卒業論文　　2002年度(平成14年度)

# カードのメタファを用いた<br>ユーザインタフェースの動的合成機構


**指導教員**

慶應義塾大学環境情報学部

徳田 英幸

村井 純

楠本 博之

中村 修

南 政樹

慶應義塾大学 環境情報学部

門田 昌哉

# 卒業論文要旨 2002 年度 (平成 14 年度)

# カードのメタファを用いた<br>ユーザインタフェースの動的合成機構


**論文要旨**

ユビキタスコンピューティング環境は，多様なサービスと多様な制御用端末から構成される．制御用端末としては小型携帯端末や環境に埋め込まれたディスプレイが挙げられ，サービスとしては白物家電機器や AV 機器等が挙げられる．同環境において，ユーザは任意の制御用端末上にサービスからユーザインタフェース記述をダウンロードし，動的に生成されるユーザインタフェースを利用してサービスを制御する．しかし，ユーザインタフェースとサービスの関係は一対一であり，複数のサービスを同時に制御可能なユーザインタフェースを生成できない問題点がある．

本論文では，サービスを分散コンポーネントの集合として定義する．サービスが提供する機能が分散コンポーネントであり，異なるサービスを構成する分散コンポーネントをネットワークを介して協調動作させることをサービス合成と呼ぶ．サービス合成機構は，複数のサービスの構成要素からなるサービスの実体を構築するための基盤ソフトウェアである．既存のサービス合成機構の中には，分散コンポーネント間の関係を動的に定義可能な機構もあるが，高度な専門知識を必要とするため，ユーザの利用に適さない問題点がある．これらのサービス合成機構は，ユーザが分散コンポーネントを制御するための視覚的なフロントエンドを提供しない．

本論文では，既存のサービス合成機構の概念に基づいて分散コンポーネント間の関係を動的に定義し，ユーザインタフェースを動的に合成する機構 **CUICs (Card-oriented User Interface Composition System)** を提案する．CUICs では，集約合成規則と接続合成規則の 2 種類の分散コンポーネント間の合成規則を用いる．ユーザインタフェースの合成とは，これらの合成規則の適用結果に従って個々の分散コンポーネントを制御する GUI コンポーネントを合成する処理を指す．CUICs ではユーザインタフェースをカードとして表示することにより，ユーザはカードを重ねることによってユーザインタフェースを合成できる．これにより，ユーザは複数のサービスを動的に組み合わせて生成されるサービスを簡易に制御できる．



**キーワード：**

1 分散コンポーネント，2 サービス合成機構，3 ユーザインタフェース，4 カード型インタフェース，5 エキスパートシステム



**慶應義塾大学　環境情報学部 環境情報学科**
**門田 昌哉**

# Abstract of Bachelor's Thesis

# Dynamic composition of user interface using card metaphor

## Academic Year 2002

**Summary**

An ubiquitous computing environment consist of various services and controlling devices. Controlling devices include compact portable devices or displays, and services inculude consumer electronic devices or A/V devices. In this environment, a user downloads the user interface description from a service to an arbitrary controlling device. The user may control the service by using the user interface dynamically generated from the description. Since the relationship between an user interface and service is one-on-one, users cannot control services simultaneously.

In this research, individual services are defined as a collection of distributed components. Functions which services provide correspond to distributed components, which all collaborate over the network. We define service composition as making services collaborate by inter-connecting distributed components. There are existing service composition mechanisms, where collaborative behavior between distributed components can be defined dynamically. Unfortunately, it is difficult for users to use these mechanisms because it requires a high degree of specialized knowledge. These mechanisms provide generation of communication pass and data flow control between distributed components, but do not provide end users with the user interface to control the component.

This paper proposes **CUICs (Card-oriented User Interface Composition System)**, a service composition mechanism based on concepts of existing mechanisms, where user interface is generated dynamically. Composition of user interface means composition of user interface descriptions which control distributed components, and generation of user interfaces as a conceptual service substance of distributed components that collaborate with other components. CUICs defines collaborative behavior of distributed components dynamically from the services specifed by the user with card style interfaces. These actions are performed independent of the service composition mechanism. As a result, users can easily control several services at once using user interfaces that can control collaborative distributed components.

**Keyworkds:**

1 Distributed Component, 2 Service Composition Mechanism, 3 User Interface,
4 Card-oriented User Interface Manager, 5 Expert System

**Keio University Faculty of Environmental Information**
**KADOTA Masaya**

# 目 次

# 図 目 次

# 表 目 次

# 第 1 章

# 序論

## 1.1 研究背景と問題意識

本節では，本研究の対象環境であるユビキタスコンピューティング環境について述べ，同環境において本研究が問題領域とするサービス合成機構，サービス制御機構について述べる．

### 1.1.1 想定環境

ユビキタスコンピューティング環境は多様なサービスによって構成される．本論文において，**サービス**とは**分散コンポーネント**の集合を指し，個々の分散コンポーネントがサービスが提供する機能に対応する．ホームネットワークにおいて，分散コンポーネントは特定のデバイス上で動作するため，サービスは通常単一の計算機上に配置される．ホームネットワークでは，多様な白物家電機器や AV 機器等のデバイスがネットワークを介して相互に接続され，それらのデバイスに組み込まれた計算機上でソフトウェアが動作する．近年の計算機の小型化により，ノート PC に限らず，ユーザは携帯電話や PDA(Personal Digital Assistant) 等の小型端末を携帯し，環境に埋め込まれた計算機資源を利用して多様なサービスを制御できる．たとえば，ユーザは小型端末を用いて遠隔地から家庭のエアコンを制御したり，至近のディスプレイを利用して DVD プレイヤから任意のディスプレイに映像を表示させたりする．本論文が想定するユビキタスコンピューティング環境の概念図を図 1.1 に示す．図 1.1 では，PDA を用いてデジタルビデオカメラを制御し，プリンタを携帯電話を用いて制御している．ここでは，デジタルビデオカメラの白黒画像出力機能とプリンタの白黒印刷機能，およびカラー画像出力機能とカラー印刷機能との協調動作の例を挙げている．

### 1.1.2 既存のサービス合成機構と問題点

同環境において，サービスは単体で動作するだけでなく，複数のサービスがネットワークを介して協調動作する．実際には，協調動作する実体は各サービスを構成する分

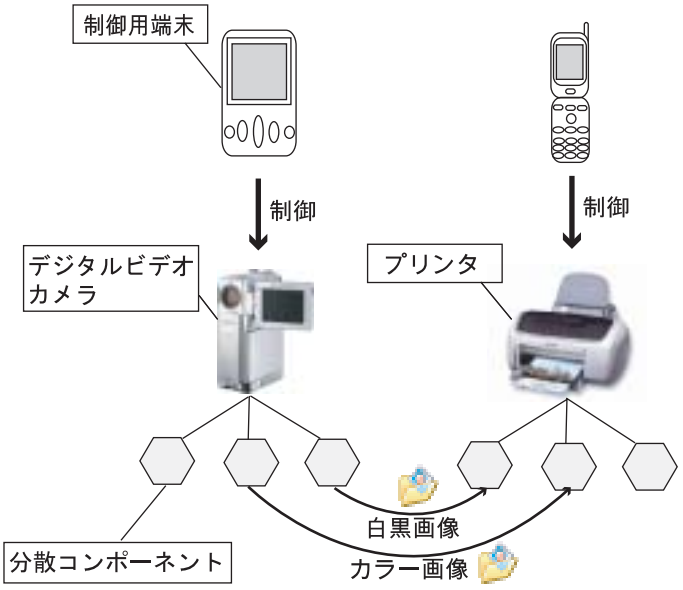



図 1.1: 想定するユビキタスコンピューティング環境の概念図

散コンポーネントである．たとえば，デジタルビデオカメラとプリンタの二つのサービスを協調動作させる場合，デジタルビデオカメラのキャプチャ機能とプリンタの印刷機能が協調動作する．各サービスはキャプチャ機能や印刷機能を分散コンポーネントとして実装する．これらの分散コンポーネント間の通信パス，およびデータフロー管理を行うミドルウェアを**サービス合成機構**と呼ぶ．この場合，サービス合成機構はキャプチャ機能と印刷機能間に通信パスを生成し，キャプチャ機能によって生成される画像データをプリンタ機能に送信する．このような分散コンポーネント間の関係の定義は，分散コンポーネントの入出力データ型や役割の型などの**メタ情報**を用いて行われる．本論文では，このような分散コンポーネント間の関係の定義方法を**関係定義モデル**と呼び，分散コンポーネント間の関係を**合成ルール**と呼ぶ．

