拡張現実感を実現する分散マルチメディアミドルウェア

Distributed Multimedia Middleware for Augmented Reality

倉橋　誠　　徳永　英治　　根本　将寛
Makoto KURAHASHI　Eiji TOKUNAGA　Masahiro NEMOTO

早稲田大学大学院理工学研究科（〒169-8555 新宿区大久保 3-4-1）
{mik, eitoku, nemoto}@dcl.info.waseda.ac.jp

ABSTRACT. Ubiquitous computing will dramatically change our lives due to the enhancement of our real world. Mixed reality (MR) is a promising technique for realizing the enhancement by superimposing computer generated images on video images. However, it is not easy to build applications using mixed reality techniques since the developer needs to deal with issues like distribution and context-awareness. It is desirable to provide a software infrastructure to hide the complexities from programmers.
In this paper, we propose a middleware called MiRAGe (Mixed Reality Application Generater) supporting mixed reality for ubiquitous computing environments. Our middleware provides sereral multimedia components that process video streams using MR techniques. New components to provide more complex functionality can be delevoped by composing these components.

1　背景および目的

近い将来の実現に向けて研究が進んでいるユビキタスコンピューティング環境 [3, 7, 10, 11] では，我々の生活する実世界に，くまなく，様々な種類のネットワーク接続されたコンピュータが配置されてサイバー空間を形成する．このような環境では，実世界とサイバー空間をシームレスに融合し，コンピュータの存在を意識することなく，ユーザがサイバー空間の能力を利用できることが重要となる．そのための技術として複合現実感 [2] が有効であると考える．複合現実感とは，コンピュータによってリアルタイムに，実世界の上に情報をスーパーインポーズするものであり，実世界を拡張する直感的なインタフェースである．　複合現実感を実現するためには，どこになにをスーパーインポーズするかを判断するために，コンピュータが現実世界の視界を理解しなければならない．そのための方法に，ビジュアルマーカと呼ばれるバーコード状のものを実世界に配置し，カメラで取り込んでコンピュータで解析する方法がある．これはよく用いられる方法のため，ビジュアルマーカの検出などを行うソフトウェアライブラリは既に利用可能である．しかし，従来，複合現実感はユビキタスコンピューティングとは直接関係ないものであったため，これを同環境で利用しようとなると，新たな問題が発生する．　まず，ユビキタスコンピューティング環境は，多種多様のコンピュータやデバイス，ネットワークで構成されるため，これらそれぞれの環境に合わせてプログラムを開発することは非常に効率が悪いこと．貧弱な機器でも複合現実感のようなサービスを利用したい場合に計算負荷の分散を図るためにアプリケーションの分散化が必要となり，複雑さが増すこと．実世界の状況に適合しながら，アプリケーションの振る舞いを動的に変えていかなくてはならないこと等が，ユビキタスコンピューティングにおける問題として挙げられる．　当事業は，これらを解決しユビキタスコンピューティング環境で複合現実感アプリケーションの容易な構築を実現するためのミドルウェアを提案し，設計・実装および評価を行い，有効性を確認することである．

2　概要

当事業で開発する，ユビキタスコンピューティング環境で複合現実感を実現するためのミドルウェアは MiRAGe(Mixed Reality Application Generator) と命名されている．MiRAGe は，図 1 のように大きく分けて 2 層からなり，下を MiRAGe 通信インフラストラクチャ，上をマルチメディアフレームワークと呼んでいる．　MiRAGe 通信インフラストラクチャは CORBA[8] ベースの分散システムである．この層の状況トレーダは，OASiS[6] と呼んでいる状況データベースを利用して，アプリケーションが利用する CORBA サービスへの参照を管理，状況に応じてアプリケーションの再構成を実現する CORBA サービスである．　マルチメディアフレームワークは，ビデオストリームを扱うアプリケーションを CORBA オブジェクトを組み立てて構築するフレームワークであり，複合現実感を実現するためのコンポーネントを用意している．　このシステムでは，CORBA の利用により分散化への対応と，OS や言語に依存しない移植性の向上を図っている．また，状況トレーダによる動的適合への対応によって，ユビキタスコンピューティング環境に適したシステムとしている．　なお，開発は Linux 上の C++および C で行い，コンパイラは gcc を使用した．

3　開発内容

3.1　MiRAGe 通信インフラストラクチャ

マルチメディアフレームワークにおいて生成するメディアコンポーネントは，独自の CORBA ベース通信インフラストラクチャを通して通信することで，コンポーネント間接続の動的な再構成が可能となる．この通信インフラストラクチャは，環境状況を認識するための CORBA サービスである状況トレーダからなり，環境状況を取得及び集積するデータベースである OASiS を利用する．メディアコンポーネント間のストリームは，状況トレーダに含まれるコンフィギュレーションマネージャによって

