ミドルウエアの応用例としては簡単すぎるが, この例をもって提案ミドルウエアのコンテキストモデルとプロセス管理について概説する. n 個のセンサから取得した値の組について, 考えられるすべての組からなる空間を, コンテキスト空間として定義する. コンテキスト空間を,「会社」,「室内」,「室外」を意味する部分空間に分割する (図 1). このコンテキスト空間の部分空間は, あらかじめプログラマが宣言的に指定する. 現在のコンテキストがどの部分空間に属しているかにより, 着信音量やバイブレーション機能の入, 切は決定される. この振る舞いを実現するために, 提案ミドルウエアは各部分空間対応付けられるプロセスを生成する. 提案ミドルウエアは, 現在のコンテキストがある部分空間内から他の部分空間内に遷移した際に, 自動的に前者に対応付けされたプロセスを休眠させ, 後者に対応付けされたプロセスを起こす. プロセスは起床した際に, 着信音量やバイブレーション機能の入, 切を設定する. たとえば,「会社」に対応付けされたプロセスは, 着信音を切り, バイブレーション機能を入れる.

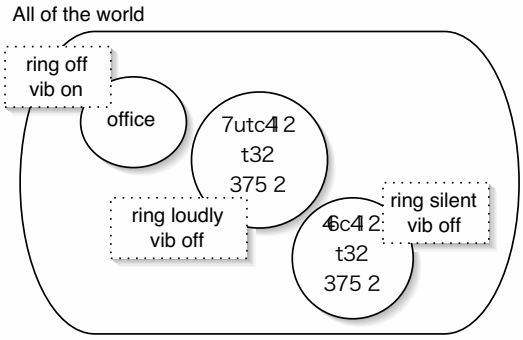


図 1　コンテキストモデルとプロセス管理の例
Fig. 1　An example application

3. コンテキストモデル及びコンテキストに基づくプロセス活動の自動制御

3.1 導　　入

本節では, 提案ミドルウエアで前提とするコンテキストモデルについて議論する.

複数の研究者によってコンテキストは議論され, 定義されている[3]. ここでは,Dey らの定義を引用する[4].

> Context: any information that can be used to characterize the situation of entities (*i.e.* whether, a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects.

この定義は, 今日, コンテキストアウエアコンピューティングの分野において想起されるコンテキストの定義としてもっとも受け入れられやすいものである. しかしながら, コンピュータ上でコンテキストを扱うには, この定義は形式性を欠く.

Henricksen らは文献 5) において, オブジェクト指向モデルを採用した形式的なコンテキストモデルを提案している. このモデルは, エンティティを定義し, それらの関係を示すことにより, コンテキストを表現する. この関係には動的/静的, 単体/複合などのクラスがある. また, 関係は情報の確かさなど自身に関する情報のクオリティを注釈として持ち, それらのクオリティはひいてはコンテキストの信頼性の評価に用いられる. さらに関係の間には依存関係が定義され, 二次的な関係

の導出に使用される. Henricksen らの研究は, コンテキストモデルの分析には利用できるが, そのソフトウエア的実装については議論の範囲外である. 本稿で示すモデルは集合論と関係代数に基づくものであり, 基本的に Henricksen らのモデルと矛盾しないが, より実装について考慮されている. ソフトウエア工学の分野で研究されている形式仕様記述言語は, 集合論・関係代数に基づくものが少なくない[6]. 本研究ではこれらの分野の成果を導入してコンテキストアウエアアプリケーションに形式的検証をほどこしコンテキストアウエアアプリケーションの信頼性・安全性・セキュリティの向上を図る.

形式的なコンテキストモデルを議論する前に, コンテキストアウエアアプリケーションに要求されるコンテキストモデルの性質を以下に挙げる.