既存のサービス合成機構の関係定義モデルでは，合成ルールは分散コンポーネントが構成するサービス同士の関係を考慮せずにフラットに定義される．たとえば，既存のサービス合成機構では，印刷機能を提供する分散コンポーネントという分類はあっても，それがどのサービスに関係しているかは考慮されない．このような関係定義モデルを，本論文では**ボトムアップ**な関係定義モデルと呼ぶ．フラットに分散コンポーネント間の関係を定義することにより，柔軟に分散コンポーネント間の関係を定義できる反面，サービスの粒度での分類を関係定義モデルに反映できない．本研究が対象とするホームネットワークにおいて，ユーザの制御対象はサービスであり，ユーザインタフェースもサービスの単位で記述される．そのため，既存のサービス合成機構のボトムアップな関係定義モデルは，ユーザが制御対象として認識する粒度との間に不一致がある．ホームネットワークにおけるサービス合成機構としては，ユーザが実行時に合成するサービスを択し，それらのサービスから合成可能な分散コンポーネント

を求める**トップダウン**な関係定義モデルが必要である．

### 1.1.3　既存のサービス制御機構と問題点

本研究が想定する環境において，ユーザは環境に埋め込まれたディスプレイや，手元の小型携帯端末を用いてサービスを制御する．本論文では，このようなサービス制御に用いる端末を**制御用端末**と呼ぶ．制御用端末としては，計算機に接続されたプラズマディスプレイや PDA，携帯電話等の画面表示能力のあるデバイスがある．制御に必要なユーザインタフェースの記述は，サービスから動的に制御用端末にダウンロードする．ヘテロジニアスな制御用端末に対応するため，ユーザインタフェースは特定のプログラミング言語に依存しない形式で記述される．たとえば HTML では GUI コンポーネントを記述する際に特定の GUI ツールキットに依存せずに

```
<input type="submit">
```

の様に記述する．制御用端末への非依存性は，ユーザインタフェースの記述自身の抽象度を上げ，個々の制御用端末上のレンダリングソフトウェアによってユーザインタフェースを生成することで実現できる．本論文では，ユーザインタフェースを実行時に生成し，サービスを制御する機構を**サービス制御機構**と呼ぶ．既存のサービス制御機構では，ユーザインタフェースを独自言語を用いて記述し，制御用端末上に配置したレンダリングソフトウェアを用いてユーザインタフェースを生成することが多い．

しかし，既存のユーザインタフェース記述言語は制御用端末のヘテロジニティに対応していても，個々のサービスとは一対一の関係であるため，複数のサービスを同時に制御できない問題点がある．サービス合成機構によって異なるサービスを構成する分散コンポーネント間の通信パスを生成しても，機能の実行自体はユーザが個々の分散コンポーネントを制御するユーザインタフェースを用いて制御する必要がある．そのため，協調動作する分散コンポーネントの組み合わせをそのまま制御可能なユーザインタフェースを動的に生成する機構が必要である．

## 1.2　研究目的と概要

本論文では，カードのメタファを用いたユーザインタフェースの動的合成機構 **CUICs (Card-oriented User Interface Composition System)** を提案する．CUICs は，既存のサービス合成機構のボトムアップな分散コンポーネント間の関係定義モデルの問題を解決するため，ユーザが任意に指定するサービスから協調動作可能な分散コンポーネントの関係を自動的に生成するトップダウンな関係定義モデルを採用する．CUICs において，サービスを制御するユーザインタフェースはカードに抽象化され，ユーザは**カードを重ねる**という簡易な操作によって合成するサービスを選択する．図 1.2 に CUICs の画面を示す．実際に協調動作可能な分散コンポーネントは，サービスを構成する分散コンポーネントに設定されたメタ情報を用いて自動的に定義される．CUICs では，分散コンポーネント間の関係を **1) 接続合成，2) 集約合成**の二つに分類する．接続合成と

は，データの入出力を介して分散コンポーネントが協調動作する関係であり，集約合成とは，同様な機能を持った分散コンポーネントが並列に動作する関係である．接続合成の例としては，DVD プレーヤの音声出力を任意のスピーカに出力したり，CS チューナの映像出力を任意のディスプレイに表示する関係が挙げられる．集約合成の例としては，ホームネットワーク上の全てのサービスが持つ電源管理機能を集約するような関係が挙げられる．