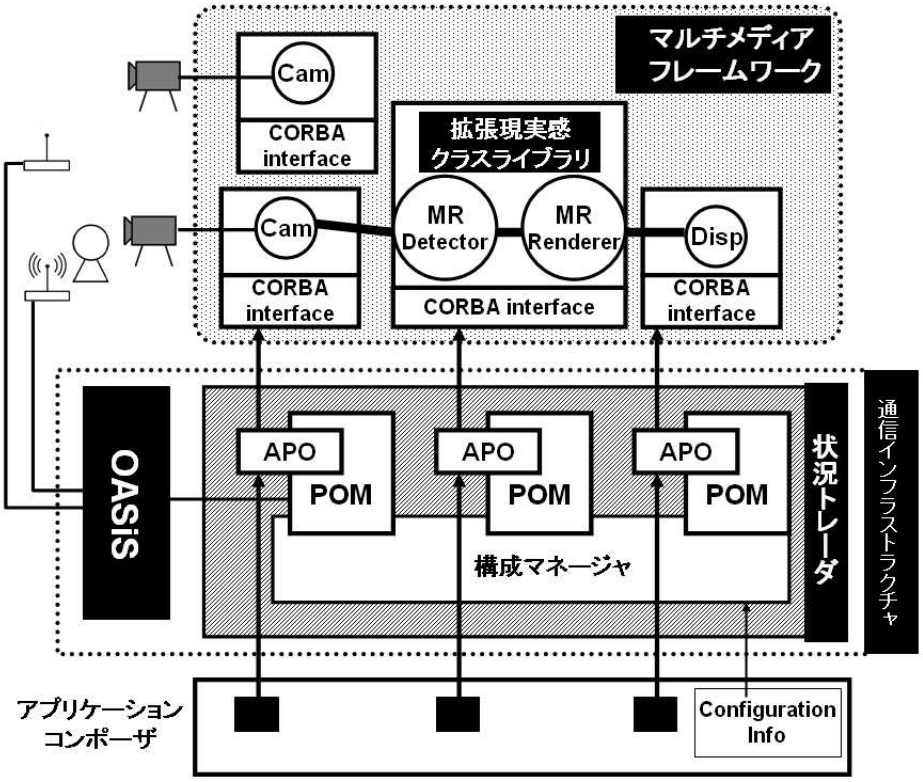

図 1: 全体像

再構成される． 開発者は，状況トレーダが提供する適応擬似オブジェクト (APO) の機能を利用して，アプリケーションコンポーザを記述する．APO は擬似オブジェクトマネージャ (POM) によって管理される．POM は OASiS と通信し，APO を自動的に再構成する．アプリケーションコンポーザは，APO を通してメディアコンポーネントと通信することになる．(図 2)．

3.1.1 適応疑似オブジェクトによるメソッドコールの動的転送

適応擬似オブジェクト (APO) は，CORBA オブジェクトの擬似オブジェクトであり，GIOP[8] で規定されている LOCATION_FORWARD メッセージを使ってメソッドコールをターゲット CORBA オブジェクトへ転送する．各 APO はそれぞれ 1 つの擬似オブジェクトマネージャ (POM) によって管理される．POM は，状況の変化に応じて，動的に APO のメソッドコール転送先を変更する．MiRAGe においては，APO はメディアコンポーネントの擬似オブジェクトであり，ターゲットのメディアコンポーネントと同じインタフェースを持つ． POM との連携により，APO は受けたメソッドコールをその状況に最も適したメディアコンポーネントへ転送する．APO のメソッドコール転送先は，POM に対して再構成ポリシ (Reconfiguration Policy) を記述することにより決定される．POM はシステム起動時に OASiS と通信し，環境状況を検索，再構成ポリシに適した初期 IOR を受け取る．状況が変化し，OASiS が再構成ポリシに適した新しいオブジェクトを発見すると，POM に新しい IOR が非同期に送信され，APO のメソッドコール転送先が動的に更新される．

図 3 は，メソッドコールの動的転送の仕組みを示す図である．以下にシーケンスの詳細を記す．

1) POM に再構成ポリシを記述する
2) POM は再構成ポリシポリシに適したオブジェクトを OASiS に問い合わせる
3) OASiS はポリシに適したオブジェクトの初期 IOR を POM に返す
4) POM は管理している APO のメソッドコール転送先を渡された IOR へ設定する
5) APO が活性化される
6) APO のメソッドを起動される
7) APO はメソッドコール転送先の IOR を格納した LOCATION_FORWARD メッセージを返す
8) メソッドコール転送先オブジェクトのメソッドを起動する
9) 状況の変化 (ユーザ位置の変更など) を OASiS が認識し，新しい状況に適合する IOR を POM に送る
10) POM が APO のメソッドコール転送先を新しい IOR へ更新する
11) 再度 LOCATION_FORWARD メッセージによって，新しいターゲットへメソッドコールを転送する．
12) クライアントのメソッドコールは，透過的に新しいターゲットへ転送される

3.1.2 状況トレーダサービス

図 2 は，状況トレーダとマルチメディアフレームワークの関係を示している．アーキテクチャの概要で述べたように，状況トレーダはメディアコンポーネントの自動的な再構成を行う CORBA サービスである．本節では，状況トレーダの働きを詳説する．以下に状況トレーダを利用する際に記述するソースコードの一例を示す．

```
CORBA::Object_var obj =
  orb->resolve_initial_reference("SituationTraderService"); //(1)
STrader::SituationTraderFactory_var factory =
              STrader::SituationTraderFactory_narrow(obj);

STrader::POManager_var camera_pom =
        factory->createPOManager("IFACE::MCompIface:1.0"); //(2)
STrader::POManager_var display_pom =
        factory->createPOManager("IFACE::MCompIface:1.0");
STrader::ConfigurationManager_var com =
                    factory->createConfigurationManager();
```

