卒業論文　　2004 年度 (平成 16 年度)

プレゼンス情報を利用した
アプリケーションのフレームワークに関する研究

慶應義塾大学 環境情報学部

氏名：谷 隆三郎

指導教員

慶應義塾大学 環境情報学部

村井 純

徳田 英幸

楠本 博之

中村 修

南 政樹

平成 16 年 12 月 29 日

卒業論文要旨 2004 年度 (平成 16 年度)

プレゼンス情報を利用した
アプリケーションのフレームワークに関する研究

個体認識技術を利用することによって実空間の情報を取得することが可能である。本研究では、この実空間情報を利用するアプリケーションへ統一されたインターフェースを提供するためのミドルウェアを提案する。

近年の非接触個体認識技術の進歩により、計算機資源を持たない個体の自動認識が可能になった。この技術は、あらゆるモノに ID を持つタグを取り付けることによって、個体を認識する。そして、ID とデータベースに存在する情報を結びつけることによって、モノに関する情報や位置情報などを取得することが可能になる。こうした技術の登場によって、実空間における個体の情報を計算機上で扱うことが可能になった。

従来の実空間情報を利用するアプリケーションでは、利用するハードウェアの特性を考慮する必要があった。しかし、個体認識技術には用途に応じて複数の異なる種類のハードウェアが存在する。そのため、どのハードウェアにも対応したアプリケーションを構築することが困難であった。

また同じハードウェアを利用して、複数の開発者が、同時に異なるアプリケーションを動作させる場合、アプリケーションによっては不必要な情報を数多く取得してしまう。アプリケーションには、全ての情報が送信されるため、必要としている情報のみを取得することは困難であった。

そこで本研究では、複数のアプリケーション開発者が複数の異なる種類のハードウェアを利用する環境を想定して、これらの問題点を解決するためのミドルウェアを提案する。そして、実空間情報を利用したアプリケーションへ統一されたインターフェースを提供する。

キーワード
1. RFID, 2. ミドルウェア, 3. 実空間ネットワーク, 4. API

慶應義塾大学 環境情報学部

谷 隆三郎

Abstract of Bachelor's Thesis

Academic Year 2004

It is possible by using individual recognition technology to acquire the information on real space. In this research, the middleware for offering the interface unified into the application using this real space information is proposed.

By progress of non-contacting individual recognition technology in recent years, automatic recognition of an individual without computer resources was attained. This technology recognizes an individual by attaching the tag which has ID in all object. And it becomes possible to acquire information, position information, etc. about object by connecting the information which exists in a database to ID. It became possible to treat the information on the individual in real space on a computer by the appearance of such technology.

The characteristic of the hardware to be used needed to be taken into consideration in the application using the conventional real space information. However, the hardware of the kind from which plurality differs according to a use exists in individual recognition technology. Therefore, it was difficult to build the application also corresponding to which hardware.

When two or more developers operate simultaneously different application using the same hardware, many unnecessary information will be acquired depending on application. Since all information was transmitted to application, it was difficult for it to acquire only the needed information.

In this research, the middleware for solving these problems is proposed supposing the environment where two or more application developers use the hardware of the kind from which plurality differs. And the interface unified into the application using real space information is offered.

Keywords :
1. RFID, 2. middleware, 3. Realspace Network 4. API

Keio University, Faculty of Environmental Information

Ryuzaburo Tani

目 次

図 目 次

表 目 次

第1章　序論

本章では、本研究の背景、及び本研究の目的について述べる。また、本論文の構成について述べる。

1.1　背景

1.1.1　ユビキタスコンピューティング環境

ユビキタスコンピューティングという言葉が普及し、様々な分野で応用・研究が進められている。ユビキタスコンピューティングとは、日常のあらゆる場所へコンピュータ及びネットワークが浸透し、利用者が意識することなく、いつでも自由に利用できる環境のことである。

このユビキタスコンピューティング環境が実現すると、センサーを用いて我々の身の回りにある情報を取得することが可能になる。例えば自分の周囲の温度、湿度といった情報や自分の位置情報や行動履歴などの情報である。

これらのセンサーから収集された情報はネットワークを利用することによって、物理的な位置にとらわれることなく、いつでもどこからでも利用することが可能な情報となる。

こうして取得された情報を利用することで自分の行動パターンに応じて予測行動の補助、利用者が今必要としている情報の提供など、利用者が意識的にコンピュータを扱わなくても、コンピュータの方から人間の行動をサポートするよう動作させることが可能になると考えられている。

こうしたアプリケーションは、実空間アプリケーションやコンテクストアウェア・アプリケーションと呼ばれている。小型センサなどを用いて取得された情報から実空間の状況を認識し、その状況に合わせて適切な動作を選択し、実行する。

1.1.2　個体認識技術

個体認識技術とは、モノに ID をつけることによって、その ID からどのようなモノなのかという情報を引き出すための技術である。現在、主に利用されている例としてはバーコードがあり、物流・商店での在庫管理、ポイントカードなどに利用されている。

こうした個体認識技術も無線技術の発達によって RFID と呼ばれる非接触個体認識技術が登場した。RFID とは、IC チップとアンテナを内蔵した RFID タグと、読み取り装置である RFID

リーダから構成されており、RFID リーダを利用することで RFID タグに格納された ID を得るという技術である。

RFID は用途に応じて様々な種類が存在する。例えば入場者をチェックするためのゲート型のリーダ既存のバーコードのような使い方を想定して小型化されたポータブル型ドアの鍵などに利用される取り付け型などが存在する。

またタグにも電池を内蔵するタイプと内蔵しないタイプが存在する。RFID には電池を内蔵したアクティブタグと呼ばれる種類が存在し、数十 m の読み取り範囲を備える。他にはパッシブタグと呼ばれる種類もあり、電池を必要としないが読み取り範囲が数十 cm〜数 m である。

この RFID を利用して、個体の位置情報を取得することが可能である。リーダは通常、検出範囲内に存在するタグの ID を読み取り、特定のコンピュータへ送信するという機能を持つ。この時、リーダに位置情報を表す ID を持たせ、読み取られたタグの ID と同時に送信する。そうすることで、どのリーダから読み取られたのかがわかり、その ID から位置情報を取得することが可能になるという仕組みである。

1.2 本研究の目的

本研究では、RFID システムを用いて、実空間アプリケーションを開発する際に考えられる問題点を解決し、実空間アプリケーションへ統一されたインターフェースを提供するためのミドルウェアの構築を目的とする。

従来の実空間アプリケーション開発環境では、リーダを直接制御することによって情報を取得、加工してアプリケーションが開発されてきた。

しかし、このような環境では、異なる種類、異なるベンダのリーダを利用する場合には、各アプリケーションがそれぞれのリーダに対応しなければならない、という問題があった。

また、どのアプリケーションでも共通して持つ機能が存在し、それらの機能をアプリケーションごとに実装することが、開発にかかるコスト増加の原因となっていた。

そこで、本研究では、これらの問題を解決するために、リーダとアプリケーションの間を取り持つミドルウェアを提案し、その役割について述べる。

1.3 論文の構成

本論文は 6章から構成される。第 2章では、実空間アプリケーションを開発する際の想定環境と問題点について述べ、本研究のアプローチを述べる。第 3章では、問題点を解決するモデルについて述べ、そのモデルに基づいたシステムの設計を述べる。第 4章では、本研究で提案するミドルウェアの具体的な実装について述べる。第 5章では、実装したミドルウェアの動作、及び有用性について検証する。第 6章では、まとめと今後の課題を挙げ、本論文の結論とする。

第2章　要件の整理

本章では、本システムが想定している前提条件について述べ、その想定環境における問題点について述べる。また、それらの問題に関連する既存研究を挙げ、本研究におけるアプローチについて述べる。

2.1　個体認識技術を利用するアプリケーション

本研究では、想定するアプリケーションを、位置情報を利用するアプリケーション、特定の範囲内の情報を利用するアプリケーション、複数のモノを関連付けるアプリケーション、特定のイベントにより動作するアプリケーションの 4 種類に分類する。

2.1.1　位置情報を利用するアプリケーション

位置情報を利用するアプリケーションとは、タグがどのリーダによって検出されているかという情報を利用した、アプリケーションである。
具体的なアプリケーションとしては、紛失したモノがどこにあるのかを発見するアプリケーションや、知り合いや友人がどこにいるのかを見つけるアプリケーションなどが挙げられる。

2.1.2　特定の範囲内の情報を利用するアプリケーション

特定の範囲内の情報とは、部屋に誰がいるのか、何があるのか、などようなの情報のことである。
特定の範囲内の情報を利用するアプリケーションとは、特定のリーダがどのようなタグを検出しているのかという情報を利用した、アプリケーションである。
具体的なアプリケーションとしては、物品管理を行うアプリケーションや、出席管理・勤怠管理を行うアプリケーションなどが挙げられる。

2.1.3　複数のモノを関連付けるアプリケーション

「複数のモノの関連付け」とは、それぞれのモノの間にどのような関係が成り立っているのかを定義することである。

複数のモノを関連付けるアプリケーションとは、複数のタグの検出状態が定義された関係と一致しているかを判定して動作するアプリケーションである。

具体的なアプリケーションとしては、忘れ物検出アプリケーションなどが挙げられる。例えば、自分自身と自分の鞄は常に同じ場所に存在しているという関係を定義したとする。そこで、もし自分が部屋から出て行って検出されなくなり、この関係が成り立たなくなった場合は、鞄を置き忘れている状態であると通知する。

2.1.4 特定のイベントにより動作するアプリケーション

特定のイベントにより動作するアプリケーションとは、特定のタグが特定のリーダによって検出された場合に動作するアプリケーションである。

具体的なアプリケーションとしては、立ち入り禁止区域に人が入った場合にアラートを送信するアプリケーションや、知人が約束の場所へやってきた場合に通知するアプリケーションなどが挙げられる。

2.2 ハードウェアの機能要件

本研究では、情報取得デバイスとして RFID システムを使用することを想定している。しかし、様々な機能を持つハードウェアが存在するため、前提として、以下のような機能要件を満たすものを対象とする。

- タグとリーダによって構成されている。
- タグは一意の識別子を保持しており、その識別子によって個体を識別することができる。
- タグの持つ識別子はリーダによって読み取ることが可能である。
- リーダは自身の個体を識別するための情報を保持している。
- リーダは設置された場所に固定されている。
- リーダはネットワークを利用して情報を送信できる。

2.3 利用モデル

本研究で想定しているアプリケーションとハードウェアの利用モデルについて述べる。

本研究は、施設や建物の全体をカバーできるシステムを想定している。

本システム内で動作する全てのリーダはアプリケーションによって共有される。そのため、同じタグが利用できるならば、他の人によってリーダが設置されている場所へ、自分で新たにリーダを設置する必要はない。既に設置されているリーダを利用して情報を取得することができる。

また、まだ設置されていない場所で、情報を取得する必要がある場合は、その場所の環境や用途に応じて自由にリーダを設置することが可能である。タグに関しても同様に、情報を取得したい人や物に自由に取り付けることができる。

アプリケーションからみた場合も、リーダは共有の情報取得デバイスとして提供される。アプリケーションの種類やリーダの利用者によって制限されることはなく、全てのアプリケーションに対して同じ様に情報が配信される。そして、同じシステム内で動作しているリーダならば、どのリーダからでも自由に情報を取得できる。