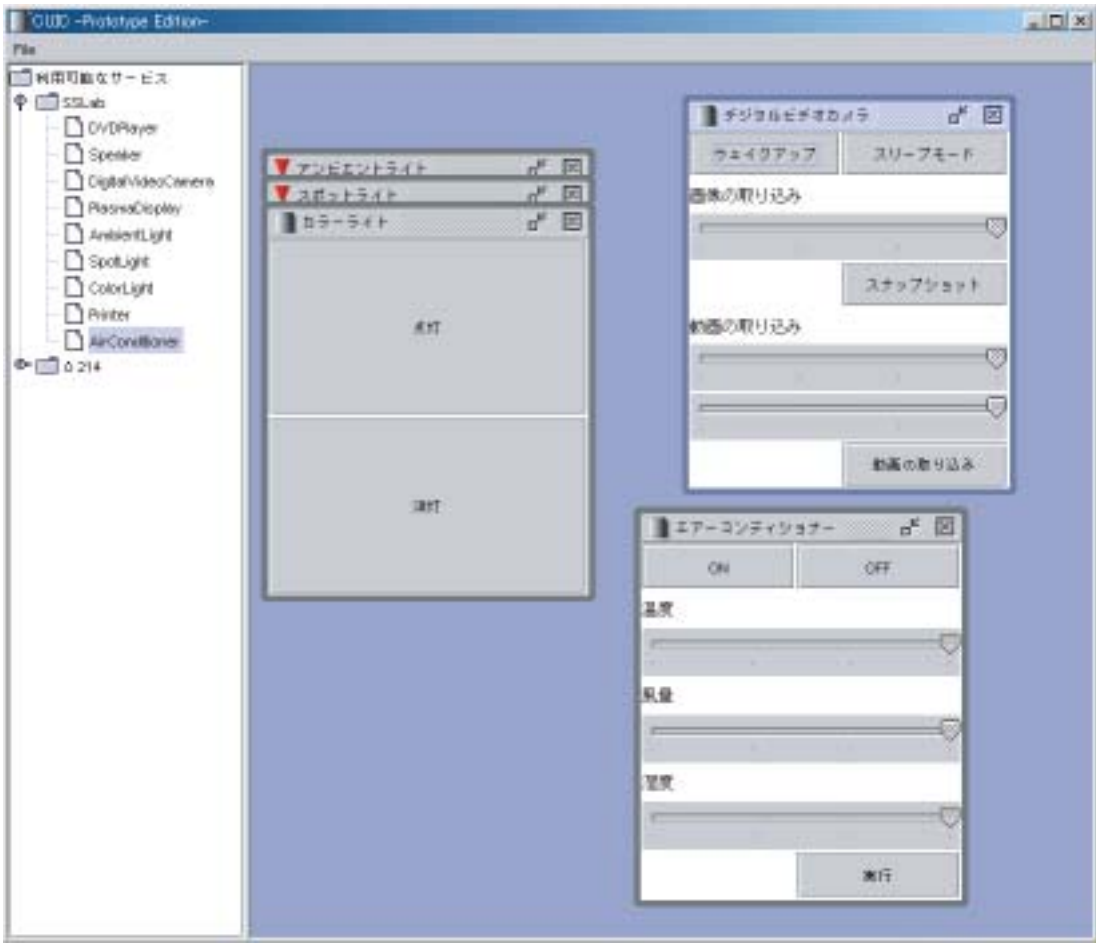



図 1.2: 提案する CUICs の画面

CUICs は分散コンポーネント間の関係に従って，個々の分散コンポーネントを制御する GUI コンポーネントの集合を合成し，協調動作する分散コンポーネント全体を制御可能なユーザインタフェースを生成する．本論文では，このような分散コンポーネントの関係に基づいて GUI コンポーネントの集合を合成することを**ユーザインタフェースの合成**と呼ぶ．ユーザインタフェースの合成により，ユーザは協調動作する分散コンポーネントを個別に制御せずに，単一のユーザインタフェースを用いて複数の分散コンポーネントを制御可能になる．ユーザインタフェースの合成を実現するため，CUICs ではサービスが提供する機能単位でユーザインタフェースを記述するユーザインタフェー

ス記述言語 **CUIL (Composable User Interface Language)** を提案する．既存のユーザインタフェース記述言語に対し，CUIL はサービスを構成する分散コンポーネント単位で GUI コンポーネントを記述するため，協調動作する複数の分散コンポーネントを制御可能なユーザインタフェースを記述できる．CUICs は，CUIL を動的に合成することにより，協調動作を制御可能なユーザインタフェースを生成する．

## 1.3 本論文の構成

本論文の構成は以下の通りである．第 2 章では既存のサービス合成機構を概観し，分散コンポーネント間の関係の定義モデルの分類を行う．これらの分類から，CUICs が対象とするホームネットワークに適した関係定義モデルを明確にする．第 3 章では，既存のサービス合成機構で採用されている関係定義モデルを概念的基礎として，協調動作する分散コンポーネントを制御可能なユーザインタフェースを生成する機構について述べる．第 4 章では，このようなユーザインタフェースの合成を実現するユーザインタフェース記述言語に求められる要件を考察し，本研究が提案する CUIL の仕様を詳細に述べる．第 5 章では，CUICs の設計方針および全体の構成について述べ，ホームネットワークに適したユーザインタフェース合成機構を定義する．第 6 章では，CUICs のプロトタイプシステムの実装について述べ，第 7 章では，プロトタイプシステムの基本性能の評価，およびボトルネックの検証を行うと共に，関連する研究との性質面での比較を行う．第 8 章で本論文をまとめ，結論を述べる．

# 第 2 章

## サービス合成機構の分類

本研究が提案するユーザインタフェースの動的合成機構 CUICs は，既存のサービス合成機構の概念に基づく．サービス合成機構は，コンポーネント指向ソフトウェア開発技術や分散コンポーネント技術を背景とする．本章では，はじめに分散コンポーネント技術を概観し，サービス合成機構に求められる要件を考察する．その後，サービス合成機構を分散コンポーネント間の関係定義モデルによって分類する．最後に本研究が対象とするホームネットワークの特徴を考察し，CUICs で採用するトップダウンな関係定義モデルの要件を定義する．

## 2.1 サービス合成機構の概観

サービス合成機構として，WEB サービスを対象とした SWORD[12]，ホームネットワークを対象とした VNA[10, 9]，広域分散ネットワークを対象とした Ninja[16]，Floch 氏等が提案するシステム [4] が挙げられる．本節では，これらのサービス合成機構の背景となる分散コンポーネント技術を概観し，サービス合成機構の要件を考察する．

### 2.1.1 分散コンポーネント技術

ソフトウェアを機能単位で部品に分解し，部品を組み合わせることによってソフトウェアを開発する手法を**コンポーネント指向ソフトウェア開発**という．コンポーネントはカプセル化されており，コンポーネント間の通信は個々のコンポーネントのインタフェースを介して行う．ここでのインタフェースは，ユーザインタフェースではなく，アプリケーションプログラミングインタフェース (API) を指す．この概念を実装したシステムとして，ActiveX/COM[3]，JavaBeans[19] がある．

コンポーネントの概念を分散環境において利用する場合，これを特に**分散コンポーネント**と呼ぶ．分散コンポーネントは，先に述べたコンポーネントの概念にネットワーク接続性を加え，異なる計算機上のコンポーネント間の通信を実現するものである．分散コンポーネントの実装としては，Microsoft 社の DCOM(Distributed Component Model)[2] や EJB(Enterprise JavaBeans)[21] が挙げられる．

図 2.1 に本論文における分散コンポーネントの概念図を示す．個々の分散コンポーネントは，入力データ型 (Input Type)，出力データ型 (Output Type)，役割の型 (Role Type)，属性情報 (Attributes) を持つ．たとえば，CORBA IDL[5] におけるメソッド名が Role Type，引数が Input Type，返り値が Output Type にあたる．Attributes としては，分散コンポーネントの位置情報や，他の分散コンポーネントとの階層情報などが挙げられる．

### 2.1.2 サービス

本論文において，**サービス**とは分散コンポーネントの集合を指し，個々の分散コンポーネントがサービスの提供する機能に対応する．通常，サービスは単一の計算機上に配置される．サービスの例としては，冷蔵庫や電子レンジなどの白物家電機器，DVD プレーヤやスピーカ等の AV 機器を制御するソフトウェアが挙げられる．図 2.2 の例では，DVD プレーヤは ON/OFF の電源管理機能に加え，再生機能，停止機能，録画機能，早送り機能，巻き戻し機能の 5 つの機能を提供しており，それぞれが一つの分散コンポーネントに対応する．

### 2.1.3 サービス合成機構の概要

**サービス合成機構**とは，分散コンポーネント間の関係を定義し，分散コンポーネント間のデータ入出力を行うための通信パスの生成，および通信パス上のデータフローを管理するミドルウェアである．分散コンポーネント間の関係定義モデルは個々のサー

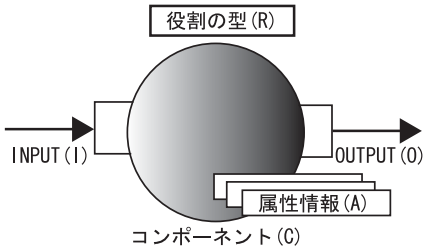



図 2.1: 分散コンポーネント概念図

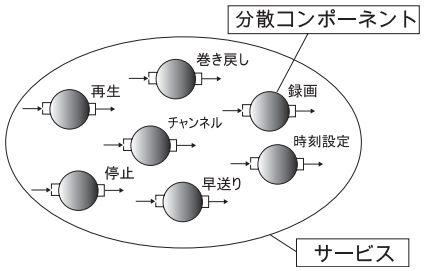



図 2.2: サービス概念図

ビス合成機構によって異なり，それによって実現される分散コンポーネント間の関係も異なる．Output Type や Input Type 等のメタ情報を用いて分散コンポーネント間の関係を定義するモデルでは，分散コンポーネント間の入出力データを接続する協調動作を実現できる．Input Type，Output Type を用いた協調動作の例としては，画像データ等の特定のデータを介した協調動作が挙げられる．一方，Role Type を用いて分散コンポーネント間の関係を定義するモデルでは，分散コンポーネント自体の型に従って分散コンポーネントを関係付けるため，入出力データを接続するだけでなく，任意の分散コンポーネント間の関係を定義可能である．そのため，入出力を伴わない任意の分散コンポーネントを順次実行するような協調動作も定義可能である．Role Type を用いた協調動作としては，電源管理などの複数のサービスに共通する機能の並列実行や，DVD プレーヤの再生機能と部屋の照明の調節などの協調動作が考えられる．これらの関係定義モデルでは，分散コンポーネント間の関係の記述言語が必要である．多くの場合，個々のサービス合成機構が独自の関係記述言語を提案しているが，WSFL[13] や XLang[22] 等の一般的な関係記述言語も提案されている．本論文では，分散コンポーネント間の関係を定義するプロセス，および通信パスを生成するプロセスを含めて**サービス合成**と呼ぶ．図 2.3 にサービス合成機構の概念図を示す．