- 再現性:
  同じ入力をモデルに与えたとき同じ結果がでる必要がある. システムの安定性, 実装の容易性の観点から, 設計時にシステムの振る舞いは予測可能である必要がある.
- 実現可能性:
  机上のモデルに終わるのではなく, 各種入力からモデルに従ってコンテキストを表現できる必要がある.
- 理解容易性:
  理解が容易でなければ, アプリケーションの実装が困難になり, バグが増加する原因となる. 再現性とともに重要な性質である.
- 検証可能性:
  コンテキストモデルは形式的に検証できなければならない. 検証可能であれば, より信頼性を必要とする分野で提案ミドルウエアを利用しコンテキストアウエアネスを持ったシステムを構築することが可能になる.

3.2 コンテキストの集合表現

本節では, 集合と関係代数を用いて, 前述した性質をもつコンテキストモデルを提案する.

提案ミドルウエアは, センサや記憶装置等からの未加工データを基に現在のコンテキストを導出する (図 2). システム中にはアプリケーションが複数存在し, アプリケーションは複数のコンテキストを扱う. また, ひとつのコンテキストは, ひとつ以上の未加工データまたは他のコンテキストから導出される. 未加工データ自身も一種のコンテキストと言えるので, 以下では未加工データもコンテキストとして扱う. センサや記憶装置中のデータソースを $s_i \in S_i$ $(1 \leq i \leq n)$ とする. システム中にコンテキスト c_i が m 個存在するものとすると,c_i $(1 \leq i \leq n)$ はデータソース s_i そのものとし,c_i $(n+1 \leq i \leq m)$ は他のコンテキストとする. c_i $(1 \leq i \leq n)$ は以下の式で与えられる.

$$c_i = s_i$$

あるコンテキスト c_i $(n+1 \leq i \leq m)$ は, 他のコンテキスト $c_{ij} \in C_{ij}$ $(ij \neq i, 1 \leq j \leq l < m)$ から構成され, 以下の式で与えられる.

$$c_i = f_i(c_{i1}, c_{i2}, \ldots, c_{il})$$

f_i は,$C_{i1} \times C_{i2} \times \ldots \times C_{il}$ から C_i への写像である. f_i は直接的または間接的に再帰的であってはならない. C_i は c_i が取りうるすべての値の集合であり, 以下の式で与えられる.

$$C_i = \{c_i = f_i(c_{i1}, c_{i2}, \ldots, c_{il}) \mid c_{ij} \in C_{ij}, 1 \leq j \leq l < m, ij \neq i\}$$

周辺環境が変化すると,c_i は C_i 内を移動することになる (図 3).

ここで出てきた, 写像 f_i は関係代数で定義する. f_i は関係演算である直積 $\times$, 射影 $\pi_{\pi c}$, 選択 $\sigma_{\sigma c}$ の組み合わせから構成される. πc は射影条件,σc は選択条件である. すなわち, 変換に利用する他のコンテキスト $c_1, c_2, \ldots, c_l$, またそれらがとりうるすべての値の集合を $C_1, C_2, \ldots C_l$ とすると, 写像 f を

$$f_i(c_1, c_2, \ldots, c_l) = \pi_{\pi c}(\sigma_{\sigma c}(c_1 \times c_2 \times \ldots \times c_l))$$

と定義する.

以上で, コンテキストモデルを定義した.

3.3 コンテキストに基づくプロセス活動の自動制御

本節では,2 節でアプリケーション例を通して示した, コンテキストに基づいて自動的にプロセス活動を制御する手法について, 詳細に議論する.