そのため、既に十分なリーダが設置されている環境では、タグさえ持っていれば自由にアプリケーションを開発することが可能である。

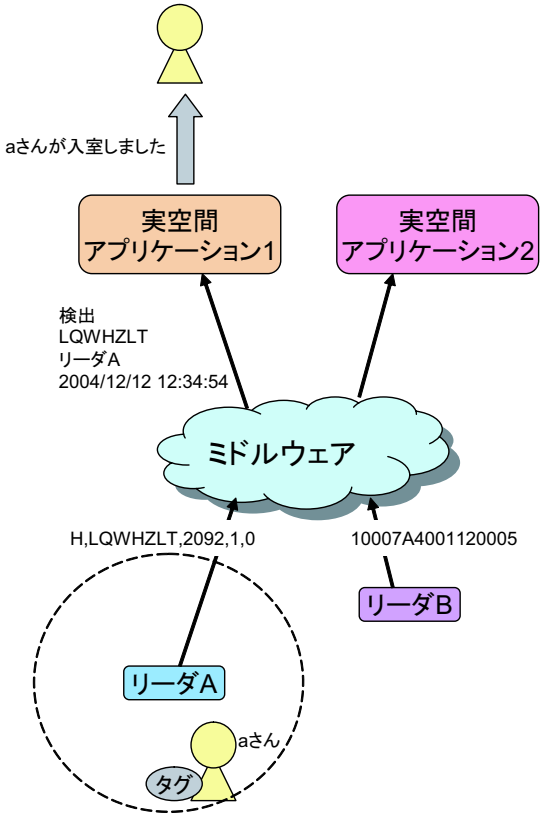

図 2.1: 利用モデルの例

図 2.1は、本システムを利用したアプリケーションとリーダハードウェアの例である。タグを持った a さんが、リーダ A の検出範囲内に入るとリーダ A からは検出した a さんのタグ ID が出力される。この時出力される情報のフォーマットは、リーダ独自のフォーマットである。もう一方のリーダ B が出力するフォーマットとは全く異なっているが、リーダはフォーマットの違いを考慮する必要はない。

リーダから出力された情報は、本システムを通り、a さんの情報を必要としている実空間情報アプリケーション 1 のみに提供される。この時のフォーマットは、どのリーダから読み取られた情報でも統一されたフォーマットで出力される。そのため、実空間アプリケーションは、リーダの種類を考慮する必要はない。また、a さんの情報を必要としていない実空間アプリケー

ション 2 には、情報が配信されない。
本システムから情報を受け取った実空間アプリケーション 1 は、その内容から a さんが入室したという情報を利用者へ配信することが可能になる。
以上のように、本システムを利用することによって、リーダに求められる機能やアプリケーションに必要な機能を大幅に軽減することが可能になる。

2.4 アプリケーション構築の際の問題点

前節で述べた利用モデルを元に、想定するアプリケーションを開発する場合、以下のような問題が考えられる。

- リーダによっては、必要となる情報を出力できない。
- リーダによって出力される情報のフォーマットが異なる。
- リーダによって情報を出力するタイミングが異なる。
- アプリケーションによって必要としている情報が異なる。
- アプリケーションによって情報を必要とするタイミングが異なる。

以上のような問題をアプリケーションごとに解決しようとすると、開発にかかるコストが非常に大きくなるという問題が発生する。
以下にそれぞれの問題点に関する詳細について述べる。

2.4.1 必要な情報の欠落

前節で述べたハードウェアの機能要件の中に、リーダの個体を識別するための情報を保持していること、という項目が存在する。この情報は、リーダの位置情報を示す位置情報ラベルを取得する際に用いられるため、不足しているとタグがどこで検出されたのかがわからない。その結果、想定するアプリケーションを構築することが不可能になるので、この情報は必要不可欠である。
しかし、複数の種類、ベンダのリーダが存在する環境においては、全てのリーダが必ずしもリーダ識別情報を保持しているとは限らない、もしくは出力できるとは限らない。そのため、必要な情報が欠落しているリーダは、本システム内で利用することができないという問題が発生する。

2.4.2 データフォーマット

種類やベンダの異なるリーダでは、それぞれのリーダによって出力される情報のフォーマットが異なる場合がある。このようなリーダが混在する環境では、すべてのリーダに対応したア

プリケーションを開発することは非常に困難である。そのため、特定のリーダに依存したアプリケーションが開発され、対応しないリーダで構築されたシステム上では、動作しないという問題が発生する。また、もし全てのリーダに対応したアプリケーションを開発したとしても、新しいリーダを追加した場合には、全てのアプリケーションをそのリーダに対応するように開発しなおさなければならないという問題が発生する。

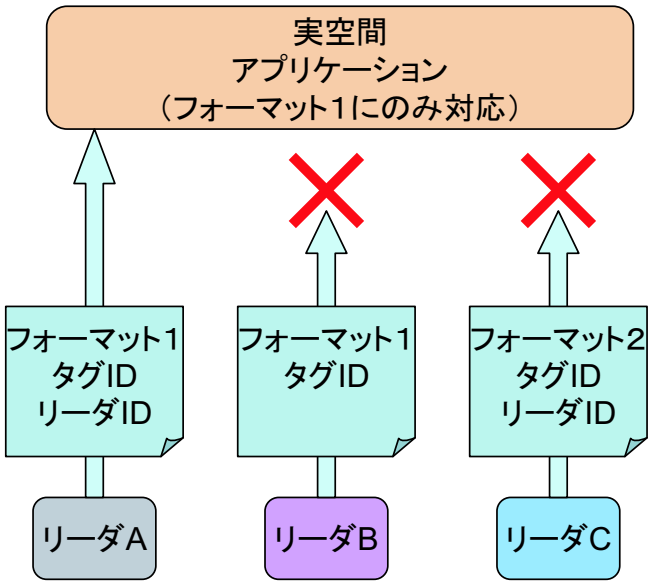

図 2.2: 必要な情報の欠落とデータフォーマットの問題

2.4.3 情報の出力タイミング

種類やベンダの異なるリーダでは、それぞれのリーダによって情報を出力するタイミングが異なる場合がある。例えば、タグが検出・消失した時のみ出力するリーダや検出範囲内にあるうちは継続的に出力するリーダなどが存在する。

このようなリーダが混在する環境では、利用しているリーダがどのタイプのリーダなのかを認識していなければアプリケーションを構築することが困難である。また、複数のタイプを利用する場合には、それぞれのタイプに応じた処理を行わなければならない。そのため、それらのタイプに対応したアプリケーションを構築する必要がある。

しかし、それぞれのアプリケーションごとに対応していると、アプリケーションの開発にかかるコストが増大するという問題が発生する。また、新しいタイプのリーダが追加された場合には、全てのアプリケーションに影響を及ぼすという問題が発生する。

2.4.4 不必要な情報

同じシステム内に様々な種類のアプリケーションが存在する場合、それぞれのアプリケーションは用途が異なるため、必要としている情報も異なると考えられる。しかし、多くのリーダが共有されている環境では、全てのリーダから出力された情報が取得されるため、アプリケーションによっては不必要な情報の量も多い。不必要な情報が取得される場合には、情報を取得する度にアプリケーションで情報の取捨選択を行う必要があるため、多くの無駄な処理が必要になるという問題が発生する。

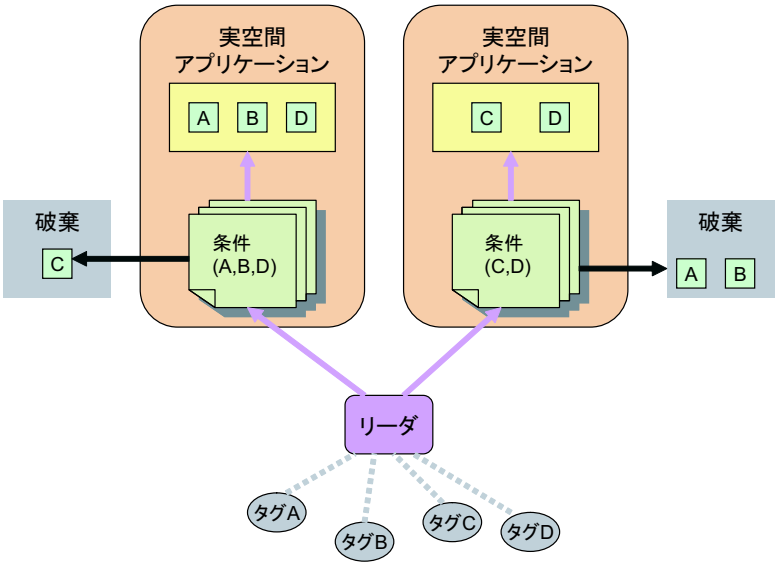

図 2.3: 必要な情報が異なる問題

2.4.5 情報の取得タイミング

複数のアプリケーションが存在する場合、それぞれのアプリケーションは、情報を必要とするタイミングが異なると考えられる。また、アプリケーションが情報を必要とするタイミングは、リーダから情報が出力されるタイミングと一致するとは限らない。そのため、アプリケーションが情報を必要とした場合にいつでも提供できるよう、リーダから出力される情報を保持しておかなければならない。

2.5 関連研究

本節では、RFID 技術を用いたアプリケーションを開発する際に必要な機能をミドルウェアとして提供している、既存研究について述べる。

2.5.1 Sun EPC Event Manager

Sun EPC Event Manager は Auto-ID Center Software Action Group (SAG) の提案する Savant version 1.0 に基づいて設計されている。[1] そして、大規模なアドレス空間における機能性と将来的な拡張性を提供する。

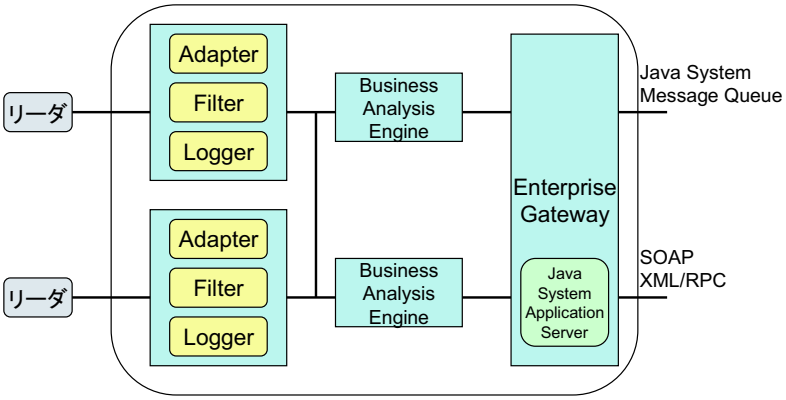

図 2.4: Sun EPC Event Manager

Sun EPC Event Manager は Device Adapter、Filter、Logger、Enterprise Gateway の 4 つのコンポーネントから構成されている。

- Device Adapter
  RFID やバーコードリーダなどの様々なメーカで作られたデバイスは、この層によって Event Manager と通信する。

- Filter
  タグが付けられたオブジェクトによって絶えず生成されたデータから、有用なデータを判読する機能を提供する。標準のフィルタは、イベントの補整、イベントのグループ化、イベントの変化、イベントの許可／拒否を提供する。

- Logger
  RFID および非 RFID のイベントデータを外部システムに通知する。Sun EPC Event Manager は、ファイル・システム、JMS キュー、XML、http、SOAP メッセージのいずれかを利用して情報を記録する。

- Enterprise Gateway
  アプリケーションが EPC Event Manager からデータを取得するための共通インターフェースを定義している。

Sun EPC Event Manager は、既存の Supply Chain Management (SCM) システムへ、スムーズに RFID を取り入れることを目標に設計されている。そのため、他のシステムとの結合という観点から、データの出力形式には、様々なフォーマットをサポートしている。また、データの集約を行う機能をモジュールとして提供し、必要に応じて利用できるようになっている。