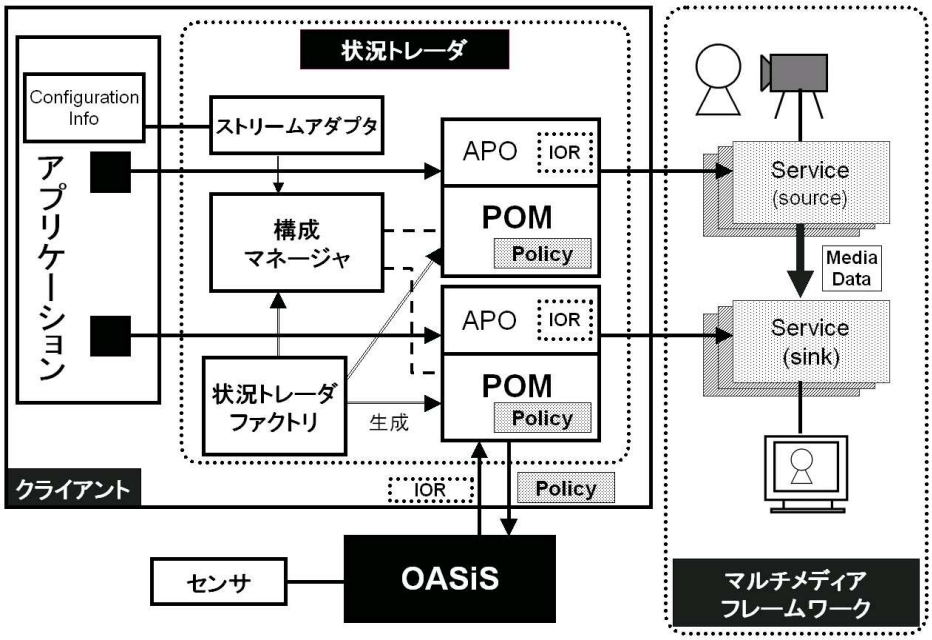

図 2: 状況トレーダ

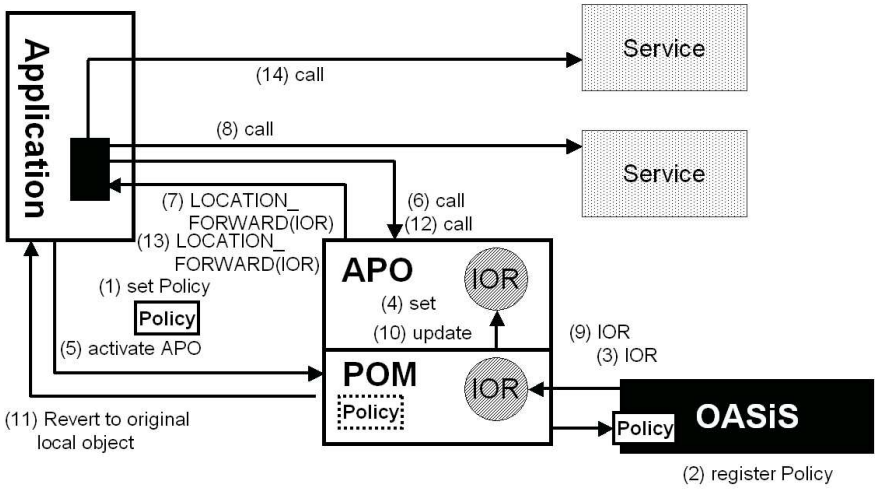

図 3: メソッドコール動的転送のシーケンス

```
STrader::ReconfigurationPolicy camera_policy;
STrader::ReconfigurationPolicy display_policy;

camera_policy.locationScope = "Laboratory" //.................(3)
camera_policy.locationTarget = "Eiji TOKUNAGA"
camera_policy.locationContext = "nearest"
camera_policy.serviceType = "Camera"

display_policy.locationScope = "Distributed Computing Laboratory"
display_policy.locationTarget = "Andrej van der Zee"
display_policy.locationContext = "nearest"
display_policy.serviceType = "Display"

camera_pom->setPolicy(camera_policy); //.....................(4)
display_pom->setPolicy(display_policy);

IFACE::MCompIface_ptr camera_apo = camera_pom->activateAPObject();
IFACE::MCompIface_ptr display_apo = display_pom->activateAPObject();

MStream stream(MStream::NORMAL);
stream.setSource(camera_apo, 1);
stream.setSink(display_apo, 1);

StreamAdapter_i* adapter_i = new StreamAdapter_i(stream); //..(5)
STrader::ConfigurationAdapter_ptr adapter = adapter_i->_this();

com->setAdapter(adapter); //.................................(6)
com->addPOManager(camera_pom); //............................(7)
com->addPOManager(display_pom);

stream.start();
```

図 2 に示されるように，状況トレーダは状況トレーダファクトリとコンフィギュレーションマネージャ及び，POM から構成される．状況トレーダファクトリは CORBA が提供する `resolve_initial_reference()` メソッドから取得する (1)． 状況トレーダファクトリはコンフィギュレーションマネージャや POM を生成する (2)．`createPOManager` メソッドはターゲットオブジェクトの型を表すレポジトリ ID を引数に取る． 各 POM には再構成ポリシが記述される (4)．POM は OASiS を利用して，ポリシに最も適したオブジェクトを検索する． 現状の再構成ポリシは `locationScope`，`locationTarget`，`locationContext` の 3 つの位置パラメータで記述される (3)．`locationScope` は，メディアコンポーネントを検索する範囲を指定するパラメータである．OASiS は `locationScope` で指定された範囲の中で，状況に適合したメディアコンポーネントを検索する．`locationTarget` は位置情報のターゲットとなる物理オブジェクトを指定するパラメータであり，`locationContext` にはその物理オブジェクトと検索するメディアコンポーネント