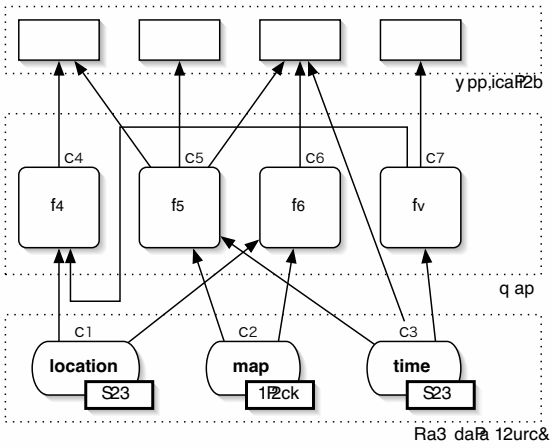


図 2 コンテキストの導出
Fig. 2 Deriving of context

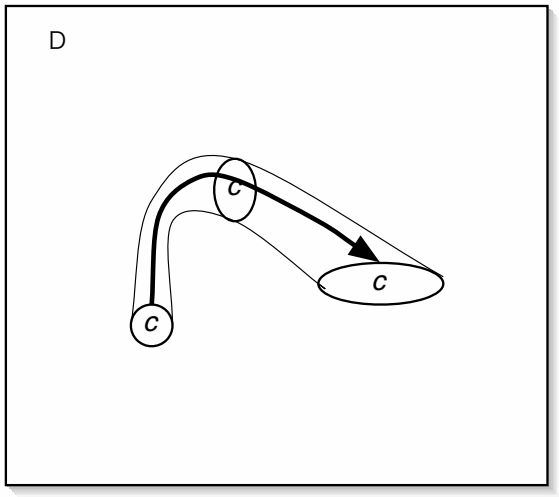


図 3 コンテキスト全体集合と現在のコンテキスト集合
Fig. 3 All set of context and a subset of current context

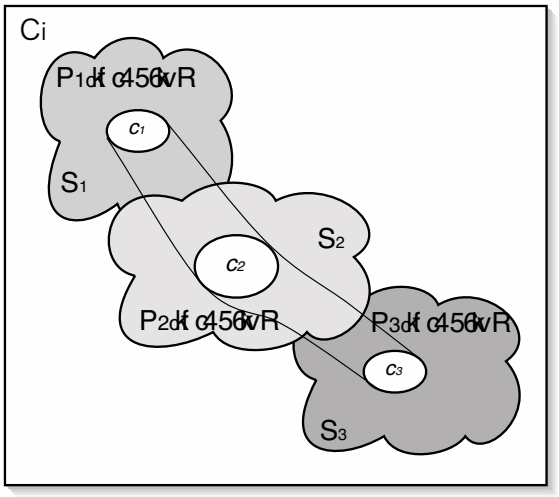


図 4 コンテキスト部分集合とプロセス
Fig. 4 Subsets of context and process.

コンテキストに基づいたプロセス活動の自動制御とは, 現在のコンテキストがアプリケーションによりあらかじめ定義されたあるコンテキスト部分集合に入ったときにその部分集合に対応するプロセスを起床させ, また, 定義されたコンテキスト集合から外れたときにはプロセスを休眠させることと定義する. これは, プロセス活動自動制御の対象とするコンテキスト集合 C_i を分割し, 部分集合 $Q_j(1 \leq j \leq q)$ に, プロセスを対応づけし, 部分集合 Q_j に現在のコンテキスト c_i が入った際に p_i を起床させ,C_i から去った際に p_i を休眠させることである (図 4, 5).

C_I の部分集合 Q_i 群を定義する, 現在のコンテキスト c_i から起床プロセス集合 P_c への写像を f_{active} とすると,P_c は以下の式で与えられる.

$$P_c = f_{active}(c_i)$$

ここで,P_c が取りうるすべての値の集合を P_{ac} と定義する. すなわち,

$$P_{ac} = \{P_c = f_{active}(c_i) \mid c_i \in C_i\}$$

と定義する. P_{ac} に含まれる値は整数値とし, ミドルウエアは整数値とプロセスを一対一に対応付けし, プロセス活動の制御を行うキーとする. この整数値は OS が管理するプロセス番号とは独立の数値であり, この整数値の導入により,OS の実装と独立にミドルウエア内でプロセスを表現する. P_c に含まれる整数値は 0 個以上であり,2 個以上同時にプロセスを実行することをミドルウエアは許可している. しかし, 複数プロレスを同時に実行することを前提としていないアプリケーションも考えられ, その場合は,P_c に含まれる整数値は, 必ず 0 個か 1 個でなくてはならない. 写像 f_{active} はすべてのコンテキスト集合を網羅し, プロセスと対応づける必要はないが, アプリケーションが真に興味があるコンテキスト集合の補集合, すなわち写像 f_{active} がプロセスと対応付けするコンテキスト集合群の和集合の補集合に, プロセスをひとつ対応付けし, 例外処理を行うのが望ましい.