2.5.2 IBM RFID Premises Server

IBM RFID Premises Server は、RFID システムをビジネスのプロセスに統合するためのプラットフォームを提供する。このシステムを利用することで、既存インフラストラクチャーを活用したり、さまざまなアプリケーションと統合したりすることが可能になる。また、サーバー上に大量のトラフィックを持つ倉庫、流通センター、店舗、製造工場をターゲットとしている。

IBM RFID Premises Server は、RFID Device Infrastructure とバックエンド統合サーバー間の中継をし、以下のような機能を提供する。

- RFID Device Infrastructure から取得した RFID 情報を解釈する。
- RFID イベントをフィルタリング、集約、モニター、およびエスカレートして、重要なビジネス運営イベントを検出したり、物理オブジェクトの位置を追跡する際の支援を行う。
- データベース機能を含む、ローカル・データのストレージおよびキャッシング機能を提供する。
- 自動的な運用上の意思決定を可能にするビジネス・コンテキストを作成する。
- メッセージの確実な配信を実行できるよう支援する。

IBM RFID Premises Server は、Sun と同じく既存の SCM システムとの統合を目的としており、トレーサビリティの支援など物品の流通過程において必要となる機能を数多く持つという特徴がある。また、システム運用の観点から自動的な意思決定を行う機能をビジネス・コンテキストを保持することで実現している。

2.5.3 ObjectStore RFID Accelerator

ObjectStore RFID Accelerator は、RFID によって生成されるイベントストリームデータを、リアルタイムで処理・管理するために設計された Object Store Event Engine の実装である。

ObjectStore Event Engine は、膨大なイベントストリームの永続データを管理するとともに、毎秒数万件のイベント処理能力が実証された強力なインメモリキャッシングおよび仮想メモリアーキテクチャを提供する。

ObjectStore RFID Accelerator は XML 識別子を使ってイベントを定義しており、一意のタグ ID、タグリーダー名または ID、イベントタイプ (EPC が規定した「Sight」(リーダのフィールド内) または「Unsight」(リーダーのフィールド外)) およびイベントの日付／時間の 4 つの属性を保持している。

開発者は、XML、C++または Java を活用し、標準の定義済みプロセッサ、およびカスタムのアプリケーション専用プロセッサを使ってイベントパイプラインを構築することが可能になっている。

RFID Accelerator を利用する全てのアプリケーションに対して、入力、処理、出力操作を提供する

- 入力:ALE の収集
  タグリーダの調整と管理を行うアプリケーションレベルイベント (ALE) ミドルウェアから RFID イベントストリームデータを収集する。データ収集を効率化するために、EPCglobal の標準に準拠している。また、非 EPC 準拠のイベントデータ (ISO など) のために、基盤となる Event Engine のインフラは、カスタムコレクションアダプタを開発できる開発ツールを提供する。

- 処理:EPC クエリ
  システムが処理する RFID イベントから重要なデータを抽出する。この作業を支援するために、ObjectStore は履歴および現在の RFID イベントデータのフィルタ、グループ化、カウントを行うクエリー／レポート機能を提供する。

- 出力:JMS のためのイベントキャッシュ
  RFID Accelerator は、イベントベースの情報を各アプリケーションに配信するため、SonicMQ JMS メッセージブローカのための EventCache を実装している。この Event Cache により、クエリーの実行結果を JMS メッセージとして取得することが可能になっている。

ObjectStore RFID Accelerator は、RFID システムから取得されるデータを高速に処理できるという特徴がある。RFID システムは膨大な量のデータ処理を行う必要があるため、リーダのイベントを高速処理する機能をインメモリキャッシングすることによって実現している。

2.5.4 TAVIS

TAVIS は RF Code 社が開発したデータ収集基盤システムである。

TAVIS のシステム概要を図 2.5に示す。

TAVIS は、個体認識技術における統合的なデータ収集基盤システムとして開発されている。利用可能なデバイスは、RFID だけではなく GPS(Global Positioning System)、バーコードなどの技術も利用可能である。それぞれのデバイスから取得される情報の違いは、共通の XML フォーマットデータによって、吸収している。

また TAVIS は、より高度な開発環境を実現するために、デバイスの動作を制御するためのインターフェースを用意している。デバイスの動作を制御することで、デバイスの起動・停止、設定の変更などが可能になり、より高度なアプリケーションの開発が可能になっている。

デバイスから取得された情報は、Device Controller によって解析され、Logging/Real-Time access モジュール及び SQL Database モジュールへと格納される。そして、アプリケーションからは、専用の API を用いて取得することが可能である。

- Device Controller
  Device Controller は、デバイスから取得した XML フォーマットのデータを解析し、Logging/Real-time Access モジュール及び、SQL Database モジュールへ送信する。また、要求に応じたデバイスの制御を行うことが可能である。

- Logging/Real-time Access
  Logging/Real-Time access モジュールは、タグの情報を保持して、移動などのイベントを検出する。また、取得した情報を元に現在の状態をリアルタイムモニタリングする。

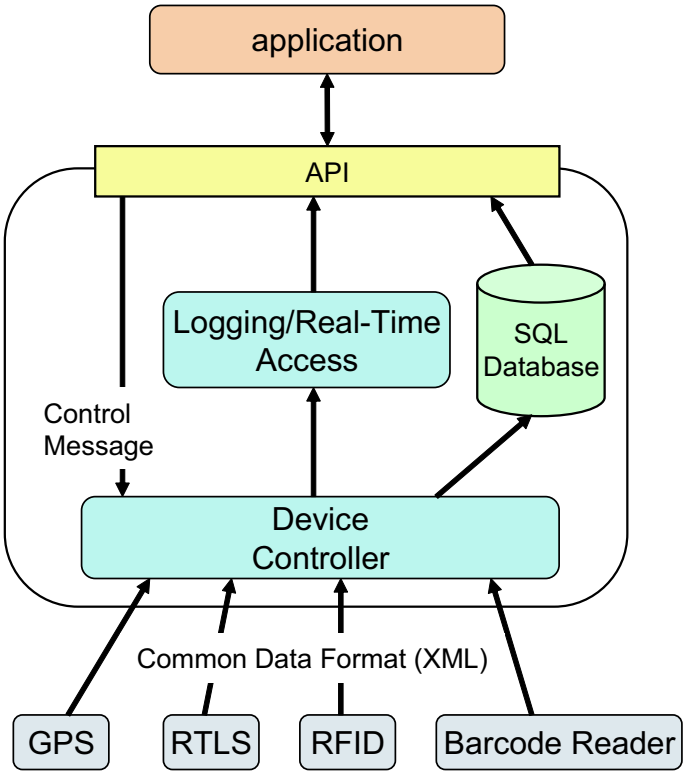

図 2.5: TAVIS

- SQL Database
  SQL Database は、デバイスから取得された情報を継続的に記録する機能を提供する。また、蓄積された情報を、リクエストに応じてアプリケーションへ提供したり、必要に応じてレポートとして出力する。

TAVIS は、アプリケーションからデバイスの管理ができるという特徴をもっている。リーダの一括管理、動作制御などを行うために、リーダへ制御コマンドを送信する機能を API が提供している。

2.6 アプローチ

本節では、2.4節で述べた問題点をふまえた上で、それらの問題を解決するための手法と機能要件ついて述べる。

2.4節で述べた問題点から、本システムでは、複数のリーダから取得された情報を全て同じリーダから取得された情報のように扱わなければならない。さらに、アプリケーションの要求に応じた処理や選別をすることでアプリケーションへ最適な形で情報を提供する、という要求事項が考えられる。

そこで本システムでは、以下の 3 種類の情報を定義する。

- リーダイベント
  どのリーダから出力された情報でも、本システム内で同じように利用するために定義した共通のデータフォーマットである。

- 実空間情報
  現在検出されているタグの ID と検出された場所や時間を表す情報である。この情報からタグがどのような状態なのかを知ることができる。リーダから取得した情報を元に本システムの内部で保持される。

- プレゼンス情報
  タグの検出・消失などのイベントを表す情報である。アプリケーションからのリクエストに応じて生成される。

本システムでは、これらの情報を適切に生成・管理・提供することによって、システムの要求事項を満たす機能を提供する。
以下に、本システムの機能要件を挙げる。

- リーダイベントの変換
- 実空間情報の管理
- プレゼンス情報の生成
- 条件一致判定

リーダイベントの変換機能は、必要な情報の欠落及びデータフォーマットの問題を解決するために、IP アドレスもしくは MAC アドレスを利用して不足している情報を補い、共通のリーダイベントへ変換する機能を提供する。
実空間情報の管理機能は、情報の出力タイミングが異なる問題を解決するために、リーダイベントを読み取ってタグの情報を保持する機能を提供する。
プレゼンス情報の生成機能は、情報の取得タイミングが異なる問題を解決するために、設定情報を保持し、その情報を元に検出・消失を判定する機能を提供する。
条件一致判定機能は、不必要な情報が取得される問題を解決するために、必要な情報の条件を保持し、フィルタリングする機能を提供する。
以下に各機能の詳細について述べる。

2.6.1 リーダイベントの変換

リーダイベント変換機能では、必要な情報の欠落の問題及びデータフォーマットの問題の 2 点を解決する。必要な情報の欠落の問題とは、識別情報を持たないリーダを利用することができないという問題である。データフォーマットの問題とは、未対応のフォーマットで出力を利用することができないという問題である。

必要な情報の欠落問題の解決手法

必要な情報を出力できない問題については、どのような情報が不足しているかがはっきりしている。本システムで扱う情報はタグの ID 情報及び、リーダの識別情報である。この 2 つの内、タグの ID 情報はリーダとして最低限の機能であるために、必ず保持していると考えられる。そのため、この問題はリーダが自身の個体を識別するための情報を保持しているかどうか、という問題であるといえる。

リーダの識別情報は、取得したタグの ID 情報がどのリーダによって検出されたかを示す情報である。この識別情報からリーダの設置されている位置情報を示す位置情報ラベルを取得することによって、タグがどこで検出されたかということを理解することができる。

識別情報を保持することができないリーダをサポートするためには、識別情報の代わりに別の付加情報を用いて位置情報ラベルへ変換するという方法が考えられる。そこで、本システムでは、IP アドレスもしくは MAC アドレスを用いてリーダの個体を識別し、位置情報ラベルへ変換することによってこの問題を解決する。

各所へ設置されたリーダは、ネットワークを利用してサーバへ情報を送信することを前提としている。シリアルインターフェースしか持たないリーダの場合には、シリアルサーバ等を用いてネットワークへ情報を送信できるようにすることができる。そのため、IP アドレスや MAC アドレスの情報は必ず保持していると考えられる。

データフォーマット問題の解決手法

異なる種類のリーダでは情報のフォーマットも異なるという問題の一つの解決方法は、リーダから出力される情報のフォーマットが標準化され、全てのリーダが標準フォーマットへ準拠すればこの問題は解決する。しかし、現時点では、まだそのような規格は標準化されておらず、リーダの種類や各ベンダによって異なるフォーマットで出力される。また、もし標準化されてもそれまでに作られたリーダには対応しなくなってしまう。

そこで、本システムでは、システム内で扱うための共通フォーマットを用意する。この共通フォーマットをリーダイベントと呼ぶ。そして、リーダから取得された情報をリーダイベントへ変換することによって、異なる種類のリーダから取得された情報を同じように扱うことを可能にする。

この手法を用いても、各リーダに対応した変換モジュールが必要になる。しかし、このモジュール群を階層化モデルの中に取り入れることによって、アプリケーションからは隠蔽化することが可能になり、それぞれのアプリケーションが各フォーマットへ対応する必要がなくなる。