図 2.3(a) は分散コンポーネントのメタ情報の関係を示している．白色の六角形がそれぞれ分散コンポーネントのメタ情報を示しており，サービスはこれらのメタ情報を持つ分散コンポーネントの集合によって構成される．外向きの三角形が Output Type を示し，内向きの三角形が Input Type を示す．また，内部の六角形が Role Type を示す．分散コンポーネント間の関係は，これらのメタ情報から定義できる．三角形間を結ぶ曲線が Input Type, Outut Type を用いた分散コンポーネント間の関係であり，六角形間を結ぶ直線が Role Type を用いた分散コンポーネント間の関係を示す．図 2.3(b) は分散コンポーネントの実体の関係を示している．灰色の六角形が分散コンポーネントの実体である．分散コンポーネントをつなぐ線は実際にデータの送受信を行うための通信パスであり，線上の四角形がデータパケットを示している．サービス合成機構は，Input Type と Output Type の一致や，Role Type の一致に基づいて分散コンポーネントの実体間の関係を定義し，分散コンポーネント間の通信パスの生成およびデータフローの制御を行う機構である．分散コンポーネント間の関係定義モデルは，(a) の白色の六角形とその関係から (b) の実体を導くテンプレート主導モデルと，(b) の実体の集合から (a) の情報を用いて分散コンポーネント間の関係を求めるルール生成モデルがある．次節

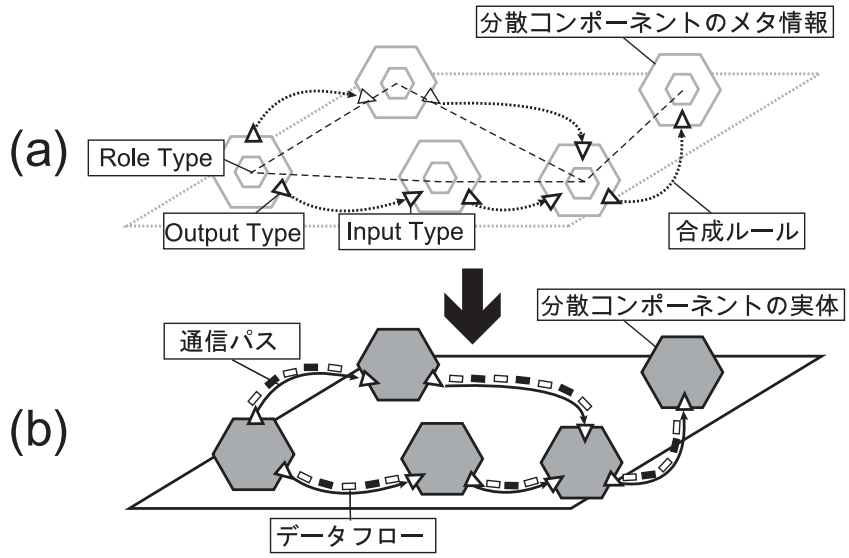



図 2.3: サービス合成機構の概念図

でこれらの関係定義モデルの分類を詳細に行う．

## 2.2 分散コンポーネントの関係定義モデルによる分類

本節では，サービス合成機構における分散コンポーネント間の関係定義モデルの分類を行う．関係定義モデルとは，分散コンポーネント間の関係を求めるための方法を指す．関係定義モデルは，大きく分けてルール生成モデル，テンプレート主導モデルの 2 つに分類できる．1.2 節で述べたトップダウンな関係定義モデルがルール生成モデルにあたる．ルール生成モデルは，与えられた分散コンポーネントの集合から関係を導くモデルである．また，テンプレート主導モデルは，あらかじめ定義されたテンプレートに分散コンポーネントの実体を当てはめるモデルである．ルール生成モデルに対して，テンプレート主導モデルでは動的に分散コンポーネントの実体を取得するため，分散コンポーネントのディスカバリの処理が必要になる．

### 2.2.1 ルール生成モデル

ルール生成モデルの概念図を図 2.4 に示す．分散コンポーネントを示す灰色の六角形の間の直線が合成ルールである．任意の分散コンポーネントの実体の集合を与えられたとき，分散コンポーネントのメタ情報から合成ルールを生成するモデルがルール生成モデルである．図では $\{A_1, A_2, B\}$ が分散コンポーネントの実体の集合を示し，個々の分散コンポーネントの外向きの三角形が Output Type，内向きの三角形が Input Type を示している．たとえば，$A$ の Output Type と $B$ の Input Type が一致し，$B$ は複数のデー

タを入力として受け取る場合，生成されるルールは $\{A'_1, A'_2\} \rightarrow \{B'\}$ となる．$\rightarrow$ がルールにあたり，$A_{1,2}$ の出力データを $B$ の入力データに出力する関係である．分散コンポーネント間の $\rightarrow$ を求めるモデルであるため，ルール生成モデルと呼ぶ．

ルール生成モデルでは，サービスを構成する分散コンポーネントをあらかじめ定義するのではなく，分散コンポーネント間のルールを動的に生成するため，任意の分散コンポーネントの集合を構成要素とするサービスを生成できる利点がある．反対に，無意味なルールを生成したり，ルール自体を生成できない可能性がある．こういった可能性を減少させるには，入出力データ型のセマンティクスを考慮する必要がある．そのため，ルール生成モデルでは後述するテンプレート主導モデルに対して，より詳細に Input/Output Type や Role Type の型付けを行う必要がある．

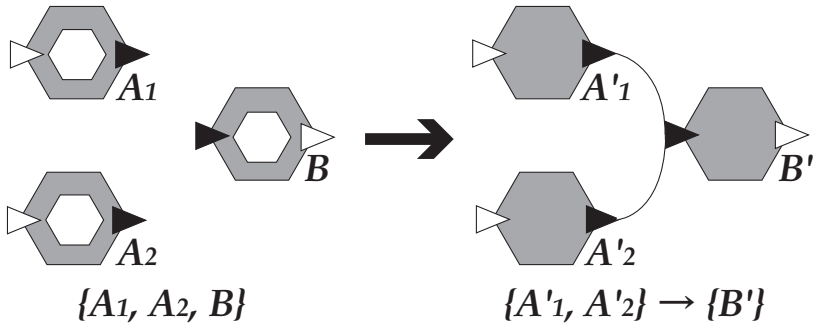



図 2.4: ルール生成モデル概念図

このようなルール生成モデルを採用しているサービス合成機構として SWORD が挙げられる．SWORD はウェブサービスを対象としたサービス合成機構である．個々の分散コンポーネントは Input Type と Output Type のデータ型を持ち，これらの入出力データから，エキスパートシステム [14] に必要な事前条件，およびルールを生成する．SWORD では，入力として名前を受け取って住所を出力する分散コンポーネント $A_{1,2}$，入力として 2 箇所の住所を受け取って現在地からの方向を求める分散コンポーネント $B$ があった場合，$\{A_1, A_2, B\}$ の分散コンポーネントの集合から，入力として二人分の人名を受け取り，方向を出力するサービスを合成する例を挙げている．この場合，入出力データから分散コンポーネント間の関係を動的に求めることができるが，入力値を単なる文字列ではなく，名前や住所といった意味のある単位で扱っている．