3.4 プロセス数の増加

本稿で提案するコンテキストに基づくプロセス自動制御手法を使用すると, プロセスが多数生成される. 実際に生成されるとは限らないが, 最大で P_{ac} の要素数分プロセスが生成される可能性がある. $P_c = f_{active}\{f(c_1, c_2, \ldots, c_l)\}$ なので, 入力数 l を減らせば,f が扱う全体集合が縮小し,P_{ac} の要素数が減少する. また,c_i 群がとりえない値を示したとき,f_{active} の結果に反映しないような f_{active} を定義すると,P_{ac} の要素数が減少する. 前者の本質は, 扱うコンテキストを, プロセス自動制御に利用するコンテキストとしないコンテキストに明確に分類することである. この分類はアプリケーションにより異なるが, アプリケーションを分類し, プロセス自動制御に使用するコンテキストのガイドラインは示せるであろう. 後者は, 真にアプリケーションが興味をもつコンテキスト集合以外はマスクし利用しないことである. 不必要なプロセス生成を防ぐ他に, アプリケーションの誤動作を防ぐ効果がある.

3.5 適 応 範 囲

コンテキストに基づくプロセス制御手法の適用範囲は以下の通りである.

- コンテキスト集合を意味のある単位で分割できること:

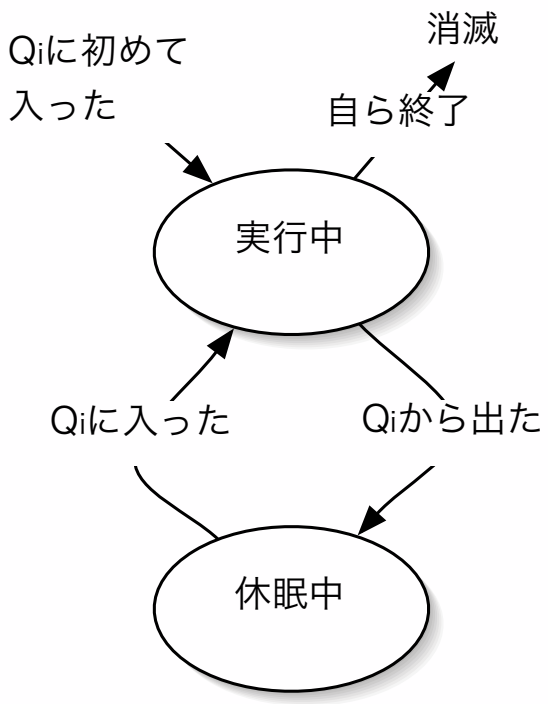


図 5 プロセス状態
Fig. 5 States of controlled process.

扱いたいコンテキストの部分集合を明確に定義できない応用には, コンテキストとプロセスを対応づけることが困難なので本提案手法は不向きである. ただし, f_{active} は静的である必要はなく矛盾しなければ動的に変化しても構わない.

- 分割された集合に割り当てられたプロセス同士が, 比較的独立であること:
  プロセス同士が互いに必要な情報を持ちあい, 情報が複数のプロセスに分散する応用では, プロセスを分割しても複数のプロセスを実行し続けなければならず, システムの性能低下につながる.
- 単純すぎないこと:
  携帯電話をコンテキストに基づいて振る舞わせる応用のような単純なシステムの場合, ひとつのプロセスに実装するほうが, 実装効率, 実行効率の両方の面で有利である.