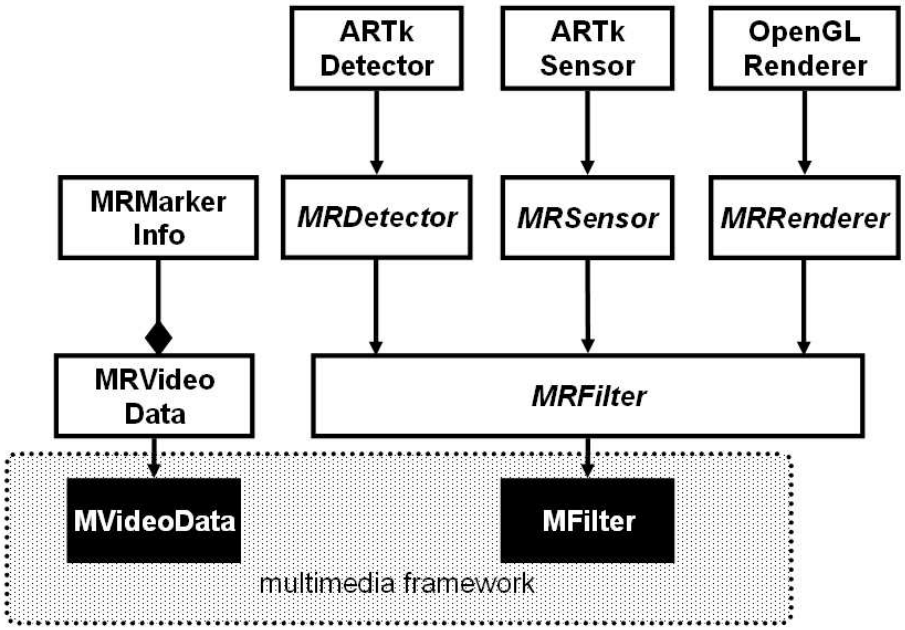

図 4: 複合現実感のためのクラス

との関係を記述する．locationTarget には機器や，人間，家具などの名前が記述できる． 現状の仕様では，locationContext には nearest と exact が指定できる．nearest は，locationTarget で記述された物理オブジェクトに最も近いメディアコンポーネントを検索することを意味する．たとえば，locationContext が”nearest”，locationTarget が”Eiji TOKUNAGA”と記述されている場合，OASiS は Eiji TOKUNAGA という人物に最も近いメディアコンポーネントを検索する．”exact” は locationTarget で指定された物理オブジェクトと接続されているメディアコンポーネントを直接指定する．たとえば，locationContext が”exact”で locationTarget が”Camera1”だった場合，Camera1 に接続されたメディアコンポーネントを指定する．現在，より有効なポリシ記述を考察中である． POM はコンフィギュレーションマネージャによって，自動的に管理される．(6) では，自動的にストリームのリダイレクションを行うためのモジュールであるストリームアダプタ (StreamAdapter) をコンフィギュレーションマネージャに登録している．コンフィギュレーションマネージャに登録されている POM が OASiS によって更新された場合，ストリームアダプタが状況の変化を受けて，メディアコンポーネント間のストリームを再構成する (5〜7).

3.1.3 ストリームの再構成

ストリームアダプタはマルチメディアフレームワークの MStream クラスを利用してストリームの再構成を行う．状況の変化に対応して，ストリームアダプタのコールバックハンドラが起動し，MStream クラスのインスタンスを通してストリームを再構成する． ここで，カメラモジュールで画像を生成し，ディスプレイモジュールに送って画面出力する例を考える．ディスプレイ付近にいたユーザが移動するなど，他のディスプレイに画像を出力するのが適当な状況になった場合，ストリームアダプタのコールバックハンドラが起動され，MStream クラスのインスタンスを更新して，ストリームを再構成する． 具体的には，まず MStream クラスのインスタンスが，ソースモジュールのストリーム識別子を取り除き，ストリームを止める．次に，古い参照を新しい参照へ変更して，ストリームを再起動する．

3.2 マルチメディアフレームワーク

マルチメディアフレームワークでは，ビデオストリームの生成（カメラからの画像取り込みやストレージからの読み込みなど）や消費（ディスプレイに表示するなど），何らかの処理などを行うオブジェクトであるマイクロモジュールを組み立てて，CORBA オブジェクトであるマルチメディアコンポーネントを立ち上げ，これを接続させて一つのマルチメディアアプリケーションを組み立てる．

3.2.1 マイクロモジュール

マイクロモジュールは，マルチメディアフレームワークでアプリケーションを構成するプログラム部品の最小単位である．実装は C++のクラスである．マイクロモジュールは，マルチメディアストリームとの関係で，ストリームを生成するソース，処理するフィルタ，消費するシンクの 3 種類に分類される．それぞれ MSource，MFilter，MSink と 3 つの抽象クラスを用意している．実際の機能を持つマイクロモジュールを実装するには，これらの抽象クラスを継承する．これらのクラスのフック関数を実装することで，目的に応じた機能を持つマイクロモジュールとすることができる．これにより，マイクロモジュールはいずれのメディアコンポーネントでも利用できるようになり，再利用性が向上している．また，後述のメディアコンポーネント間でストリームを送受信するためのソケットである MClientSocket と MServerSocket の二つのクラスを定義している．これらは，メディアコンポーネントが持つ CORBA の MConnIface インタフェースが提供する関数によって生成，接続される．MServerSocket オブジェクトは入力ポートを持ち，接続された MClientSocket オブジェクトから TCP コネクションを通じてデータを受信する．MClientSocket オブジェクトは，接続した MServerSocket オブジェクトにデータを送信する．

3.2.2 マルチメディアコンポーネント

マルチメディアコンポーネントは，マイクロモジュールを複数持つことのできるプログラム部品で，CORBA インタフェースを持ち，CORBA を利用してネットワーク経由で操作を行うことができる．マイクロモジュールを利用してメディアコンポーネントを生成するために，MComponent