2.6.2 実空間情報の管理

実空間情報の管理では、情報の取得タイミングが異なる問題を解決する。情報を必要とするタイミングが異なる問題とは、アプリケーションが情報を必要とするタイミングが異なるために、いつでも提供できるよう、リーダから出力された情報を保持する必要があるという問題である。

この問題を解決するために、リーダから出力された情報を保持する機能を、アプリケーションから切り離し、本システムで提供するこの管理機能は、リーダから出力される情報を解読して、タグの現在の状態を保持する機能である。

また、この管理機能をアプリケーションから切り離すことによってアプリケーションの開発コストを大幅に軽減している。

2.6.3 プレゼンス情報の生成

プレゼンス情報の生成機能では、情報の出力タイミングが異なる問題を解決する。タイミングが異なる問題とは、リーダが情報を出力するタイミングによって、異なる処理を行わなければならない問題である。

リーダが情報を出力するタイミングは、2通りのタイプが存在する。タグの検出・消失といったイベントが発生したときのみに出力するタイプと、タグが検出されている間は継続的に出力するタイプである。前者はリーダから出力される情報が、直接検出・消失を表しているため、特別な処理をする必要はない。一方、継続的に出力するタイプのリーダでは、検出・消失を判定するために、新しく検出されたタグかどうか、情報がこなくなったかどうかを検出しなければならない。この時、検出・消失と判定するまでの回数や時間といったパラメータがアプリケーションごとに異なることが考えられる。

そこで、本システムでは、アプリケーションごとに検出・消失と判定するための設定情報を保持する。そして、リーダの出力する情報から正しく検出・消失を判定する機能を提供することによってこの問題を解決する。

2.6.4 条件一致判定

条件一致判定機能では、不必要な情報が取得される問題を解決する。必要な情報が異なる問題とは、不必要な情報が数多く取得されることによって、アプリケーションが多くの無駄な処理を行ってしまうという問題である。

リーダは検出範囲内に存在して読み取り可能なタグの情報を全て出力する。そのため、リーダを共有している環境では、大量の不必要な情報が取得される可能性がある。例えば、自分の所有しているタグの情報のみを取得したい場合などは、取得した全ての情報に対して自分のタグかどうかチェックする機能をアプリケーションで提供しなければならない。フィルタリング機能を持つリーダも存在するが、リーダでフィルタリングをしてしまうと、他のアプリケーションにも影響を及ぼす恐れがある。

そこで、アプリケーションには条件を記述する機能のみを提供して、本システム内であらかじめフィルタリングすることによってこの問題を解決する。

第3章　設計

本章では、2章で議論した機能要件を元に、プレゼンス情報を利用した実空間ミドルウェアの設計について述べる。機能要件整理の結果、本ミドルウェアは3層のレイヤ構造をなしている。本章では、先ず、それぞれのレイヤの役割について述べ、ついで、各レイヤの詳細と動作概要について述べる。

3.1　設計概要

2章では、実空間アプリケーションの構築の機能要件として、リーダイベントの変換、実空間情報管理、プレゼンス情報の生成、条件一致判定が必要不可欠であることを述べた。本研究では、以上の機能をアプリケーションから切り離し、ミドルウェアとして実現する。

本ミドルウェアでは、大きく分けて以下の2つの機能を実現している。

一つは、異種のインタフェースを備えるリーダハードウェアの出力情報を共通のリーダイベント情報へ変換し、それぞれのリーダで検出されたタグの情報を利用可能にする機能である。もう一つは、リーダの出力情報をアプリケーションが要求する情報へ加工し、そのための設定・要求のインタフェースを提供する機能である。

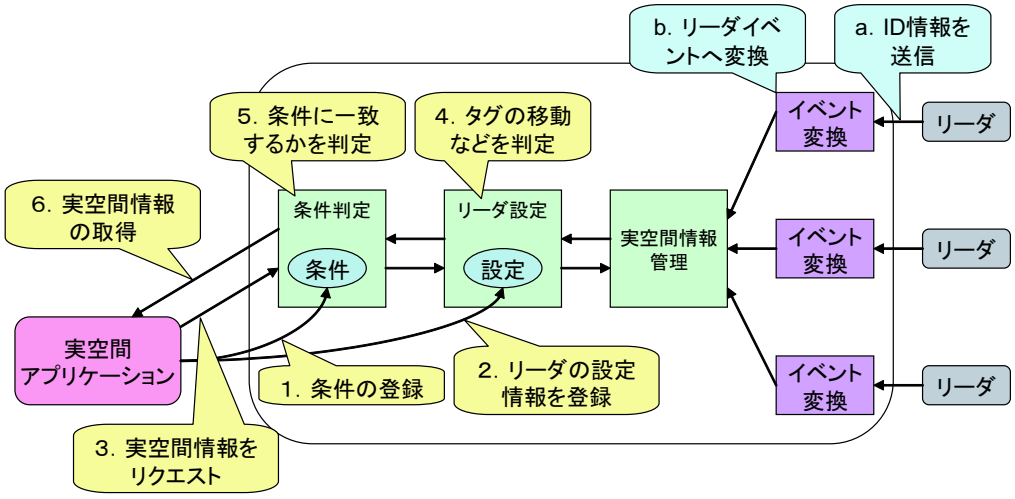

図 3.1: ミドルウェア、リーダおよびアプリケーションの関係

図3.1に、ミドルウェア、リーダおよびアプリケーションの関係図を示す。図中の1〜6は、

アプリケーションがミドルウェアを利用する手順を表す。

アプリケーション毎に、リーダに対する設定と条件の登録を行うことができる。リーダに対する設定を、リーダ設定情報と呼び、検出・消失と判断するまでの時間などを設定できる。また、条件の登録をコンディション情報と呼び、必要としている情報の条件を記述することができる。

アプリケーションが情報取得リクエストを送信することで、実空間情報管理部がリクエストの内容に応じた情報を配信する。この際、リーダ設定部及び条件判定部では、設定された情報に基づいて取得された情報がアプリケーションの必要としている情報かどうかを判定し、条件に一致している情報のみをアプリケーションへ配送する。

また、図中の a〜b は、リーダからアプリケーションへの情報の流れを表している。

それぞれのリーダ・ハードウェアから出力される情報は、ハードウェアの種類毎に異なる。このリーダ・ハードウェアからの情報は、イベント変換部でリーダイベントと呼ばれる共通フォーマットへ変換される。

リーダイベントは、アプリケーションの要求に基づいて設定されたリーダ設定情報を用いて、適切に変換され、目的のアプリケーションに配送される。

これらの機能により、異種のリーダハードウェアが複数存在する環境においても、アプリケーションからそのインタフェースの際を意識することなく利用可能となる。また、個々のアプリケーションは、リーダ設定情報を設定することにより、要求する形での出力情報を取得できる。

3.2 システムの構成

本ミドルウェアは、階層化された以下の 3 つのモジュールによって構成されている。

- リーダイベント層
- プレゼンス層
- コンディション層

リーダイベント層ではリーダイベントデータの変換を、プレゼンス層では実空間情報管理およびプレゼンス情報の生成を、コンディション層では、条件一致判定を実現している。

図 3.2に本ミドルウェアのシステムモデル概要図を示す。以下に各層の役割およびミドルウェア内部で扱われる情報と動作概要について述べる。

3.2.1 リーダイベント層

リーダイベント層は、多種多様なリーダハードウェアの出力情報を本機構共通なリーダイベントへ正規化する役割を持つ。本層は、リーダハードウェアと直接通信するソフトウェアモジュールである。また、個々のリーダハードウェアから取得した出力情報をリーダイベントに変換し、プレゼンス層へ提供する。

リーダイベントモジュールの動作概要を図 3.3に示す。

1. 管理するリーダの個体情報と位置情報ラベルの対応リストを生成する。

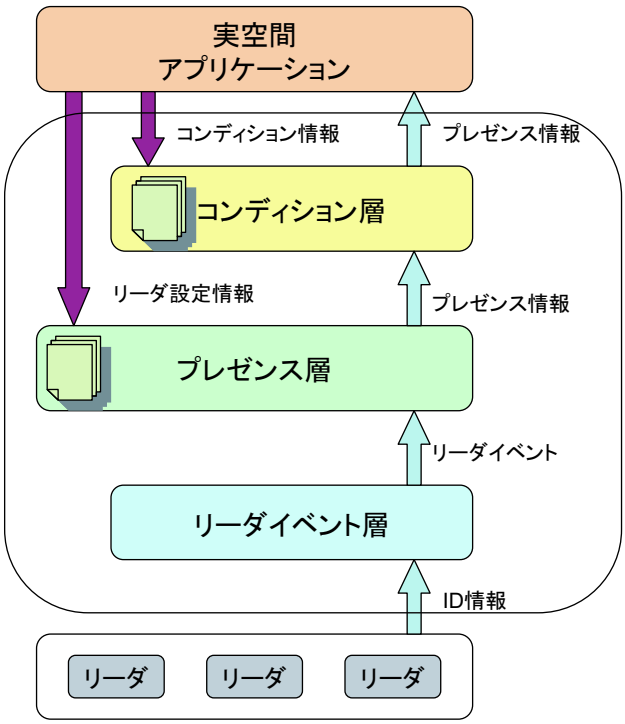

図 3.2: ソフトウェアモデル概要図

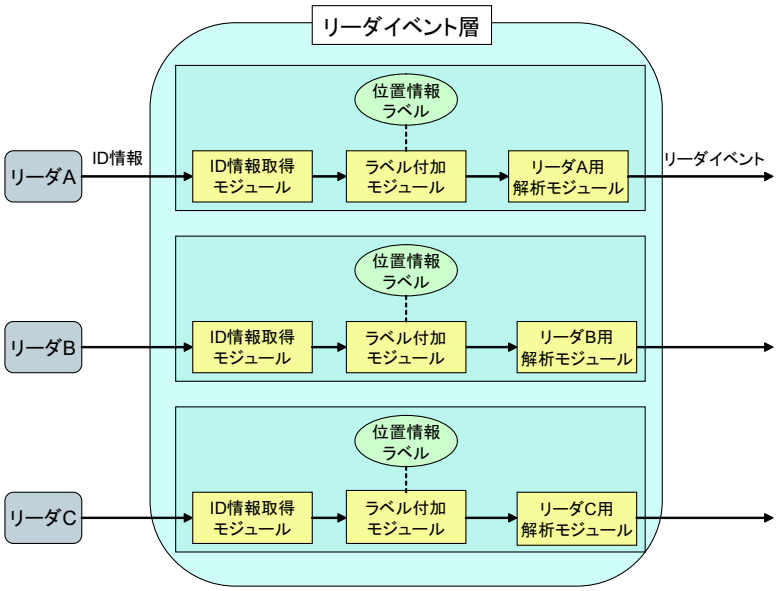

図 3.3: リーダイベント層の動作概要

2. リーダから送信された情報を取得する。

3. 取得した情報のリーダ識別情報から位置情報ラベルを取得する。

4. リーダから取得した情報とリーダの位置情報ラベルからリーダイベントを生成する。

5. 生成されたリーダイベントをプレゼンスモジュールへ送信する。

3.2.2 プレゼンス層

プレゼンス層は、リーダイベント層からリーダイベントを取得し、アプリケーション毎に設定された設定情報を元にプレゼンス情報を生成する役割を持つ。本層は、リーダイベント層からリーダイベントを取得するソフトウェアモジュールである。また、アプリケーション毎の設定情報を保持し、その情報を元にプレゼンス情報を作成し、コンディション層へ提供する。