このような条件を踏まえ, 特に提案方式が有用な応用を以下に挙げる.

紛失・盗難対策

ノートパソコン, ウエアラブルコンピュータ等には, 個人的な情報が記録されており, 紛失・盗難の際のデータの漏えいが問題となる. 機密情報にアクセスできるアプリケーションを一つに限定し, そのアプリケーションの動作範囲を, コンピュータの持ち主が近傍に存在するとき, オフィス内などに限定する. 近傍に存在する, オフィス内などの周辺状況はセンサからの情報から導出する. これにより, コンピュータの持ち主が近傍に存在しないとき, オフィス外に持ち出されたときには機密情報にアクセスできなくなる. この方式の利点は, 利用者にはパスワードを入力する等の負担をかけずに機密情報を守ることができる点であり, また, 汎用のミドルウエアを応用しているので, 機能を作り込むことなしにユーザの設定のみで実現可能な点である. ただし, センサ情報のかく乱対策, ミドルウエアのセキュリティ強化などが重要な課題となる.

4. 実　　装

本研究で提案するミドルウエアを使用したシステムは, 各種センサ群, ミドルウエア, アプリケーション群から構成される (図 6). ミドルウエアは, すべての入出力を管理するドライバ群, イベント配送を制御するイベントコントローラ, プロセス活動の自動制御を行うプロセス活動コントローラ, 関係代数を解釈しコンテキストの導出を行うアクティブデータベースから構成される. 本節では, これらについて記述しつつ, 実装の方針を示す.

4.1 コンテキスト情報のアプリケーションへの伝達

アプリケーションがあらかじめ定義した写像 f_i に基づき, ミドルウエアはデータソースからのデータをコンテキスト情報に変換する. そのコンテキスト情報をアプリケーションに伝達する方式は二種類あり, 一方はアプリケーションが明示的にコンテキスト情報を取得する方式, 他方はコンテキスト情報が変化した際にイベントとしてアプリケーションにコンテキスト情報の変化を伝達する方式である. 前者は, 写像 f_i に他のコンテキスト $c_j(1 \leq j \leq l)$ からの情報を渡し評価することが可能であれば, トップダウンに f_i が必要とするコンテキスト c_j を評価することにより, アプリケーションはコンテキスト情報を取得することができる. この機能は後述するアクティブデータベースにより実現される. 後者は, あるデータソースが変化した際に, ボトムアップにそのデータソースに関連する写像を評価し, 最終的にアプリケーションにコンテキスト情報の変化を通知する必要がある. 後者の実現のために, $c_i(1 \leq i \leq m)$ の変化をイベントとして扱い, そのイベントをアプリケーションやプロセス活動コントローラに分配する機構が, イベントコントローラである. アプリケーション等は, 変化の通知を希望するコンテキスト c_i をイベントコントローラに登録する. イベント