### 2.2.2 テンプレート主導モデル

テンプレート主導モデルの概念図を図 2.5 に示す．分散コンポーネント間の合成ルールが静的に与えられていて，そのテンプレートに分散コンポーネントを当てはめるモデルがテンプレート主導モデルである．たとえば，$\{A_1, A_2, B\}$ の分散コンポーネントのメタ情報の集合と $\{A'_1, A'_2, B'\}$ の分散コンポーネントの実体の集合があるとする．このとき，$\{A_1, A_2\} \rightarrow \{B\}$ から，$A_{1,2}$ と $B$ のメタ情報の型を持つ分散コンポーネントの実

体を当てはめ，$\{A'_1, A'_2\} \rightarrow \{B'\}$ を求めるモデルがテンプレート主導モデルである．分散コンポーネントの型と合成ルールがあらかじめ与えられているため，テンプレート主導モデルと呼ぶ．

分散コンポーネントの実体は，システムが動的に現在利用可能な分散コンポーネントの中から選択する．分散コンポーネントを選択する過程をディスカバリと呼ぶ．サービス合成機構は，選択した分散コンポーネント間の通信パスを動的に生成する．ネットワークの帯域や応答性等のコンテキスト情報から分散コンポーネントの実体を動的に入れ替えることにより，QoS 制御やエラー処理を実現できる．しかし，テンプレートを集めた知識ベースは静的であるため，ユーザが任意の分散コンポーネントから成るサービスを合成できない問題点がある．

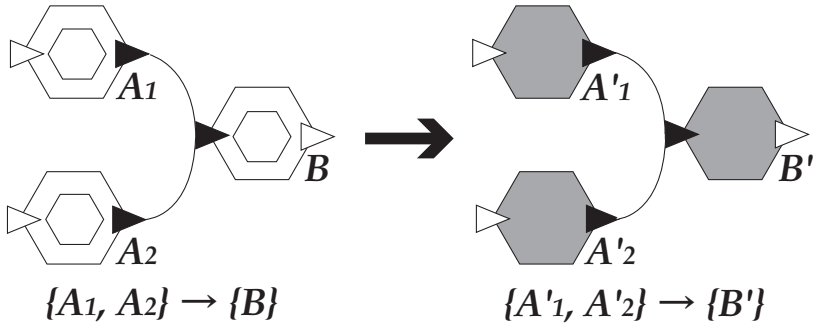



図 2.5: テンプレート主導モデル概念図

テンプレート主導モデルを採用する既存のサービス合成機構として，Ninja や Floch のシステムが挙げられる．Ninja では，分散コンポーネントの入出力データの型を用いてテンプレートを作成する．反対に，Floch のシステムでは，分散コンポーネントに役割の型を与え，それに基づいてテンプレートを作成する．たとえば，Floch のシステムで挙げられている例では，電気通信事業におけるエンドポイントと中継点という役割を分散コンポーネントに割り振ることによって分散コンポーネントの関係を定義する．電気通信事業では，必要な分散コンポーネントの型とルールはあらかじめ静的に定義できる．したがって，テンプレート主導モデルは，あらかじめ必要な分散コンポーネントが自明である分野に有効な関係定義モデルだと言える．

以上のテンプレートベースモデルで利用されるテンプレートの抽象度をさらに上げて，メタ情報をデータのプロバイダ，コンシューマなどの役割とした場合，GOF のデザインパターンにおけるファサードや，コマンドパターンをサービス合成機構で表現できる．これも大きな意味でのテンプレート主導モデルに含まれる．しかし，この場合では分散コンポーネントを一意に決定する情報が少ないため，上に挙げた Role Type や I/O Type 以外の分散コンポーネントの実体を決定するための情報を与える必要がある．

以上の 2 つのモデルをまとめたものを，表 2.1 に示す．関係の変更性の項目は，動的に分散コンポーネントの集合を与えらたとき，分散コンポーネント間の関係を定義可能な性質を示す．代替性の項目は，分散コンポーネントの実体が利用可能でなかった

り，求める性能でない場合の分散コンポーネントの実体の代替性を示す．また，実体の指定の項目は，特定の分散コンポーネント間の関係を定義可能な性質を示す．簡易性の項目は，分散コンポーネント間の関係を求めるためのコストを示す．ルール生成モデルは実体の指定をユーザが行う必要があるためコストが高く，テンプレート主導モデルでは実体の指定までユーザが関わる必要がないためコストが低い．

表 2.1: 関係定義モデルの比較

| | 関係の変更性 | 代替性 | 実体の指定 | 簡易性 |
|---|---|---|---|---|
| ルール生成モデル | ○ | × | ○ | × |
| テンプレート主導モデル | × | ○ | × | ○ |

## 2.3 サービス合成機構の考察

本節では，ルール生成モデルとテンプレート主導モデルの分類から，本研究が対象とするホームネットワークに適したモデルを考察する．はじめにホームネットワークの特徴を挙げ，その後本研究で採用するモデルについて述べる．

### 2.3.1 ホームネットワークの特徴

表 2.1 で述べた関係の変更性，代替性，実体の指定，簡易性に関して，ホームネットワークがどのような特徴を持つかを考察する．

#### 関係の変更性

協調動作の変更性とは，ユーザが動的に協調動作する分散コンポーネントの実体を変更可能であることを指す．ホームネットワークにおいて，協調動作するサービスの組み合わせは動的に変化する．たとえば，昼間は CS チューナの映像出力を居間のディスプレイに接続するが，夜間は 2 階のディスプレイに表示する場合が考えられる．このような場合，ユーザは動的に協調動作する分散コンポーネントを変更可能である必要がある．

#### 実体の指定と代替性

ホームネットワークでは，特定の条件を満たす分散コンポーネントを指定するより，特定のサービスを構成する分散コンポーネントの実体を指定する必要がある．たとえば,「画像をキャプチャする分散コンポーネントの画像出力」を,「画像を入力として受け取る分散コンポーネント」に接続すると指定するのではなく,「玄関にあるデジタルビデオカメラが出力する画像」を「2 階の液晶ディスプレイの入力」に接続すると指定する必要がある．ホームネットワークにおいては，分散コンポーネントの性質よりも，

分散コンポーネントのサービスへの所属の方が重要である．たとえば，動画データを手元のディスプレイに表示したい時，2 階のディスプレイが動画データを表示できても意味をなさない．ホームネットワークでは，分散コンポーネントの実体が重要であり，同様なメタ情報を持つ他の分散コンポーネントとの代替性はあまり求められない．

**簡易性**

ユーザが動的に分散コンポーネント間の関係を変更する際には，関係定義の簡易性が重要である．ユーザに対して個々のサービス合成機構が提案する独自の関係記述言語の文法の理解を求め，テキストエディタを用いて組み合わせを再定義する様な方法は現実的でない．また，前項で述べた様に，ユーザは特定の分散コンポーネントの実体を指定する必要性があるが，この関係を分散コンポーネント間の関係ではなく，分散コンポーネントが構成するサービスの指定から導き出すことができれば簡易性が向上する．ユーザがサービスとその機能を選択するのではなく，サービスを指定するのみで協調動作可能な機能の組み合わせをシステムが自動的に認識するシステムが必要である．

### 2.3.2 本研究が採用する関係定義モデル

2.3.1 で述べたホームネットワークの特徴から，本研究が採用する分散コンポーネント間の関係定義モデルについて考察する．以下に，表 2.2 に，ホームネットワークの特徴と，2.1 に示した分散コンポーネント間の関係定義モデルの比較を示す．

表 2.2: 関係定義モデルとホームネットワークの特徴

| | 協調動作の変更性 | 代替性 | 実体の指定 | 簡易性 |
|---|---|---|---|---|
| **ホームネットワークの特徴** | ○ | **×** | ○ | ○ |
| ルール生成モデル | ○ | × | ○ | × |
| テンプレート主導モデル | × | ○ | × | ○ |