リーダ設定情報とは、タグの検出・消失を判定する際に用いるパラメータのことであり、検出と判定するまでの回数や消失と判定するまでの時間を設定することができる。本情報は、アプリケーション毎に保持され、リーダイベント層から取得されるリーダイベントを、プレゼンス情報に変換する際の判断と変換の基準と成る。また、リーダ設定情報を設定・変更するためのインターフェースがアプリケーションへ提供される。

プレゼンスモジュールの動作概要を図 3.4に示す。

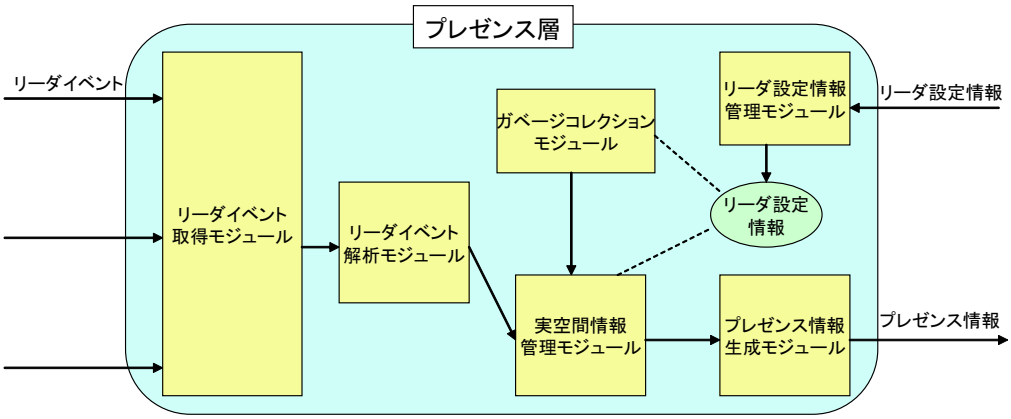

図 3.4: プレゼンス層の動作概要

1. アプリケーションからの要求に応じて、プレゼンス情報を生成するためのリーダ設定情報を生成する。

2. リーダイベント層からリーダイベントを取得する。

3. 取得したリーダイベントを解析し、実空間情報管理モジュールで保持する。

4. プレゼンス情報生成モジュールでは、保持しているリーダ設定情報を元にプレゼンス情報を生成する。

5. 生成されたプレゼンス情報をコンディションモジュールへ送信する。

3.2.3 コンディション層

コンディション層は、プレゼンス情報を、アプリケーション毎に設定されたコンディション設定情報を元に、個々のアプリケーションへ要求する形で配送する役割を持つ。

コンディション設定情報とは、アプリケーションがどのようなプレゼンス情報を必要としているのかを記述する情報である。本情報は、アプリケーション毎に保持され、プレゼンス層から取得されるプレゼンス情報を、個々のアプリケーションへ配送するか否かの判断の基準と成る。また、コンディション設定情報を設定・変更するためのインターフェースもアプリケーションへ提供される。

コンディションモジュールの動作概要を図 3.5に示す。

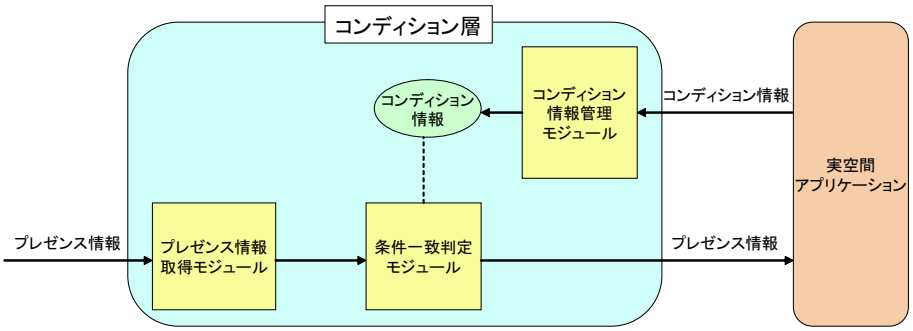

図 3.5: コンディション層の動作概要

1. アプリケーションの要求に応じて、コンディション情報を生成する。
2. プレゼンスモジュールからプレゼンス情報を取得する。
3. 取得したプレゼンス情報が設定されたコンディション情報と一致するかどうかを判断する。
4. 一致していた場合にはアプリケーションへプレゼンス情報を提供する。

第4章　実装

本章では、本研究のミドルウェアおよびミドルウェアを利用するシステムの実装について述べる。

本システムは、複数のリーダハードウェア、ミドルウェアが動作するホスト、ミドルウェアを利用するアプリケーションおよび動作ホストから成る。

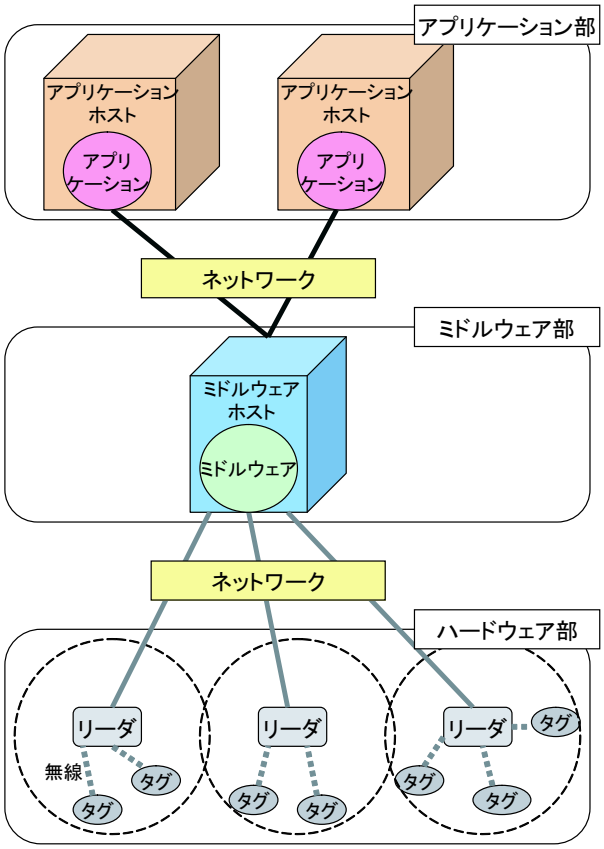

図 4.1: 本システムの全体概要図

図 4.1は、本システムの全体を表した図である。ハードウェア部では、検出範囲内のタグの情報を検出するために RFID システムを使用している。ミドルウェア部では、ネットワークに接続されたホスト上で本ミドルウェアが動作している。アプリケーション部では、同じくネットワークに接続されたホスト上でアプリケーションソフトウェアが動作している。それぞれの部分は、全てネットワークで相互に接続され、通信を行っている。

4.1 ハードウェア部の実装

ハードウェア部の実装は、RFID システムを用いて行った。RFID システムとして、RF CODE 社の SPIDER READER 及び東京特殊電線社 (TOTOKU) の MEGRAS の 2 種類を使用した。本システムで使用した機器を以下の表 4.1に示す。

RFID システム	TOTOKU, MEGRAS
RFID システム	RF CODE, SPIDER READER
シリアルサーバ	Cyclades, Cyclades-TS100

表 4.1: 仕様機器一覧

MEGRAS(図 4.2) は、アクティブ型の RFID システムで、タグには 7 桁の ID 情報が格納されている。発信間隔は約 1 秒で、検出範囲は半径 10m 程度である。また、タグにはスイッチが搭載されており、定期的に発信する電波とは別の信号を送信することが可能である。

リーダの通信インターフェースとしてネットワークインターフェースを搭載しており、本ミドルウェアに対して直接情報を送信することが可能である。

MEGRAS のハードウェア仕様を以下の表 4.2に示す。

図 4.2: MEGRAS

製品名	MEGRAS
周波数	315.000MHz ± 200Hz
通信インターフェース	10/100BASE-TX EtherNet

表 4.2: MEGRAS ハードウェア仕様

SPIDER READER(図 4.3と図 4.4) は、MEGRAS と類似のアクティブ型の RFID システム

で、タグには 7 桁の ID 情報が格納されている。電波の発信間隔は約 1 秒で、検出範囲は半径 20m 程度である。

SPIDER READER のハードウェア仕様を以下の表 4.3に示す。

図 4.3: SPIDER READER

図 4.4: SPIDER READER タグ

製品名	SPIDER READER
周波数	303.825MHz
通信インターフェース	RS232C

表 4.3: SPIDER READER ハードウェア仕様

SPIDER READER の通信インターフェースは RS232C のみとなっているため、ネットワークを利用して情報を収集する本ミドルウェアに対して、直接情報を送信することはできない。そこで、RS232C インターフェースからの入力をネットワークに接続されたサーバへ送信するためのシリアルサーバが必要になる。

以下の図4.5は本ミドルウェアで使用したシリアルサーバのCyclades-TS100である。Cyclades-TS100のハードウェア仕様を以下の表4.4に示す。

図 4.5: Cyclades-TS100

OS	TS LINUX
CPU	MPC855T(PowerPC Dual-CPU)
Memory	16MB SDRAM / 4MB Flash

表 4.4: Cyclades-TS100 ハードウェア仕様

4.2 ミドルウェア部の実装

ミドルウェア部の実装は、C 言語を用いて行った。

本ミドルウェアは、リーダイベント層、プレゼンス層、コンディション層の3つのソフトウェアモジュールから構成されている。

図4.6はそれぞれのソフトウェアモジュールの内容とその関係を表した図である。

リーダイベント層では、リーダの出力情報を取得してリーダイベントへ変換する機能を実装した。

プレゼンス層では、リーダイベントを受信して実空間情報の管理する機能及び、実空間情報からアプリケーションの要求に応じてプレゼンス情報を生成する機能を実装した。

コンディション層では、取得したプレゼンス情報が条件に一致するかを判定し、必要に応じて取捨選択する機能を実装した。

それぞれのソフトウェアモジュールの詳細な実装について以下に述べる。

4.2.1 リーダイベント層の実装

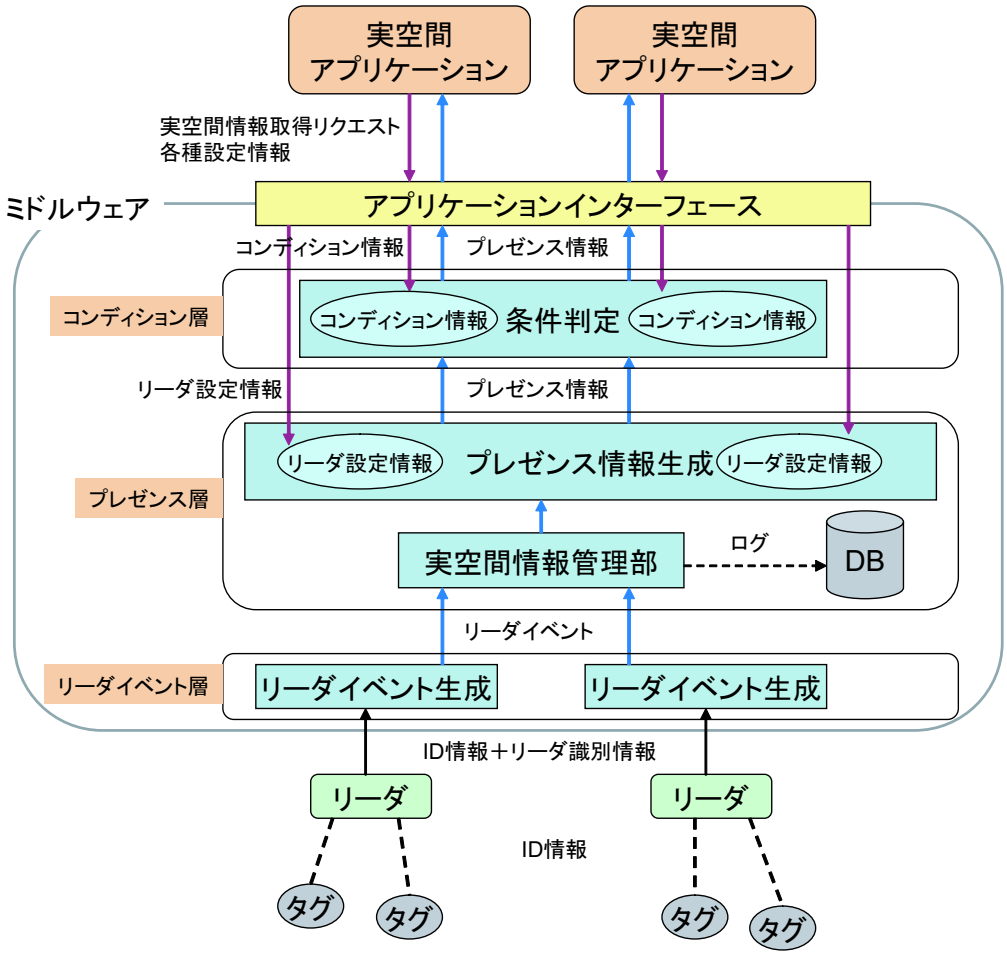

図 4.6: ソフトウェアモジュール概要

リーダイベント層では、リーダの出力情報からリーダの設置場所を表す位置情報ラベルを取得し、付加する機能及び、それらの情報からリーダイベントへ変換する機能を実装した。