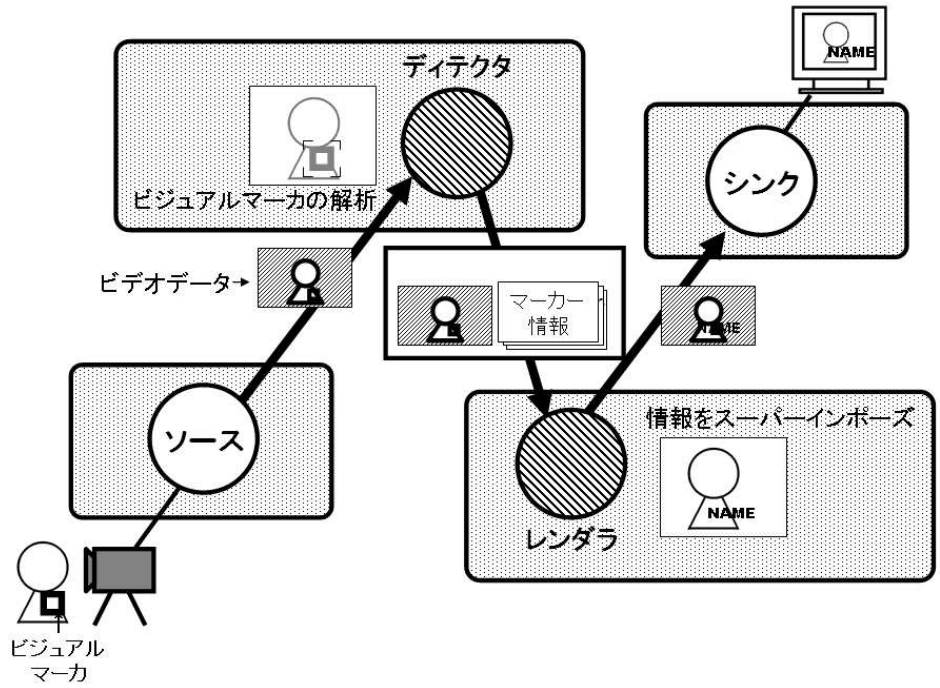

図 5: 複合現実感アプリケーション構築例

クラスがある．このオブジェクトにマイクロモジュールを追加することで，メディアコンポーネントを生成できる．たとえば，MSource，MFilter，MSink を継承して各機能を持たせた Camera，RBSwapper(生画像データの赤要素と青要素の順番を入れ換えるフィルタ．Video for Linux で取り込んだ画像データは”GRB”という順番になっているため，このようなフィルタを利用して一般的な”RGB”という順番のデータに変換する)，Display の各オブジェクトを持ったメディアコンポーネントを生成するソースコードは以下のようになる．

```
MyComponent::
MyComponent():
   MComponent()
{
   m_pCamera = new Camera;
   m_pSwapper = new RBSwapper;
   m_pDisplay = new Display;

   addObject(m_pCamera);
   addObject(m_pSwapper);
   addObject(m_pDisplay);
}
```

生成したメディアコンポーネントは，内部の各マイクロモジュールのスレッドを起動し，ネームサーバにユーザ指定の名前で自身の参照を登録した後，CORBA のリクエストを待機する．なお，メディアコンポーネント内のマイクロモジュールは，マイクロモジュール識別子によって識別される．これは，メディアコンポーネントにマイクロモジュールを追加した時点で割り振られ，メディアコンポーネントとマイクロモジュールの ID の組の型で表される．メディアコンポーネント接続などの操作を行うために CORBA インタフェースを定義している．CORBA インタフェースには，MCompIface，MConnIface，MservIface の 3 つがある．MCompIface インタフェースは，メディアコンポーネントの基本的なインタフェースであり，すべてのメディアコンポーネントに共通するインタフェースである．これを通じて，下の 2 つの CORBA インタフェースを取得することができる．MConnIface インタフェースは，マイクロモジュール間のストリーム接続を制御するためのインタフェースである．これもまた，全てのコンポーネントに共通の CORBA インタフェースである．MServIface インタフェースは，メディアコンポーネント内のマイクロモジュールに対して固有の操作をするためのインタフェースである．このインタフェースを通して，マイクロモジュールの状態を変更したり，問い合わせたりする．たとえば，ディスプレイマイクロモジュールに対しては，ディスプレイマイクロモジュール特有の，解像度を変更したり，問い合わせたりするインタフェースを提供することができる．これは，各コンポーネントが持つマイクロモジュールの種類によって異なる．

3.2.3 ストリーム

本フレームワークでは，マルチメディアデータがソースで生産されてから，シンクに至って消費されるまでを一本のストリームとして扱う．ストリームはストリーム識別子を使って識別され，ソースモジュールで生産されたデータを最初に送信するときに，それが特定のストリームのデータであることを示すため，データに識別子を振る．出力ポートのあるマイクロモジュールには，ストリームテーブルがある．この中では，各ストリームの経路情報が [StreamId，ObjectId][*1] という形で設定されていて，データを出力するときには，テーブル内のマッチする情報に基づいて出力先を決定する．このテーブルの情報の設定や変更は，MConnIface を通じて行う．また，あるストリームが利用する各コンポーネントの MConnIface を操作し，ストリームの設定を一括して行うためのファサードとして MStream クラスがある．このクラスは，ストリーム設定のための直感的なインタフェースを持っており，この MStream オブジェクトにストリームのソース，シンク，と通過するフィルタを指定，start や stop の関数でストリームの動きを制御することができる．

3.2.4 複合現実感ツールキット

本システムでは，複合現実感処理を行うためにマルチメディアフレームワーク上のクラスライブラリを提供している．本システムが提供する複合現実感機能の主要な機能は，以下のものに分けられる．

[*1] StreamId はストリーム識別子，ObjectId はマイクロモジュール識別子

図 6: 家電制御インタフェース

図 7: リモコン

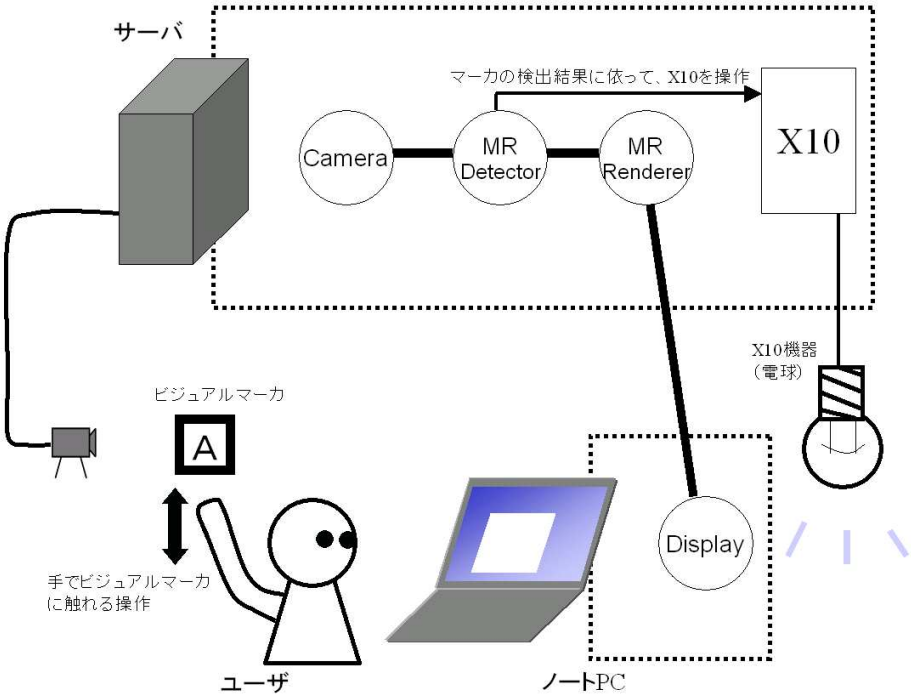

図 8: 複合現実感アプリケーション構築例

- 処理の対象とする現実世界の情報を取得する機能．具体的にはビデオフレーム等の画像の中からビジュアルマーカを検出する機能．本システムでは，この働きをするマイクロモジュールを，ディテクターと呼ぶ．
- ビジュアルマーカの解析結果から，ビデオフレーム上に 3D オブジェクトなどをスーパーインポーズする機能．本システムでは，この働きをするマイクロモジュールを，レンダラーと呼ぶ．
- ビジュアルマーカの解析結果の表現．本システムではこの働きをするマイクロモジュールを，マーカー情報と呼ぶ．