本研究では，分散コンポーネント間の関係定義モデルとしてルール生成モデルを採用する．ホームネットワークでは分散コンポーネントの実体の指定が必要であり，ユーザが動的に協調動作する分散コンポーネントを変更可能である必要がある．テンプレート主導モデルでは，協調動作する分散コンポーネントの型が静的に定義されるため，動的に環境に追加されるサービスに対応できない．しかし，簡易性という観点からは，ルール生成モデルはテンプレート主導モデルに対して劣る問題点がある．ルール生成モデルの問題点は，既存のサービス合成機構がユーザに対して簡易に協調動作する分散コンポーネントを指定するフロントエンドを提供していないためである．本研究では，この問題点を解決するために，分散コンポーネントの実体を簡易に指定可能なカード型インタフェースを提案する．これにより，ルール生成モデルがテンプレート主導モデルに対して劣る簡易性を埋められる．

## 2.4　本章のまとめ

本章では，はじめに既存のサービス合成機構を概観し，分散コンポーネント間の関係定義モデルの観点から分類した．その後，ホームネットワークの特徴を考察し，本研究が採用するトップダウンな分散コンポーネント間の関係定義モデルとしてルール生成モデルを採用する理由を述べた．

# 第 3 章

# ユーザインタフェース合成機構

本章では，本論文で提案するユーザインタフェース合成機構について述べる．ユーザインタフェース合成機構とは，前章で述べたルール生成モデルに基づき，複数の分散コンポーネントを制御可能なユーザインタフェースを動的に生成する機構である．本論文では，サービスが提供する機能の単位で GUI コンポーネントを定義し，分散コンポーネント間の関係定義と同様にユーザインタフェースを合成する．個々の分散コンポーネント間の関係として，接続合成規則と集約合成規則を定義し，分散コンポーネント間のデータの入出力に基づく協調動作や，役割の型に基づく分散コンポーネントの並列実行などの協調動作を制御可能なユーザインタフェースを動的に生成する．

## 3.1 ユーザインタフェース合成機構の概要

本論文では，第 2 章で述べたルール生成モデルによって生成される分散コンポーネント間の関係に基づき，個々の分散コンポーネント毎に定義される GUI コンポーネントの集合を合成することを**ユーザインタフェースの合成**と呼ぶ．ルール生成モデルによって生成可能な分散コンポーネント間の関係としては，**(1) 接続合成規則，(2) 集約合成規則**が考えられる．接続合成規則は，分散コンポーネントの Input Type と Output Type の一致に基づく関係であり，集約合成規則は Role Type の一致に基づく関係である．本論文で提案するユーザインタフェース合成機構は，これらの分散コンポーネント間の関係に従い，個々の分散コンポーネントを制御する GUI コンポーネントの合成を行う．分散コンポーネント間の関係と，GUI コンポーネントの集合間の関係を図 3.1 に示す.

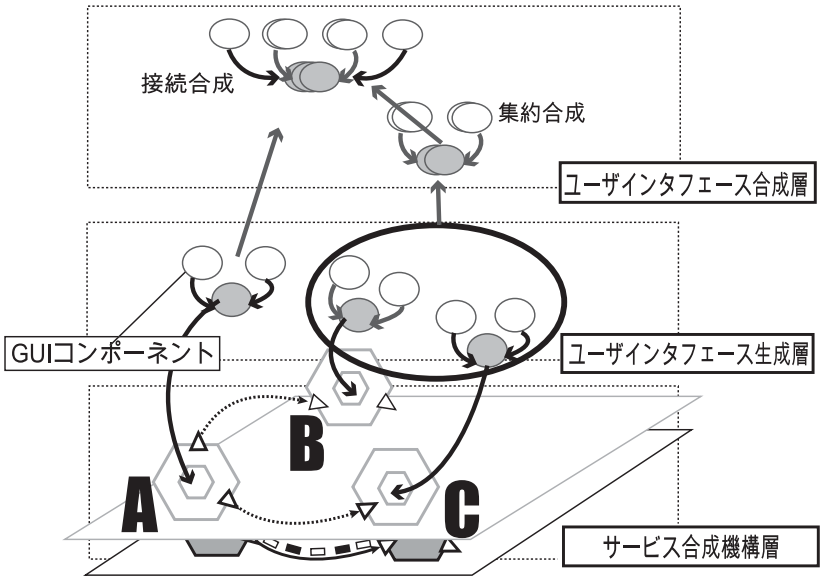



図 3.1: サービス合成とユーザインタフェース合成の関係

図 3.1 に示すサービス合成機構の層は，メタ情報を用いて分散コンポーネント間の関係を定義し，分散コンポーネント間の通信パスの生成を行う層である．ユーザインタフェース生成層は，個々の分散コンポーネントに対して記述される GUI コンポーネントの集合から，実際にユーザインタフェースを生成する層である．ユーザは，この層で生成されるユーザインタフェースを用いて引数を設定し，分散コンポーネントに対してコマンドを発行する．ユーザインタフェース合成層は，サービス合成層で定義される分散コンポーネントの関係に基づき，個々の分散コンポーネントを制御する GUI コンポーネントの合成を行う．これにより，ユーザは複数の分散コンポーネントを単一の GUI コンポーネントの集合を用いて制御可能になる．図 3.1 では，中央の分散コンポーネント B，C を制御する GUI コンポーネントの集合に集約合成規則を適用し，生

成されたGUIコンポーネントの集合と分散コンポーネントAを制御するGUIコンポーネントの集合に接続合成規則を適用している．合成規則は一対の分散コンポーネントを制御するGUIコンポーネントの集合に対して適用し，それを繰り返し行うことにより，複雑に配置された分散コンポーネントを制御可能なユーザインタフェースを生成できる．

## 3.2　ユーザインタフェース合成機構の機能要件

以下に本研究におけるユーザインタフェース合成機構の要件を詳述する．ユーザインタフェース合成機構を実現するには，合成規則の動的適用，実行コマンドの一貫性の保持，GUIコンポーネントの自動配置3つの要件を考慮する必要がある．

### 合成規則の動的適用

本論文で提案するユーザインタフェース合成機構では，特定のサービス指定によるトップダウンな分散コンポーネント間の関係定義モデルを採用する．トップダウンな関係定義モデルでは，与えられたサービスを構成する分散コンポーネントの集合間の関係をメタ情報を用いて動的に導き出す必要がある．ここでの分散コンポーネント間の合成ルールを求めるアルゴリズムにはエキスパートシステムで用いられる前向き推論アルゴリズムを用いる．ユーザインタフェースの合成は，動的に関係付けられる分散コンポーネント間の合成ルールに従って行われるため，GUIコンポーネント間の関係を静的に記述できない．そのため，ユーザインタフェースの合成では，動的にGUIコンポーネントの合成規則を適用する必要がある．また，適用する合成規則は，分散コンポーネントの関係から求められるため，自動的にGUIコンポーネントの集合に対して適用される．合成規則の適用に際してユーザの要求は，分散コンポーネント間の関係を定義する過程で反映される．

### 実行コマンドの一貫性の保持

ユーザインタフェースを合成する際，GUIコンポーネントは集約され，ユーザが操作するGUIコンポーネントの総数が減少する．その際，合成以前のGUIコンポーネントとその実行コマンドの対応を考慮する必要がある．通常，一つのGUIコンポーネントは一つの実行コマンドに対応するが，合成規則を適用することにより，一つのGUIコンポーネントが複数の実行コマンドを発行する．また，本研究では分散コンポーネントには形式的な型情報を求めるが，分散コンポーネントを制御するGUIコンポーネントは定義しない．分散コンポーネントの実装は個々に異なるため，実行に必要な入力値が異なるためである．従って，型情報が同一でも，GUIコンポーネントの集合が異なる場合，それらを合成した際の入力値の不整合を考慮する必要がある．また，実行コマンドを発行するGUIコンポーネントが一つの機能に対して複数存在する不整合を考慮する必要がある．たとえば，複数のエアコンを制御可能なユーザインタフェー