位置情報ラベルの付加

リーダの個体識別情報からリーダの位置情報を表す位置情報ラベルを取得してリーダから取得された情報に付加する機能を提供する。

プロトタイプ実装では、リーダの個体識別情報として IP アドレスを利用した。「IP アドレス, 位置情報ラベル」のような CSV 形式の設定ファイルを用意する。この設定ファイルを読み込み、内部に位置情報ラベル変換テーブルを保持する。

リーダから情報を受け取った際の送信元 IP アドレスを調べ、変換テーブルを用いて位置情報ラベルを取得することが可能になっている。

リーダイベント変換

リーダイベント変換機能は、位置情報ラベルを付加したリーダの出力情報をリーダイベントへ変換し、プレゼンス層へ送信する機能を提供する。

本機能で変換するリーダイベントは以下のような構造体である。

```
struct reader_event {
  int flag;
  char tagid[128];
  char location[128];
  long current_time;
  long enter_time;
  int interval_time;
  long elapsed_time;
  long count;
};
```

図 4.7: reader_event 構造体

このリーダイベント変換機能は、リーダの出力情報を解読する必要があるため、リーダの種類ごとに異なる。

以下の図 4.8は MEGRAS の出力する情報のフォーマットである。この情報をリーダイベントへ変換する関数を以下の図 4.9に示す。

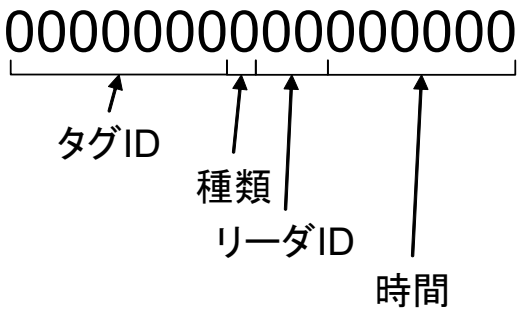

図 4.8: MEGRAS のメッセージフォーマット

4.2.2 プレゼンス層の実装

プレゼンス層では、実空間情報を管理する機能及び、プレゼンス情報を生成する機能を実装した。

```
int create_msg(struct reader_event *msg, char *buf, char *location)
{
  char tmp[32];
  time_t tval;

  if(strlen(buf) != 16){
    return -1;
  }

  memset(msg->tagid, 0, sizeof(msg->tagid));
  strncpy(msg->tagid, buf, 7);

  if(strlen(location) < 128){
    memset(msg->location, 0, sizeof(msg->location));
    strcpy(msg->location, location);
  }

  memset(tmp, 0, sizeof(tmp));
  strncpy(tmp, &buf[7], 1);
  msg->flag = todec(tmp);

  memset(tmp, 0, sizeof(tmp));
  strncpy(tmp, &buf[10], 6);
  msg->elapsed_time = atol(tmp);

  tval = time(NULL);
  msg->current_time = tval;

  return 0;
}
```

図 4.9: MEGRAS create_msg 関数

実空間情報管理機能では、リーダイベントを解読し、検出されているタグの状態を保持する機能及び、ガベージコレクション機能を持つ。

また、プレゼンス情報生成機能では、アプリケーション毎にリーダ設定情報を保持する機能及び、リーダ設定情報に基づいてプレゼンス情報を生成する機能を持つ。

実空間情報の管理

実空間情報の管理機能は、リーダイベントを取得して、検出されている全てのタグの情報を保持する機能を提供する。

検出されているタグの情報は、以下の構造体から構成されたリストとして保持されている。

```
struct listnode {
  struct reader_event *msg;
  struct listnode *next;
  struct listnode *prev;
};
```

図 4.10: reader_event list 構造体

実空間情報の管理機能は、受け取ったリーダイベントを解読して、リストの更新を行う。リーダイベントが検出・消失を示す情報の場合は、その内容に応じてリストから追加・削除を行う。継続的に送信される情報の場合は、新しく検出されたタグかどうかをチェックする。新しいタグの場合は、リストへの追加を行い、既に検出されているタグの場合は、最終検出時間を更新する。さらにこの種類の情報の場合は、タグの消失を示すリーダイベントが発行されないため、ガベージコレクション機能が必要になる。この機能に関する詳細は次節で述べる。

これらの機能によって、リストを常に最新の状態に保っている。

ガベージコレクション

ガベージコレクション機能は、前節で述べた実空間情報管理機能が保持しているリストの中から既に消失したタグを発見し、削除する機能を提供する。

継続的に送信される情報の場合、タグが検出範囲内に入った時は、リーダイベントが発行されるため、検出と判断してリストの追加を行うことが可能である。しかし、タグが検出範囲外に出て行った時は、リーダイベントが発行されなくなるため、リーダイベントのみによって消失したと判断することは不可能である。

そこで、リーダイベントを処理する機能と並行して、リーダイベントが発行されなくても定期的にリストの状態を監視するガベージコレクション機能が必要になる。

ガベージコレクション機能は、リストの中から一定時間以上検出されていないタグの情報を削除する。この時、どのくらいの時間で消失と判断するかは、リーダ設定情報を参照して決定される。

リーダ設定情報の保持

リーダ設定情報は、タグの検出・消失を判断するための情報である。前節で述べたように、消失と判断するまでの時間などを指定することができる。この設定情報は、利用者によって異なる利用方法を想定して、アプリケーションごとに異なる値を設定することが可能である。

プロトタイプ実装では、検出と判定するまでの検出回数及び、消失と判定するまでの時間をリーダ設定情報として保持している。

プレゼンス情報の生成

プレゼンス情報は、タグの検出・消失が発生した場合に、どのようなイベントが発生したかを表す情報である。
プレゼンス情報は、以下の構造体で構成される。

```
struct result {
  int flag;
  char tagid[128];
  char location[128];
  long current;
  long enter;
  struct result *next;
};
```

図 4.11: result 構造体

実空間情報管理機能やガベージコレクション機能によって、管理機能で保持しているリストに変化が生じた場合に、プレゼンス情報を生成してコンディション層へ送信する機能を提供する。

4.2.3 コンディション層の実装

コンディション層では、アプリケーション毎に必要としている情報の条件を保持する機能及び、プレゼンス情報が条件に一致しているかを判断する機能を実装した。

コンディション情報の保持

コンディション情報は、アプリケーションが必要としている情報の条件が記述された情報である。
複数のアプリケーションが存在する場合、それぞれのアプリケーションが必要としている情報は異なるということを想定して、アプリケーション毎に必要な情報の条件を記述できる機能を提供する。
コンディション情報は、以下の構造体から構成されている。

```
struct cond_list{
  int type;
  int flag;
  struct condition *cond;
  struct cond_list *prev;
  struct cond_list *next;
};
```

図 4.12: condition_list 構造体

```
struct condition{
  struct cond_flag *flag_head;
  struct cond_flag *flag_tail;
  struct cond_tagid *tagid_head;
  struct cond_tagid *tagid_tail;
  struct cond_location *location_head;
  struct cond_location *location_tail;
};
```

図 4.13: condition 構造体

```
struct cond_flag{
  int flag;
  struct cond_flag *next;
};
```

図 4.14: cond_flag 構造体

```
struct cond_tagid{
  char *tagid;
  struct cond_tagid *next;
};
```

図 4.15: cond_tagid 構造体

```
struct cond_location{
  char *location;
  struct cond_location *next;
};
```

図 4.16: cond_location 構造体

条件一致判定

条件一致判定機能は、プレゼンス情報がコンディション情報の条件を満たしているかどうかをチェックする機能を提供する。

取得したプレゼンス情報に含まれるそれぞれの要素が、コンディション情報の中に存在するかをチェックする。全ての要素がコンディション情報の中に存在した場合は、条件に一致したとみなして、アプリケーションへプレゼンス情報を送信する。一つでもコンディション情報の中に存在しなかった場合は、条件に一致しないとみなして、そのプレゼンス情報は破棄される。

また、コンディション情報の中に条件が記述されていない要素が存在した場合には、その要素は条件に一致していると判断される。

4.3 アプリケーション部の実装

アプリケーション部は、アプリケーションへ提供する C 言語用ライブラリを実装した。アプリケーションは、本ライブラリを利用することで、コンディション情報の登録、リーダ設定情報の登録、リクエストの送信、結果の取得を行うことが可能になる。

本ライブラリで提供する主な機能について述べる。

4.3.1 ミドルウェアとの接続・切断

ミドルウェアとアプリケーションは、異なるホスト上で動作する別々のソフトウェアモジュールである。それぞれのホストは、ネットワークへ接続されていることを前提としており、この 2 つのソフトウェアモジュールはネットワークを利用して通信を行う。そのため、アプリケーションがミドルウェアから情報を取得する場合には、まず、ミドルウェアとの接続を確立しなければならない。そこで、ミドルウェアとの接続を開始・終了する関数を実装した。

これらの関数を以下の表 4.5表と 4.6に示す。

関数	int RS_connect(const char *hostname, const char *port);
引数	接続するホスト名とポート番号
返り値	成功した場合はソケットディスクリプタ、失敗した場合は-1
説明	ミドルウェアとの接続を開始する

表 4.5: RS_connect 関数

関数	void RS_close(int con);
引数	接続済みのソケットディスクリプタ
返り値	なし
説明	ミドルウェアとの接続を終了する

表 4.6: RS_close 関数

4.3.2 コンディション情報の登録

ミドルウェアの提供する機能に、アプリケーションの必要としている情報を選別する機能が存在する。この機能は、コンディション情報と呼ばれる条件を記述することによって、アプリケーションに必要な情報のみを提供することが可能になる。そこで、コンディション情報を登録するための関数を実装した。

プレゼンス情報の種類

コンディション情報の 1 つ目の要素は、プレゼンス情報の種類である。プレゼンス情報の種類は、今のところ検出・消失・スイッチ押下の 3 種類が存在する。このプレゼンス情報の種類を登録することによって、その種類のプレゼンス情報のみを取得することが可能になる。また、これらの種類はそれぞれ数字に対応しており、その数字を用いることによって必要なプレゼンス情報の種類を登録することが可能になっている。