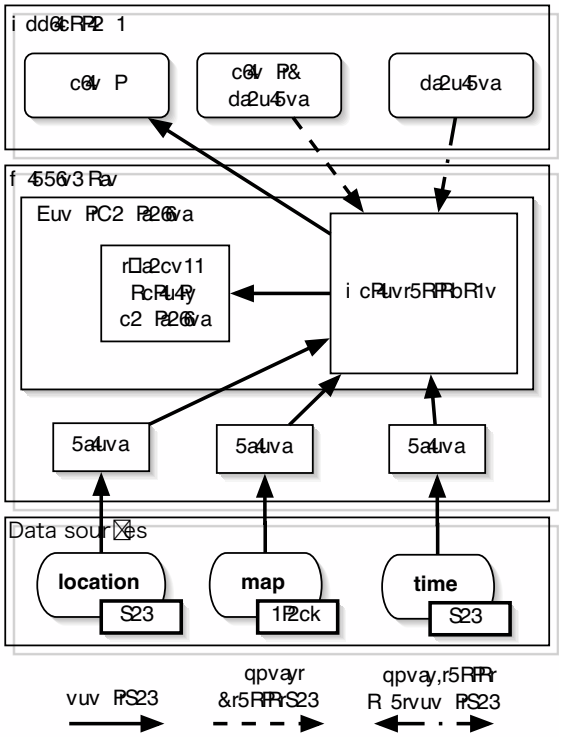


図 6 システム構成
Fig. 6 System architecture

コントローラは, アクティブデータベースに c_i が変化した際にイベントを発生させる設定を行う. アクティブデータベースは c_i が変化した際にイベントをイベントコントローラに通知するので, それを利用しイベントコントローラは登録者にイベントを通知する. そのイベントを受信することにより, 登録者はコンテキスト情報の変化を知覚できる.

4.2 データソースとドライバ

データソースは, データソースからのデータの即時性の観点からみて, 静的なデータソースと動的なデータソースに分類される. 静的なデータソースの例としては, スケジュール, 地図があり, 動的なデータソースの例としては位置情報, 向き, 時刻, 温度などが挙げられる.

データソースはデータの取得する際の性質から, プッシュ型, プル型に分類される. プッシュ型データソースは, データソースが変化した際に, データソースからミドルウエアにデータの変更イベントを通知する. プル型データソースは, データソースが変化してもミドルウエアに通知せず, ミドルウエアがデータを取得しない限り, ミドルウエアはデータの変化を知りえない. ここで問題となるのがプル型のデータソースであり, プル型のデータソースはデータの変化を即時的に検知できない. プッシュ型データソースであれば, プッシュ型データソースからのイベントを最初の契機として, f_i 群を評価することにより, 最終的にアプリケーションにコンテキスト情報の変化をイベントとして通知できる. ところが, データの変化を即時的に検知できない場合, データソースからのイベントを f_i 群の評価のための最初の契機とすることができず, コンテキスト情報の変化をアプリケーションに通知できない. ここで, 文献 7) はコンテキスト情報のクオリティをコンテキスト情報のメタ情報として持たせることを提案している. この文献中ではいくつかの指標を提案しているが, 本システムにプル型データソースを利用する際に重要となるのは *frequency* である. *frequency* はデータソースの変化頻度の尺度である. 周期的にデータソースが変化する場合は *frequency* に基づきデータをプル型データソースから取得してくることにより, プッシュ型のデータソースと見なすことが可能である.

静的なデータソースはさらに, ミドルウエア外部のデータソースである外部データソース, ミドルウエア内部のアクティブデータベース内のデータである内部データソースの二種類に分類される.

外部データソースからデータを取得し, アクティブデータベースにデータを受け渡すのが, ドライバである. ドライバは外部データソースひとつづつにそれぞれ用意するが, 受け渡すデータの内容が一般的なバイト列であれば, システムが標準提供するドライバを設定し利用でき, 個々に実装する必要はない. ドライバはデータソースの型により動作が異なり, プッシュ型データソース用ドライバは, データソースからのイベントをもとにアクティブデータベースにデータを渡し, プル型データソース用ドライバは, データソースをメタ情報 *frequency* に従いデータを取得し, データが変化した際にアクティブデータベースにデータを渡す. ドライバは, 特別な理由がない限りデータソースのデータをそのままアクティブデータベースに渡すことが望ましい. データソースは複数のアプリケーションにより利用され, あるひとつのアプリケーションに都合の良い前処理をデータに対して行うと, 他のアプリケーションがそのデータを利用できなくなる可能性があるからである. データを加工する必要があれば, 写像 f_i のひとつを前処理用に定義するべきである.