スを生成した場合，スライドバーを操作するとエアコンにコマンドが送信され，設定ボタンを押すとスライドバーの値が他のエアコンに送信される不具合を考慮する必要がある．

### GUI コンポーネントの自動配置

接続合成規則，集約合成規則の適用によって GUI コンポーネントを合成した際には，生成される GUI コンポーネントの配置方法を考慮する必要がある．GUI コンポーネントの合成は動的に実行されるため，静的に GUI コンポーネントの配置を記述できない．そのため，GUI コンポーネントの自動配置アルゴリズム，および自動配置可能な GUI コンポーネントのレイアウト記述方法を考慮する必要がある．

## 3.3 GUI コンポーネントの階層化

分散コンポーネントを制御する GUI コンポーネントは，MainComponent と SubComponent に分類できる．MainComponent は，実際に分散コンポーネントに対してコマンドを発行する GUI コンポーネントであり，SubComponent は MainComponent が分散コンポーネントに対して発行するコマンドの引数を入力するための GUI コンポーネントである．図 3.2 に MainCompoent と SubComponent の関係を示す．たとえば，簡単なエアコン制御の GUI コンポーネントの集合は，温度設定用 GUI コンポーネントと設定項目を実行する転送用 GUI コンポーネントから構成される．この場合，実際にコマンドを発行する GUI コンポーネントが転送用 GUI コンポーネントであるため，MainComponent にあたる．

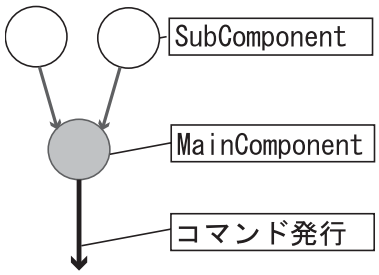



図 3.2: MainComponent と SubComponent の関係

## 3.4 GUI コンポーネントの合成規則

本研究では，ユーザインタフェースの合成規則として**接続合成規則と集約合成規則**の 2 つの合成規則を用いる．合成規則は，既存のサービス合成機構における分散コンポーネントの配置に基づいて適用される．既存のサービス合成機構では，分散コンポーネント間の関係を定義するために，Output Type，Input Type，Role Type のメタ情報を用いる．以下に集約合成規則，接続合成規則の概念を述べる．

### 3.4.1 集約合成規則

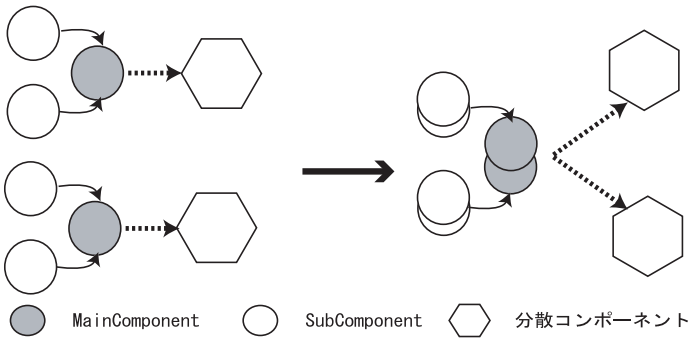



図 3.3: 集約合成規則の概念図

集約合成規則では，分散コンポーネント間の Role Type が一致する関係において，個々の分散コンポーネントを制御する GUI コンポーネントを合成する．Role Type が一致する分散コンポーネントの関係は，同様な役割を持つ分散コンポーネントであるため，集約的に実行可能な関係にある．図 3.3 に集約合成規則の概念図を示す．白色の丸が SubComponent を示し，黒色の丸が MainComponent を示す．集約合成規則の適用によって生成される GUI コンポーネントの構成が右側に示す丸の集合である．重なった白い丸が集約された SubComponent を示し，重なった黒い丸が集約された MainComponent を示す．GUI コンポーネントの集約可能性は，GUI コンポーネントの操作によってユーザが入力可能な値によって決定される．たとえば，ある GUI コンポーネントのセットがそれぞれスライドバーとボタンで構成される場合を考える．スライドバーは整数値を入力可能な GUI コンポーネントであり，ボタンはブール値を入力可能な GUI コンポーネントであるため，スライドバーとボタンはそれぞれ集約される．SubComponent 間の入力値の違いについては，3.5 節で述べる．

### 3.4.2 接続合成規則

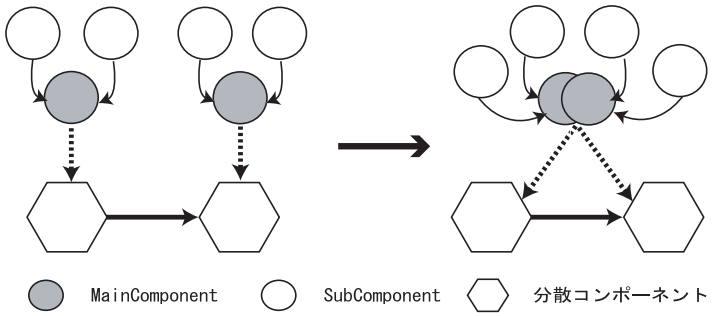



図 3.4: 接続合成規則の概念図

接続合成規則では，分散コンポーネント間の Output Type と Input Type が一致する関係において，個々の分散コンポーネントを制御する GUI コンポーネントを合成する．Output Type と Input Type が一致する分散コンポーネント間の関係は，データの入出力の接続による分散コンポーネントの協調動作である．図 3.4 に接続合成規則の概念図を示す．六角形が分散コンポーネントを示し，その間の矢印がデータの流れを表す．灰色の丸が MainComponent を示し，白色の丸が SubComponent を示す．接続合成規則によって生成される GUI コンポーネントの構成が図の右側に示す丸の集合である．重なっている白い丸は集約合成規則の適用によって集約された SubComponent を示す．接続合成規則では，それぞれの分散コンポーネントを実行する MainComponent から，新規に両方の分散コンポーネントを制御できる GUI コンポーネントを生成する．接続される MainComponent の順番は，分散コンポーネント間の関係定義の層で決定される．集約された SubComponent の入力値はシステムの内部でそれぞれの実行コマンドに必要な値に分類され，MainComponent によって個々の分散コンポーネントに対して発行される．

## 3.5 GUI コンポーネントと実行コマンド

本節では，GUI コンポーネントと実行コマンドの関係について述べる．ユーザインタフェースを合成する際，GUI コンポーネントは集約され，ユーザが操作する GUI コンポーネントの総数が減少する．その際，合成以前の GUI コンポーネントとその実行コマンドの対応を考慮する必要がある．以下に，ユーザインタフェースの合成における GUI コンポーネントと実行コマンドに関して考慮すべき要件を述べる．

### 実行コマンドの分岐

複数の分散コンポーネントを制御可能なユーザインタフェースを生成する際，実行コマンドの個々の分散コンポーネントへの対応を考慮する必要がある．たとえば，分散コンポーネント A は引数にブール値を取るが，分散コンポーネント B はユーザが設

定する整数値を引数に取る場合がある．分散コンポーネント A を制御するユーザインタフェースは単一のボタンで構成され，分散コンポーネント B を制御するユーザインタフェースはボタンとスライドバーから構成される場合，スライドバーの値は分散コンポーネント A の実行コマンドには反映されない．この場合，ユーザインタフェース合成機構は内部的に実行コマンドを分離する必要がある．