プレゼンス情報の種類を登録する関数を、以下の表 4.7に示す。

関数	int set_cond_type(int con, int type);
引数	接続済みのソケットディスクリプタとプレゼンス情報の種類
返り値	成功した場合は 0、失敗した場合は-1
説明	コンディション情報にプレゼンス情報の種類を登録する

表 4.7: set_cond_type 関数

タグ ID

コンディション情報の 2 つ目の要素は、タグ ID である。必要としているタグの ID を登録することによって、そのタグの情報のみを取得することが可能になる。

タグ ID を登録する関数を、以下の表 4.8に示す。

関数	int set_cond_tagid(int con, char *tagid);
引数	接続済みのソケットディスクリプタとタグ ID
返り値	成功した場合は 0、失敗した場合は-1
説明	コンディション情報にタグ ID を登録する

表 4.8: set_cond_tagid 関数

位置情報ラベル

コンディション情報の 3 つ目の要素は、位置情報ラベルである。必要としているリーダの位置情報ラベルを登録することによって、そのリーダから検出された情報のみを取得することが可能になる。

位置情報ラベルを登録する関数を、以下の表 4.9に示す。

関数	int set_cond_location(int con, char *location);
引数	接続済みのソケットディスクリプタと位置情報ラベル
返り値	成功した場合は 0、失敗した場合は-1
説明	コンディション情報に位置情報ラベルを登録する

表 4.9: set_cond_location 関数

4.3.3 リーダ設定情報の登録

ミドルウェアの提供する機能に、検出・消失を判断する際のパラメータを変更するという機能が存在する。このパラメータをリーダ設定情報と呼び、消失と判断するまでの時間などを変更することが可能になる。そこで、リーダ設定情報を登録するための関数を実装した。

現在設定可能な情報は、検出と判断するまでの検出回数と消失と判断するまでのタイムアウト時間の 2 種類である。これらの設定を変更するための関数を、以下の表 4.10と表 4.11に示す。

関数	int set_config_count(int con, int count);
引数	接続済みのソケットディスクリプタと回数
返り値	成功した場合は 0、失敗した場合は-1
説明	リーダ設定情報の検出回数を変更する

表 4.10: set_config_count 関数

関数	int set_config_timeout(int con, int timeout);
引数	接続済みのソケットディスクリプタとタイムアウト時間
返り値	成功した場合は 0、失敗した場合は-1
説明	リーダ設定情報のタイムアウト時間を変更する

表 4.11: set_config_timeout 関数

4.3.4 リクエストの送信

ミドルウェアから情報を取得する際に送信するリクエストには 2 つのタイプが存在する。1 つは応答型リクエストで、リクエストを送信すると、すぐに結果が返ってくるタイプのリクエストである。もう一つは通知型リクエストで、あらかじめリクエストを送信しておくと、タグが移動してプレゼンス情報が生成された場合に、その情報を通知してくれるタイプのリクエストである。以下に、これら 2 つのタイプの詳細について述べる。

応答型リクエスト

応答型リクエストは、すぐに結果が返ってくるタイプのリクエストである。このタイプのリクエストは、既に検出されているタグの現在の状態を知りたい場合や、特定のリーダで検出されているタグを知りたい場合などに利用される。

このタイプのリクエストでは、複数の結果が返ってくることが考えられる。そのため、複数の結果を保持することができるデータ型が必要がある。しかし、プレゼンス情報 (図 4.11) には、あらかじめ複数の結果をリスト化するための next フィールドが用意されているため、新しいデータ型を用意する必要はない。複数の結果が返ってくる場合には、結果を連結し、先頭のポインタを返すことで対応できる。また、プレゼンス情報には RS_RESULT という別名をつけた。

応答型のリクエストとして、稼動しているリーダの位置情報ラベルを取得する関数、タグ ID から検出されている位置情報ラベルを取得する関数、位置情報ラベルから検出されているタグ ID を取得する関数を実装した。

これらの関数を、以下の表 4.12、表 4.13、表 4.14に示す。

関数	RS_RESULT *get_reader(int con);
引数	接続済みのソケットディスクリプタ
返り値	成功した場合は結果リストの先頭を指す RS_RESULT 型のポインタ、失敗した場合は NULL
説明	稼動しているリーダの位置情報ラベルを取得する

表 4.12: get_reader 関数

関数	RS_RESULT *get_location(int con, char *tagid);
引数	接続済みのソケットディスクリプタとタグ ID
返り値	成功した場合は結果リストの先頭を指す RS_RESULT 型のポインタ、失敗した場合は NULL
説明	タグ ID から検出されている位置情報ラベルを取得する

表 4.13: get_location 関数

関数	RS_RESULT *get_tagid(int con, char *location);
引数	接続済みのソケットディスクリプタと位置情報ラベル
返り値	成功した場合は結果リストの先頭を指す RS_RESULT 型のポインタ、失敗した場合は NULL
説明	位置情報ラベルから検出されているタグ ID を取得する

表 4.14: get_tagid 関数

通知型リクエスト

通知型リクエストは、プレゼンス情報が生成された場合に、その情報を通知してくれるタイプのリクエストである。このタイプのリクエストは、タグが移動したらすぐにアクションを起こす必要があるアプリケーションやタグの状態を常に監視しているアプリケーションなどで利用される。

このタイプのリクエストでは、取得されるプレゼンス情報は必ず一つであるため、結果をリスト化する必要はない。プレゼンス情報が生成された場合には、接続されたソケットにプレゼンス情報が通知され、その情報を読み取ることによってどのようなイベントが発生したかを知ることができる。

通知型のリクエストとして、プレゼンス情報の通知を開始する関数、通知を停止する関数を実装した。

これらの関数を、以下の表 4.15と表 4.16に示す。

関数	int start_event(int con);
引数	接続済みのソケットディスクリプタ
返り値	成功した場合は 0、失敗した場合は-1
説明	プレゼンス情報の通知を開始する

表 4.15: start_event 関数

関数	int stop_event(int con);
引数	接続済みのソケットディスクリプタ
返り値	成功した場合は 0、失敗した場合は-1
説明	プレゼンス情報の通知を停止する

表 4.16: stop_event 関数

4.4 実装環境・運用環境

本節では、システムの開発において利用した実装環境及び、本システムの動作する運用環境について述べる。

4.4.1 実装環境

本システムを開発するにあたって利用した実装環境を以下の表 4.17に示す。

開発言語	C 言語
OS	FreeBSD 5.3 Release
コンパイラ	gcc 3.4.2

表 4.17: 実装環境

4.4.2 運用環境

運用環境として、本ミドルウェアの動作するサーバの構成を以下の表 4.18に示す。

OS	FreeBSD 5.3 Relaese
CPU	Pentium4 1.6GHz
Memory	256MB
Database	PostgreSQL7.3.8

表 4.18: サーバの構成

アプリケーションの動作するホストの構成を以下の表 4.19に示す。

OS	FreeBSD 5.3 Release
CPU	Pentium4 2.4GHz
Memory	256MB

表 4.19: アプリケーションホストの構成

RFID システムは RF CODE 社の SPIDER READER と TOTOKU の MEGRAS を使用した。SPIDER はシリアルインターフェースしか持たないためシリアルサーバとして Cyclades-TS100 使用した。SPIDER と Cyclades-TS100 はシリアルストレートケーブルで接続され、設定は以下の通りである。

Baud rate	19200bps
Data bit	8bit
Parity	none
Stop bit	1bit
Flow contorol	none

表 4.20: シリアルの設定

Cyclades-TS100、MEGRAS、ミドルウェアホスト、アプリケーションホストは、全て同一セグメントのネットワークに接続され、グローバル IP アドレスが割り振られている。

Cyclades-TS100 と MEGRAS は、TCP クライアントモードで動作し、ミドルウェアホストとの接続が確立されるまで繰り返し接続を試みるという動作をする。

第5章　検証

本章では、4章で実装したミドルウェアの機能を検証する。

5.1　動作検証

本ミドルウェアにおいて検証するべき項目は以下の通りである。

- リーダイベントへ変換
  リーダごとに異なる位置情報ラベルを取得できているかどうか。また、異なる種類のリーダから取得された情報が、正しくリーダイベントへ変換されたかどうか。
- 実空間情報の管理
  検出されているタグの情報を保持できているかどうか。また、タグが移動した場合に反映されるかどうか。
- プレゼンス情報の生成
  リーダ設定情報に基づいて、検出・消失のプレゼンス情報が生成されるかどうか。
- 条件一致判定コンディション情報の記述通りにフィルタリングされているかどうか。

5.1.1　実験環境

本研究において実装したミドルウェアの動作検証を行うために構築した実験環境について述べる。
動作検証に用いた機材及び環境は、4章で述べた運用環境と同じである。

5.1.2　リーダイベントへの変換

本項では、リーダごとに異なる位置情報ラベルを取得できているかどうか、リーダから出力された情報が正しくリーダイベントへ変換されたかどうかを検証する。
リーダイベントへの変換機能を検証するために、リーダから出力された変換前のデータ及び変換後の構造体に含まれている各要素を、標準出力へ書き出す機能を取り入れて実行した。

図 5.1は、MEGRAS 用のリーダイベント変換プログラムの動作画面である。本プログラムでは、location.conf に記述された内容にしたがって IP アドレスから位置情報ラベルの取得を行う。

```
> cat location.conf
203.178.139.210 z202
```

```
> ./RF_megras
rport:8002 shost:localhost sport:8003 location:location.conf

1000794000091238

flag          : 0
tagid         : 1000794
location      : z202
current_time  : 1104267950
enter_time    : 0
interval_time : 0
elapsed_time  : 91238
count         : 0
```

図 5.1: MEGRAS 用リーダイベント変換プログラムの動作画面

図 5.2は、SPIDER 用のリーダイベント変換プログラムの動作画面である。

リーダから出力された情報は、MEGRASが「1000794000091238」、SPIDERが「H,HFLWTHN,1,1,8」である。どちらの情報にもリーダの識別情報は含まれていないが、変換後のリーダイベントには設定ファイルに記述されている位置情報ラベルが正しく格納されていることが確認できた。また、2 つのリーダからは異なるフォーマットの情報が取得されているが、変換後のリーダイベントにはタグ ID(tagid) 及び検出時間 (current_time) が正しく格納されていることが確認できた。その他の要素は、付加情報であるため、利用するリーダによって結果が異なる。

以上の結果から、異なるフォーマットの情報を出力するリーダから正しくリーダイベントへ変換されることが確認できた。

5.1.3 実空間情報の管理

本項では、検出されているタグの情報を保持できているかどうか、タグが移動した場合に正しく反映されるかどうかを検証する。

実空間情報の管理機能を検証するために、コマンドライン上からコマンドを入力することにより、管理機能が保持している実空間情報をすべて標準出力へ書き出す機能を取り入れて実行した。

図 5.3は、検証の開始時に保持していた実空間情報の一覧である。出力されている情報は左

```
> cat location.conf
203.178.143.220 z203
```

```
> ./RF_ts100
rport:8001 shost:localhost sport:8003 location:location.conf

H,HFLWTHN,1,1,8

flag          : 0
tagid         : HFLWTHN
location      : z203
current_time  : 1104268662
enter_time    : 0
interval_time : 8
elapsed_time  : 1
count         : 0
```