図 4 の中の黒地のクラスは先に述べた分散マルチメディアフレームワークが提供するクラスであり，同フレームワークの MFilter クラスを継承した MRDetector と MRRenderer，MRSensor の三つが，複合現実感のための主要な機能を提供するクラスである．また，そのほかに，ビデオデータにマーカ情報を付加して扱うための，MVideoData を継承した MRVideoData がある．MRDetector クラスは，受け取ったストリームのビデオフレーム内にあるビジュアルマーカーを検出し，その検出結果を付加したビデオストリームを出力する．MRDetector オブジェクトは，実際の解析を行う MRProcessor オブジェクトを一つ持ち，利用する．MRProcessor を継承したクラスには，ARToolKit[1] のライブラリを利用した ARTkProcessor クラスがある．MRDetector で検出されたマーカーの情報は，MRMarkerInfo オブジェクトで表す．MRMarkerInfo は抽象クラスであり，ARTkProcessor を利用した場合の検出結果は，MRMarkerInfo を継承した ARTkMarkerInfo

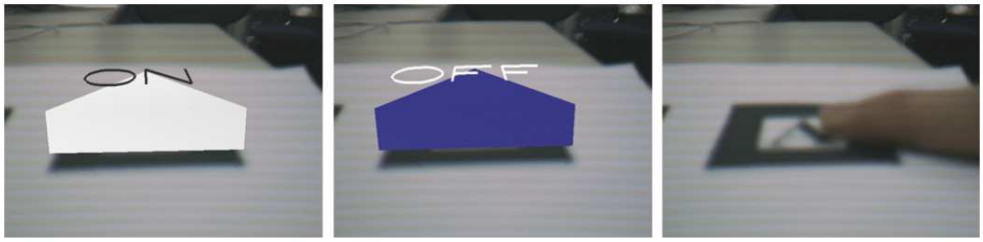

図 9: 実行画面

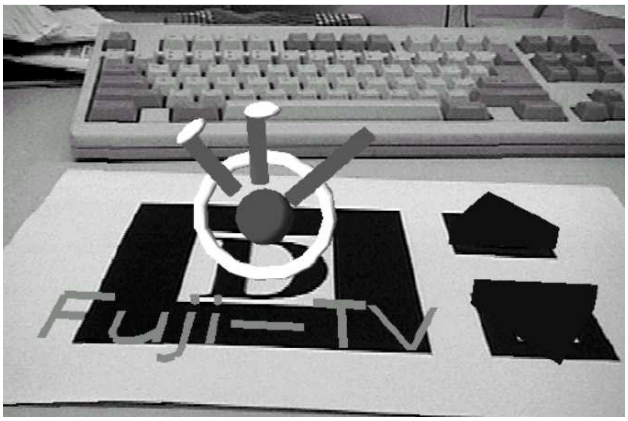

図 10: テレビコントロールインタフェース例

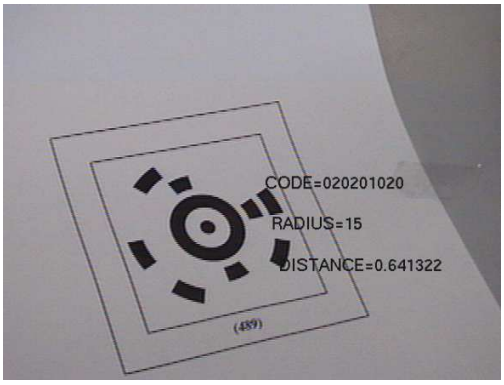

図 11: TRIP 利用例

オブジェクトとなる．MRVideoData オブジェクトは，元のビデオフレームに，この検出結果を付加したものである．MRRenderer クラスは，MRVideoData オブジェクトを受け取り，ビデオフレーム上に 3D オブジェクトなどをスーパーインポーズする．MRRenderer クラスは抽象クラスであり，これを継承した ARTkRenderer クラスは，OpenGL[9] を利用してオブジェクトをレンダリングする．MRSensor クラスも MFilter を継承するクラスである．MRDetector の場合と同様に，これを継承する ARTkSensor 等のクラスでビデオフレームを解析し，その結果を，あらかじめ設定しておいた OASiS に送る．これにより，ビジュアルマーカを利用したイメージセンサー的な役割を果たす． これらを組み立てて，簡単な複合現実感アプリケーションを構築した例は図 5 のようになる．

3.3 サンプルアプリケーション

本システムを利用したアプリケーションの一例として，家電制御インタフェースアプリケーションを構築した (図 6)．これは，ビジュアルマーカの検出を通じて，ユーザが行う ON/OFF の操作を認識行うものである．

このアプリケーションの構造は図 7 のようになる．このアプリケーションでは，ビジュアルマーカが写る位置に固定カメラ (写真左) が設置され，通常，ビジュアルマーカが検出され続ける．ユーザが操作を行うには，手などでビジュアルマーカに触れることによって，マーカが完全にカメラに写らない状況を作る．すると，カメラからビデオフレームが送られる MRDetector オブジェクトでマーカの非検出が起こり，これによってユーザの操作が行われたことを検出，X10[12] を利用して照明機器の電源 ON/OFF を行う．なお，ビデオフレームはその後 MRRenderer に送られ，現在の状態を示すイメージをスーパーインポーズし，ユーザ近くのノート PC の画面に表示される (図 8)．

このアプリケーションを利用して，X10 など PC から操作可能な機器であれば様々な機器を操作することが可能である．たとえば，PC からシリアル経由でテレビやビデオなどの機器を制御できるリモコン [4] (図 7) を利用して，それらの機器を操作することができる．この場合，複数のビジュアルマーカを利用して，電源 ON/OFF の他にチャンネル切り替え等の様々な操作を行うことができる (図 10)．