4.3 アクティブデータベース

本稿で提案するコンテキスト導出法では, 各種センサなどからの未加工データを関係代数で定義された写像を用い, アプリケーションにとって意味のあるコン

テキストに変換する. 関係代数で定義された写像を評価できるシステムとして, リレーショナルデータベースがある. リレーショナルデータベースは, トップダウンに関係代数を評価しデータソースをコンテキスト情報に変換することは可能だが, データソースが変化した際にボトムアップに関係代数を評価しコンテキスト情報に変換するのは困難である. ボトムアップに関係代数を評価可能なデータベースシステムとしてアクティブデータベース[8] があり, 提案ミドルウエアの実現に適している. 従来のリレーショナルデータベースではユーザが問い合わせをしてはじめて処理を行うのに対して, アクティブデータベースでは一般にあるイベント (すなわちデータベースの検索, 更新などの操作) に反応して別のアクション (データベースの検索, 更新などの操作) が起動される. その動作は, イベント (事象), コンディション (発火条件), アクション (動作) の組で表される ECA ルールで記述する. アクティブデータベースを提案ミドルウエアに応用すると,「写像 f_i が使用する入力が変更された」がイベント,「常に」がコンディション,「写像 f_i を評価する」がアクションとなる. この ECA ルールの組み合わせにより, ボトムアップに写像 f_i 群を評価することが可能である.

あるデータソースが変化し, ドライバからイベントが通知されデータが渡された際には, そのデータをもとにアクティブデータベース内のテーブルを変更する. データベースが変更されると, アクティブデータベース内のそのデータソースを利用する写像 f_i がボトムアップに再評価される.

イベントコントローラが c_i の変化通知を希望した場合,「写像 c_i が変化した」がイベント,「 常に」がコンディション,「イベントコントローラに通知する」がアクションとなる ECA ルールをアクティブデータベースに登録し,c_i が変化した場合にイベントコントローラにイベントを通知する.

写像 f_i 群は動的に設定可能である. また, 写像 f_i 群を評価する際に必要なデータを自由にデータベースに登録可能であり, これが前述した内部データソースである. 内部データソースには二種類存在する. 一方は, 写像 f_i を評価する際に必要なデータを, アプリケーションがあらかじめデータベース内に登録しておいたものである. 例えば, 温度を「暑い」「普通」「寒い」に分類する際には, それぞれの間のしきい値をデータベースに登録し, しきい値と現在の温度をもとに,「暑い」「普通」「寒い」を決定する. 他方は, 写像 f_i の結果をアプリケーションに渡すデータとして利用せず, データベースに登録したものである. 過去の履歴をもとに, コンテキストを導出する際に使用する.

このアクティブデータベースはシステム内にただひとつである. つまり, 写像や内部データソースは複数のアプリケーションから共有されうる. 他のアプリケーションから読まれたくない写像や内部データソースは, データベースのアクセス制限の機能を利用し, 読み書き変更不可に設定することにより, 安全に運用できる. しかし, 共有したい写像や内部データソースは, あるアプリケーションの利用中に, 他のアプリケーションにより変更されると問題が発生する場合がある. これは, 読み出しは許可し, 変更を不可に設定することにより, 安全に運用することが可能である.

4.4 プロセス活動コントローラ

本節では 3.3 節において議論したプロセス活動の自動制御手法の実装について議論する. アプリケーションは, ミドルウエア中のプロセス活動コントローラにあらかじめ写像 f_{active} を設定することで, プロセス活動の自動制御を行わせる. 写像 f_{active} の設定機会は, アプリケーションインストール時にインストーラによる設定, アプリケーション実行時に自ら設定, 外部ユーティリティにより設定, の三種類ある.

ミドルウエアは, 写像 f_{active} が示す集合に基づき, プロセスの生成, 起床を行う. また, 休眠要求はイベントとして休眠すべきプロセスに通知され, そのイベントを受けてプロセスは自発的に休眠する. これは, 休眠前処理を行う機会を与えるためである.

プロセス活動コントローラは写像 f_{active} をアクティブデータベースに登録し, また, イベントコントローラに f_{active} にイベント通知を依頼する. プロセス活動コントローラは,f_{active} が変化した際にイベントを受け取り, 以下の処理を行うことによりプロセス活動の自動制御を行う.