図 5.2: SPIDER 用リーダイベント変換プログラムの動作画面

から、タグ ID、位置情報ラベル、最終検出時間、最初の検出時間、である。この結果から、管理機能が実空間情報を保持できることが確認できた。

```
> telnet localhost 8004
Trying ::1...
Connected to localhost.
Escape character is '^]'.
SHOW
HFLWTHN -> z203 : 1104269701 : 1104269593
AHAHEFD -> z203 : 1104269699 : 1104269596
10007A4 -> z202 : 1104269701 : 1104269686
100079C -> z202 : 1104269692 : 1104269686
1000793 -> z202 : 1104269702 : 1104269686
100079B -> z202 : 1104269696 : 1104269686
1000791 -> z202 : 1104269686 : 1104269686
```

図 5.3: 開始時の実空間情報

次に、図 5.4は、新しくタグを検出範囲内に持ち込んだ時の実空間情報一覧である。一番下に 1000792 というタグ ID の実空間情報が追加されている。これにより、タグの検出に応じて新しく実空間情報を追加できる、ということが確認できた。

次に、図 5.5は、検出されているタグを検出範囲外へ持ち出した時の実空間情報一覧である。

```
SHOW
HFLWTHN -> z203 : 1104269801 : 1104269593
AHAHEFD -> z203 : 1104269798 : 1104269596
10007A4 -> z202 : 1104269801 : 1104269686
100079C -> z202 : 1104269797 : 1104269686
1000793 -> z202 : 1104269799 : 1104269686
100079B -> z202 : 1104269801 : 1104269686
1000791 -> z202 : 1104269802 : 1104269686
1000792 -> z202 : 1104269800 : 1104269800
```

図 5.4: 新しいタグを検出させた場合の実空間情報

先ほど検出した 1000792 というタグ ID の実空間情報が削除されている。これにより、タグの消失に応じて実空間情報を削除できる、ということが確認できた。

```
SHOW
HFLWTHN -> z203 : 1104269950 : 1104269593
AHAHEFD -> z203 : 1104269948 : 1104269596
10007A4 -> z202 : 1104269951 : 1104269686
100079C -> z202 : 1104269948 : 1104269686
1000793 -> z202 : 1104269947 : 1104269686
100079B -> z202 : 1104269922 : 1104269686
1000791 -> z202 : 1104269950 : 1104269686
```

図 5.5: タグを消失させた場合の実空間情報

以上の結果から、実空間情報管理機能は、正しく情報を保持していることが確認できた。

5.1.4 プレゼンス情報の生成

本項では、リーダ設定情報に基づいて、検出・消失のプレゼンス情報が生成されるかどうかを検証する。

プレゼンス情報の生成機能を検証するために、コマンドライン上からリーダ設定情報を変更する機能を取り入れた。また、プレゼンス情報が生成された場合に、そのプレゼンス情報と現在時刻を標準出力へ書き出す機能を取り入れて実行した。

図 5.6は、消失と判定するまでの時間を 10 秒に設定したプレゼンス情報生成プログラムの動作画面である。now はプレゼンス情報が生成された時間、current_time は最終検出時間、enter_time は最初の検出時間である。

消失判定時間を 10 秒に設定した後は、最終検出時間から 10 秒以上経過したタグ ID の消失プレゼンス情報が生成されていることが確認できる。

```
> telnet localhost 8004
Trying ::1...
Connected to localhost.
Escape character is '^]'.
TIME
timeout = 60
TIME 10
timeout = 10
TAG LEAVE
tagid        : AHAHEFD
location     : z203
now          : 1104271773
current_time : 1104271762
enter_time   : 1104271754
TAG LEAVE
tagid        : 100079B
location     : z202
now          : 1104271780
current_time : 1104271769
enter_time   : 1104271759
TAG LEAVE
tagid        : 1000792
location     : z202
now          : 1104271803
current_time : 1104271792
enter_time   : 1104271792
```

図 5.6: プレゼンス情報の生成プログラムの動作画面

以上の結果から、リーダ設定情報に基づいて正しくプレゼンス情報が生成されていることが確認できた。

5.1.5 条件一致判定

本項では、コンディション情報の記述通りにフィルタリングされているかどうかを検証する。

条件一致判定機能を検証するために、コマンドライン上からコンディション情報を設定できる機能を取り入れて実行した。また、検証にはスイッチ機能つきタグを用いて、そのタグのスイッチを押したときに生成されるプレゼンス情報が、コンディション情報を設定したコマンドライン上に表示されるかどうかによって検証を行った。

表 5.1は、検証した条件の登録パターンとその結果を示したものである。

○……送信されるプレゼンス情報と一致する値を登録した時
△……送信されるプレゼンス情報とは異なる値を登録した時
×……コンディション情報を登録していない時

データタイプ	タグ ID	位置情報ラベル	結果
×	×	×	成功
○	×	×	成功
△	×	×	失敗
×	○	×	成功
×	△	×	失敗
×	×	○	成功
×	×	△	失敗
○	○	○	成功
△	△	△	失敗

表 5.1: 条件一致機能の検証結果

以上の結果から、一つでも条件に一致しない項目がある場合には、情報はフィルタリングされるということが確認できた。この結果は、本ミドルウェアの持つフィルタリング機能が正しく動作していることを示している。

5.2 サンプルコード

本節では、従来の手法で書かれたコードと本ミドルウェアを利用して書かれたコードを比較して、どれだけアプリケーションの開発コストを軽減できたかを検証する。

サンプルコードとして、新しく部屋に入ってきた人を検出する関数を以下に示す。

```
char *detect_into_room(int con)
{
  struct listnode *node;
  char *p[5], *h;
  int flag = 0;
  int i;
  time_t now;
  char *res;

  char buf[MSGLEN];
  int len;

  res = (char *)malloc(MSGLEN);
  memset(res, 0, sizeof(res));
```

```
memset(buf, 0, sizeof(buf));

while((len = read(con, buf, sizeof(buf))) > 0){
  strcpy(res, buf);
  now = time(NULL);

  h = strtok(buf, ",");
  if((strncmp(h, "H", 1)) != 0){
    continue;
  }

  for(i=0 ; i < 5 ; i++){
    p[i] = strtok(NULL, ",");
    if(p[i] == NULL){
      continue;
    }
  }
  if(strlen(p[0]) != 7){
    continue;
  }

  /* リストの探索 */
  for(node=listhead ; node ; node=node->next){
    /* 同じ ID と場所だった場合 */
    if((strcmp(node->tagid, p[0])) == 0 && (strcmp(node->location, p[4])) == 0){
      flag = 1;
      node->current_time = now;
      node->interval_time = atoi(p[3]);
      node->elapsed_time = atol(p[1]);
      node->count++;
      break;
    }
  }

  /* 新しいタグが検出された場合 */
  if(flag == 0){
    node = listnode_alloc(p[0], p[4], now, atol(p[1]), atoi(p[3]));
    break;
  }
}
```

```
  return res;
}
```

```
RS_RESULT *detect_into_room(int con)
{
  RS_RESULT *res;

  set_cond_type(con, 1);

  start_event(con);

  res = (RS_RESULT *)malloc(sizeof(RS_RESULT));
  memset(res, 0, sizeof(RS_RESULT));

  read(con, res, sizeof(RS_RESULT));

  return res;
}
```

図 5.7: サンプルコード

図 5.7は、従来の手法で記述したコードと本ミドルウェアを利用して書かれたコードである。本ミドルウェアを利用して書かれたコードは、従来のコードと比較して、取得した情報のフォーマットを解析する部分と既に検出されている ID かどうかをチェックする部分が省略されているため、非常に簡潔に記述することが可能になっている。

5.3 運用実績

本節では、本ミドルウェアの運用実績について述べる。

本システムの運用実績としては、研究室内における在席記録システムとして 2004 年 6 月 21 日 18:03:20 から 2004 年 8 月 24 日 09:05:51 までの間 1 度も停止することなく運用した。運用環境としては、2 台をリーダを設置して、20 個ほどのタグを配布した。その他の環境については、4章で述べた運用環境と同じである。

また、2004 年 9 月 6 日から 2004 年 9 月 9 日の間に行われた WIDE 合宿においてミーティング資料自動配布アプリケーションを開発し、本システム上で運用した。運用環境としては、8 台のリーダを設置して、およそ 280 名ほどの参加者へタグを配布した。その他の環境については、4章で述べた運用環境と同じである。

第6章　結論

本章では、本研究をまとめ、今後の課題を挙げて、本論文の結論とする。

6.1　まとめ

本研究では、RFID システムを用いた実空間アプリケーションを開発する際の問題点を解決し、実空間アプリケーションへ統一されたインターフェースを提供するためのミドルウェアを提案し、設計・実装を行った。

従来のアプリケーション構築環境では、複数の種類のリーダに対応するアプリケーションを構築することが、困難であるという問題を述べた。また、複数のアプリケーションでリーダを共有すると多くの不要な情報が収集され、本来は必要としない処理を行わなければならない、という問題点があることを述べた。これらの問題点をアプリケーションごとに解決するには、多大な開発コストを必要とする。

そこで、本研究では、これらの問題点を解決するために必要な機能要件を挙げ、その機能要件を満たす 3 層のレイヤー構造からなるミドルウェアを提案した。そして、そのモデルに基づいた設計及び実装を行い、問題点を解決するのに十分な機能を提供できるミドルウェアであることを述べた。

本システムを用いることによって、複数のリーダ、複数のアプリケーションが存在する環境でも統一されたインターフェースでアプリケーションを開発することが可能になった。

6.2　今後の課題

本節では、今後、本システムを利用していく上で考えられる今後の課題について述べる。

6.2.1　他言語用ライブラリ

本システムが提供するライブラリは、C 言語用のライブラリのみである。しかし、C 言語だけでは、利用できる環境が制限されたり、既存システムとの親和性において、充分な機能を提供しているとはいえない。そこで、他言語のライブラリを用意することによって、より柔軟なアプリケーション開発環境を提供することができる。

6.2.2 リーダ認証

本システムは、誰が設置したリーダであっても同じシステム内のアプリケーションから共有される、というモデルである。しかし、このような環境では、悪意のある利用者が偽の情報を発信するリーダをシステム内で動かした場合に偽の情報が流通してしまうという問題が発生する。そこで、正しい情報を発信するリーダを認証することによって、悪意のあるリーダからの情報を受け付けないようにする機構が必要である。

6.2.3 ユーザ認証

リーダ認証と同様に、共有されたリーダから取得される情報は、全てアプリケーションへ提供されるため、利用者がその情報を悪用する可能性が考えられる。また、利用者によっては他人に提供したくない情報が存在するかもしれない。そこで、利用者ごとに提供する情報を変化させるための認証機構が必要になると考えられる。

6.2.4 データ処理の高速化

RFID システムが本格的に利用されるようになると劇的に情報量が増加すると予想されている。リーダが 1 台増えるごとに大量の情報が発信されるようになるため、情報を集中的に管理するモデルでは、高速なデータの処理能力が要求される。そのため、将来的に利用するためには、より高いパフォーマンスを発揮する技術を取り入れなければならない。

6.2.5 高度なフィルタリング機能

本システムで述べたようなフィルタリング機能は、基本的な機能のみに留まっている。しかし、様々な利用者の要求に対応していくためには、より高度なフィルタリング機能を実現していく必要がある。例えば複数の利用者で条件を共有できる機能など様々な利用方法が考えられる。

謝辞

本研究を進めるにあたり，御指導を頂きました，慶應義塾大学環境情報学部教授村井純博士，徳田英幸博士，同学部助教授の楠本博之博士，中村修博士，同学部専任講師の南政樹氏，重近範行博士に感謝致します.

参考文献

[1] Epcglobal. *http://www.epcglobalinc.org/*.