また，TRIP[5] と呼ばれるビジュアルマーカを利用すると，カメラからビジュアルマーカまでの距離を取得することができる (図 11)．このように，ビジュアルマーカにはそれぞれ特徴をもった様々な種類が存在するが，本システムでは，ビジュアルマーカの検出をコンポーネント化することにより，それらを状況に合わせて簡単に使い分けることができる．

3.4 テスト，評価等

本システムの分散化によるパフォーマンス向上を測定するため，次のようなテストを行った．ノート型コンピュータで複合現実感を利用するために，カメラから取り込んだ画像に複合現実感処理を施し，ノート型コンピュータのディスプレイに映像を映すアプリケーションを考える．この場合，単体のコンピュータによる複合現実感アプリケーションであれば，このサービスを利用したいノート型コンピュータでアプリケーションのすべての処理を行うはずである．この状態を（ハ）とする．次に，本システムの特長を生かした，高いパフォーマンスが得られると考えられる方法として，画像中のビジュアルマーカの解析およびレンダリングまでをより高性能のミドルタワー型コンピュータで行い，ノート型コンピュータでは表示だけを行う場合を（イ）とする．最後にこの中間の，ビジュアルマーカの解析をミドルタワー型コンピュータ，レンダリングをノート型コンピュータで行う場合を（ロ）とする．なお今回の実験では，接続方法を含めたカメラのキャプチャ性能によって全体のパフォーマンスに大きく影響が出るにもかかわらず，ノート型コンピュータに直接，ミドルタワー型コンピュータに接続するのと同様に性能の高いカメラを取り付ける方法がなかったため，（ハ）のケースも含めてビデオの取り込みはすべてミドルタワー型マシンで行った．ミドルタワー型マシンには Bt878 系キャプチャボードを

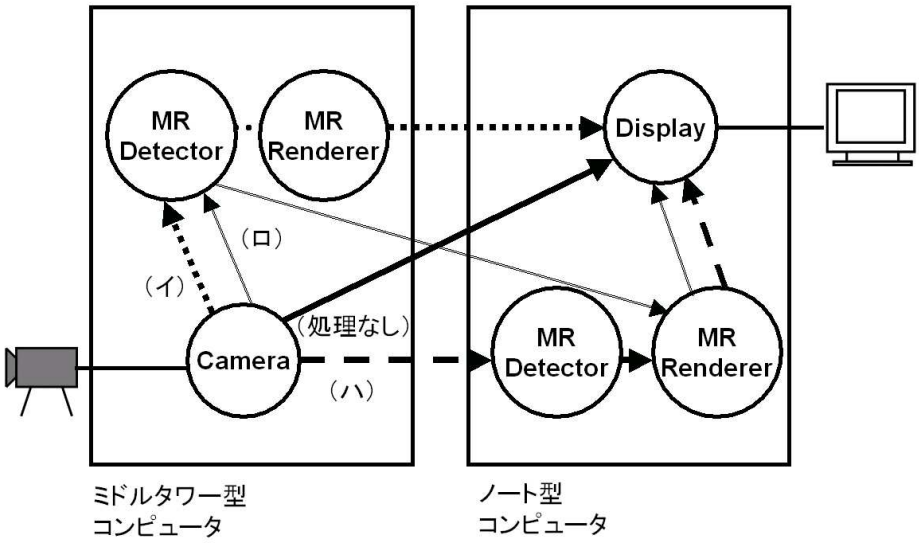

図 12: 接続図

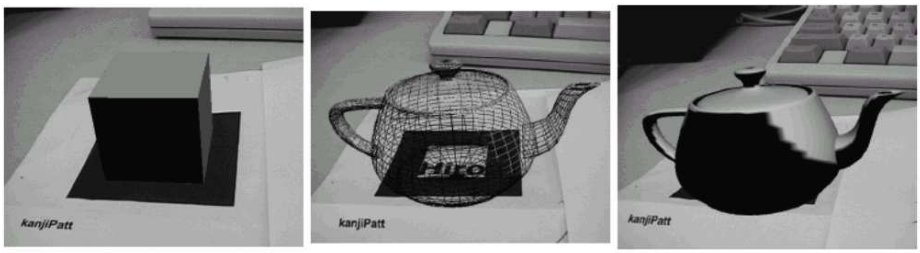

図 13: 描画したオブジェクト

通じてカメラを取り付けており，30fps でのビデオキャプチャが可能である．これらのケースのメディアコンポーネント，マイクロモジュールの接続状態を図にすると，図 12 のようになる．

これらの各条件で，カメラから 30fps で画像を取り込んだ際，ディスプレイ側での表示にかかる，フレーム 1 枚当たりの所要時間を計算した．また，データ量との関係を調べるため，扱うビデオフレームの解像度が 160 × 120，320 × 240，640 × 480 の場合をそれぞれ測定した．さらに，描画するオブジェクトの複雑さによる性能差を比較するために，『立方体図』，『急須（フレーム）』，『急須』の 3 通りのオブジェクト描画を行った (図 13)．

各条件でのパフォーマンスは，表 1，表 2，表 3，表 4 のようになった．

複合現実感処理を利用しない場合は，160 × 120 と 320 × 240 の場合はキャプチャ性能通り，1 フレーム 33ms 程度となっている．しかし 640 × 480 の場合はネットワーク帯域がボトルネックとなり，134ms に延びている．複合現実感処理を行った場合では，（イ）と（ハ）を比べると，いずれのケースでも性能に差が出ている．たとえば立方体を描画した場合，160 × 120 ではほとんど性能に差はないが，320 × 240 では（イ）に比べて（ハ）の処理時間が 213 %，640 × 480 では同 261 %となっており，データ量が大きくなるにつれて，処理時間の差が大きくなっている．また，急須の描画と立方体の描画を比べると，たとえば 320 × 240 の場合で（ハ）が（イ）に比べて立方体では 213 %であるのに比べて急須では 109 %となり，解像度が変化した場合に比べると差は少ない．画像の大きさ，オブジェクトの複雑さとも処理時間に影響を与えているといえ，分散化により様々なケースで利点があると考えられる．