- P_c に p_i が新たに追加された場合:
  ミドルウエアはプロセス p_i を起床させ, 起床イベントを p_i に通知する. p_i が起動されていなければ,p_i を起動する.
- P_c から p_i が取り除かれた場合:
  ミドルウエアはプロセス p_i に休眠要求イベントを通知し, 休眠を促す.

コンテキスト情報の変化を検知したい場合は, 別途イベントコントローラに登録を行う. 起床中は問題なくイベントを受信できるが, 休眠中はコンテキスト情報の変化イベントは受信できない. 休眠中に受信できるイベントは, 起床イベントのみである.

4.5 複雑なコンテキストの取り扱い

本稿で提案するコンテキストモデルは, 形式的に検

証することが容易であるが, オブジェクトやエージェントを基本としたモデルと比較すると, 記述力が劣る. そこで, より複雑なコンテキストを手続き的に記述可能なモジュールを定義し, このモジュールをデータ源として利用できるモデルを導入する. このモジュールは, 他のデータソースやコンテキスト情報を利用し, 内部でデータを手続き的に変換し, データ源としてミドルウエアに変換後のデータを渡す. この機能はコンテキストの記述力を向上させるが, 網羅的に形式的検証ができなくなるため, 検証が必要な応用には向かない.

5. おわりに

コンテキストアウエアネスはウエアラブルコンピュータに必要不可欠である. コンテキストアウエアネスをもつシステムは, 利用者の状況を判断し自動的に補助を行う反面, 誤動作が致命的な結果をもたらす可能性があり, 信頼性・安全性・高セキュリティが必要不可欠である. 本稿では, 信頼性・安全性・高セキュリティを形式的検証により向上すべく, コンテキストを集合論・関係代数を用いて定義した. また, コンテキストアウエアアプリケーション実装者の負担を軽減するために, コンテキストに基づくプロセスの自動制御手法について述べた. 本手法はコンテキスト部分集合をプロセスと対応付けし, 現在のコンテキストが部分集合内に入ったときにプロセスを再開させ, 出たときに休眠させることにより, 現在のコンテキストによりプロセスの切り替えを行う. この手法によりアプリケーションプログラマは煩雑なコンテキストの扱いを明示的に記述しなくてすむ. また, 利用者に負担をかけないセキュリティ対策への応用が可能である.

本稿では, コンテキストモデル及びコンテキストに基づくプロセス自動制御手法の実装方針を述べた. 現在, 提案ミドルウエアのプロトタイプを実装中であり, 今後提案手法の実証を行う予定である.

謝辞 本研究の一部は，科学技術振興事業団（JST）の戦略的基礎研究推進事業（CREST）「高度メディア社会の生活情報技術」プログラムの支援によるものである.

参考文献

1) 科学技術振興事業団戦略的基礎研究推進事業「高度メディア社会の生活情報技術」: 日常生活を拡張する着用指向情報パートナーの開発 研究発表資料集 (2002).
2) Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V. and de Velde, W. V.: Advanced interaction in context, *Proc. of HUC'99*, pp. 89–101 (1999).
3) Chen, G. and Kotz, D.: A Survey of Context-Aware Mobile Computing Research, Technical Report TR2000–381, Dartmouth Computer Science Thechnical Report (2000).
4) Dey, A.K. and Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, *HCI Journal 2001* (2001).
5) Henricksen, K., Indulska, J. and Rakotonirainy, A.: Modeling Context Information in Pervasive Computing Systems, *Proc. of Pervasive 2002*, pp. 167–180 (2002).
6) Potter, B., Sinclair, J. and Till, D.: *An Introduction to Formal Specification and Z*, Prentice hall (1996).
7) Gray, P. and Salber, D.: Modelling and Using Sensed Context Information in the Design of Interactive Applications, *Proc. of EHCI 2002*, pp. 317–335 (2001).
8) 石川博: アクティブデータベース, 情報処理, pp. 124–129 (1994).