4 開発成果の特徴

本システムの特長は，複合現実感アプリケーションを分散環境で簡単に構築可能とし，ユビキタスコンピューティング環境で容易に同アプリケーションを利用可能とした点である． また，本システムでは，MiRAGe 通信インフラストラクチャによって，アプリケーションの動的適合がサポートされている． これらにより，従来の複合現実感クラスライブラリなどを利用し，分散環境で複合現実感アプリケーションを構築しようとする場合の複雑さを解消している． さらに，本システムでは，ディテクタオブジェクトを入れ換えることによって，多くの種類のビジュアルマーカに対応することが可能である．ビジュアルマーカは，種類によって特徴があり，これらを状況に応じて使い分けられることは非常に有用であると考えられる．

5 今後の課題・展望

マルチメディアフレームワークの改良

マルチメディアフレームワーク（複合現実感クラスライブラリを含む）に関しては，改良すべき点がいくつかある． まず，現在マルチメディアコンポーネントの間では，非圧縮のビデオフレームデータを流しているため，限られた条件以外では，これがアプリケーション全体のパフォーマンスに関してボトルネックになってしまう．そのために，ビデオフレームデータの圧縮を考えている．方法としては，本フレームワークではフレーム一枚ごとにフィルタにイメージ処理を行う可能性があるため，フレーム毎の圧縮が望ましく，MJPEG のような方法が適していると考えられる．しかし，本フレームワークではストリームは複数のフィルタで画像処理を行うことも可能であり，この場合

解像度	所要時間
160 × 120	33.5
320 × 240	33.5
640 × 480	134

表 1: 性能評価（MR 処理なし）

	(イ)	(ロ)	(ハ)
160 × 120	33.8	34.6	42.2
320 × 240	99.5	93.8	104
640 × 480	321	349	361

表 3: 性能評価（フレーム急須を描画）

	(イ)	(ロ)	(ハ)
160 × 120	34.0	34.0	33.5
320 × 240	34.1	38.9	72.6
640 × 480	134	232	350

表 2: 性能評価（立方体を描画）

	(イ)	(ロ)	(ハ)
160 × 120	44.3	80.3	82.4
320 × 240	128	135	140
640 × 480	368	449	456

表 4: 性能評価（急須を描画）

圧縮解凍の繰り返しで画質が低下してしまう．パフォーマンスや画像品質のバランスを考えた方法を検討する必要がある． 次に，複合現実感ツールキットにおいて，現在検出元のビデオフレームに付加してコンポーネント間でやりとりしているマーカ情報を，分離して単独のストリームとすることを考えている．これにより，空間的に離れた場所同士で，同一の複合現実感空間を共有することが可能となる． なお，これらの機能拡張によって利用においての複雑さが増すことが考えられるが，それを軽減する努力もまた必要である．

アプリケーション開発

本システムではミドルウェアを作成したが，今後は有用なアプリケーションを考えることも必要である． また，現在の MiRAGe 通信インフラストラクチャは研究色の強いものであるが，これを実用的になるよう精錬する必要もある．

6 学会等における成果の発表

本システムの開発成果は，以下の学会または展示会で発表を行った，あるいは行う予定である．

- 第 4 回 組込みシステム技術に関するサマーワークショップ (SWEST4) 浜松市 2002 年 7 月（査読無し）
「拡張現実環境を実現する分散マルチメディアミドルウェア」
- ソフトウェア科学会第 19 回大会 東京都 2002 年 9 月 (査読無し)
「拡張現実環境を実現する分散マルチメディアミドルウェア」
- The 9th International Conference on Real-Time & Embedded Computing System and Applications (RTCSA2003) 台湾，台南市 18-20 Feb 2003
“System support for distributed augmented reality in ubiquitous computing environment”
- 日本ソフトウェア科学会第 6 回プログラミングおよび応用のシステムに関するワークショップ (SPA2003) 神奈川県箱根町 2003 年 3 月
「ユビキタス環境における複合現実感のためのミドルウェア」
- The 6th IEEE International Symposium on Object-oriented Real-time distributed Computing(ISORC2003) 函館市 2003 年 5 月
“Object-Oriented Middleware Infrastructure for Distributed Augmented Reality”

参考文献

[1] ARToolkit, http://www.hitl.washington.edu/people/grof/SharedSpace/Download/ARToolKitPC.htm.

[2] Azuma, Ronald T. A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments 6, 4 (August 1997), 355 - 385. Earlier version appeared in Course Notes #9: Developing Advanced Virtual Reality Applications, ACM SIGGRAPH ’95 (Los Angeles, CA, 6-11 August 1995), 20-1 to 20-38.

[3] G.D. Abowd, E.D. Mynatt, “Charting Past, Present, and Future Research in Ubiquitous Computing”, ACM Transaction on Computer-Human Interaction, 2000.

[4] Hal コーポレーション, Crossam2+, http://www.across.or.jp/e_sugi/

[5] D.Lopez de Ipina, “Visual Sensing and Middleware Support for Sentient Computing”, PhD thesis, Cambridge University Engineering Department, January 2002

[6] 森元 靖庸 『柔軟なコンテキストアウェアネスを実現する適応可能な環境情報データベース』早稲田大学大学院理工学研究科修士論文 2003 年 2 月

[7] T.Nakajima, H.Ishikawa, E.Tokunaga, F. Stajano, “Technology Challenges for Building Internet-Scale Ubiquitous Computing”, In Proceedings of the Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems, 2002.

[8] OMG, “The Common Object Request Broker Architecture: Architecture and Specification”, October 1999

[9] OpenGL http://www.opengl.org/

[10] K.Raatikainen, H.B.Christensen, T.Nakajima, “Applications Requirements for Middleware for Mobile and Pervasive Systems”, Mobile Computing and Communications Review, Octorber, 2002.

[11] M. Weiser, “The Computer for the 21st Century”, Scientific American, Vol. 265, No.3, 1991.

[12] X10 http://www.x10.com