### コマンド引数と GUI コンポーネントの対応

同様な役割を持つ分散コンポーネントを制御する GUI コンポーネントであっても，実行コマンドの引数が異なる場合がある．たとえば，音量を設定する分散コンポーネントは，整数値と小数値を取る場合がある．この場合，整数値と小数値に対応する GUI コンポーネントがスライドバーであっても，その操作が意味する実際の値が異なる．そのため，実際に分散コンポーネントへ送信されるコマンドの引数は，スライドバーの割合と，分散コンポーネントが入力値として受け取れる値の最小値と最大値から算出する必要がある．なお，CUIC において，ユーザインタフェースの合成は分散コンポーネントのメタ情報を反映して行われるため，整数値と文字列などの入力値の矛盾は想定しない．

### ユーザインタフェースの一貫性の保持

ユーザインタフェースを合成する際，GUI コンポーネントの動作結果を考慮する必要がある．GUI コンポーネントの動作結果とは，GUI コンポーネントを操作した際に分散コンポーネントに対して発行される実行コマンドの有無を指す．たとえば，分散コンポーネント A は GUI コンポーネントの操作がそのまま実行コマンドになるが，分散コンポーネント B は実行コマンドの値を入力する GUI コンポーネントと実行コマンドを送信する GUI コンポーネントが別々に定義されている場合，これらの GUI コンポーネントをそのまま合成するとユーザインタフェースの一貫性を保てない．複数の分散コンポーネントを集約的に制御するユーザインタフェースを生成しても，個々の GUI コンポーネントが個別の分散コンポーネントに対応していてはユーザインタフェースの合成の意味がない．そのため，ユーザインタフェースを合成する際には，実行コマンドを発行する GUI コンポーネントを一意に決定する必要がある．

## 3.6　本章のまとめ

本章では，はじめにユーザインタフェース合成機構の概要を述べ，サービス合成機構とユーザインタフェース合成機構の関係を定義した．その後，ユーザインタフェース合成機構の要件をまとめ，本研究が提案する接続合成規則と集約合成規則，および GUI コンポーネントと実行コマンドの関係について述べた．

# 第 4 章

# 言語仕様

本章では，本研究で用いるユーザインタフェース記述言語 **CUIL (Composable User Interface Language)** の言語仕様，およびメタ情報の記述言語 **STDL (Standard Type Definition Language)** の言語仕様について述べる．前章で述べたユーザインタフェース合成機構を実現するには，サービスが提供する単位でのユーザインタフェース記述言語と，分散コンポーネントのメタ情報の記述言語が必要である．本章では，はじめに既存のユーザインタフェース記述言語を分類し，CUIL の設計指針について述べる．その後，詳細な言語仕様について述べる．

## 4.1 全体の構成

CUIL はサービスが提供する機能をベースにしたユーザインタフェース記述言語である．ホームネットワークにおいて，サービスが提供する不可分な機能が分散コンポーネントにあたる．そのため，ユーザインタフェースは機能単位で記述する必要がある．機能単位で GUI コンポーネントを記述することにより，サービス合成の概念をユーザインタフェース合成に反映できる．また，これらの機能間の関係を定義するためには，機能自身を説明する情報が必要である．本論文では，サービスが提供する機能のメタ情報記述言語である **STDL (Standard Type Definition Language** を提案する．

### 4.1.1 既存のユーザインタフェース記述言語の分類

既存のユーザインタフェース記述言語としては，Java や C++等のプログラミング言語が挙げられる．しかしプログラミング言語は，特定の GUI ツールキットを用いて GUI コンポーネントを静的に記述するため，多様な制御デバイス上で動作できない．たとえば，Visual C++で記述されたユーザインタフェースの記述は PDA 上では動作できない．反対に，サービスが多様な制御デバイスに対応するため，複数のユーザインタフェース記述を持つことは，情報家電機器の記憶容量が限られることや，開発コストの面で現実的でない．そのため，特定の GUI ツールキットに依存しない形式でユーザインタフェースを記述する言語が必要である．そのため，現在多くのユーザインタフェース記述言語が提案されている．これらの言語の多くは XML を用いており，ユーザインタフェースの生成は各制御デバイス上に配置されたレンダリングソフトウェアを用いて行う．制御デバイス上にレンダリングソフトウェアを配置するコストは，各サービスが個々の制御端末のプラットフォームに対してユーザインタフェースを記述するコストに比べて低い．

このような XML を用いたドキュメントベースのユーザインタフェース記述言語として，サービスが提供する機能単位でユーザインタフェースを記述する ISL(Interface Specification Language)[8]，SDL(Service Description Language)[15]，ASL(Appliance Specification Language)[7] が挙げられる．これらの言語ではサービスが提供する機能単位で GUI コンポーネントを記述し，個々の機能を階層化することによって GUI コンポーネント間の関係を記述する．

### 4.1.2 CUIL の特徴と独自言語を提案する理由

CUIL は前節で述べた他の機能単位でユーザインタフェースを記述する言語と同様に，機能単位でユーザインタフェースを記述する言語である．本研究では，分散コンポーネント間の関係に基づいて GUI コンポーネントを合成するため，サービスが提供する機能単位で GUI コンポーネントを合成する必要がある．既存の言語はユーザインタフェースのプラットフォーム非依存性を主眼とするのに対し，CUIL はユーザインタフェースを合成することを主眼とする．CUIL では，既存の言語が実現するプラット

フォーム非依存性を保持したまま，合成可能なユーザインタフェース記述言語である．そのため，CUIL は既存の機能単位でユーザインタフェースを記述する言語に対して以下の点で異なる．

- **機能単位でのメタ情報の記述**
- **GUI コンポーネントの階層化**
- **GUI コンポーネントの動的再配置を実現するレイアウト記述**

.

機能単位でのメタ情報の記述とは，サービスが提供する機能の Role Type，I/O Type を記述するものである．GUI コンポーネントの階層化とは実際にサービスを実行するコマンドを発行する GUI コンポーネントと，機能の実行に必要な情報の入力をユーザに促す GUI コンポーネントの分類である．これらの GUI コンポーネントは機能単位で記述されるため，サービス合成の概念に基づいてユーザインタフェースの合成を実現できる．動的 GUI コンポーネント配置を実現するレイアウト記述とは，以上のユーザインタフェースの合成を行った際に，動的に GUI コンポーネントを配置するためのレイアウト記述である．CUIL では，静的にユーザインタフェースの座標を指定するのではなく，グリッドレイアウトとコンポーネント間の重み付けによって GUI コンポーネントを配置する．

既存のユーザインタフェース記述言語は，GUI コンポーネントの合成を主眼としないため，サービスが提供する単位で GUI コンポーネントを記述していても，実際にその機能自身の情報は含まない．そのため，分散コンポーネント間の関係を定義する情報が不足する問題点がある．また，GUI コンポーネント間の階層化がなされていない．レイアウトの記述方法も，静的に個々の GUI コンポーネントを配置するために，自動的に GUI コンポーネントを配置できない問題点がある．そのため，CUICs では以上の要件を満たす独自のユーザインタフェース記述言語である CUIL(Composable User Interface Language) を提案する．

## 4.2 CUIL: Composable User Interface Language

本節では，CUIL の言語仕様について詳細に述べる．CUIL は機能単位でのメタ情報の記述，GUI コンポーネントの階層化，動的 GUI コンポーネントの配置の点で既存のユーザインタフェース記述言語と異なる．CUIL は GUI を対象とした言語であるが，ボタンやスライドバーなどの記述は行わず，ユーザが入力する情報によって抽象的にユーザインタフェースを記述する．ユーザが入力する情報としては，boolean，int，string 等が挙げられる．CUICs では，これらの情報と実際の GUI コンポーネントのマッピングを個々の制御用端末上のレンダリングソフトウェア毎に行う．これにより，特定の GUI ツールキットに依存せず，個々の制御デバイスで利用可能な GUI コンポーネントによって CUIL からユーザインタフェースを生成できる．

### 4.2.1 CUIL 概要

CUIL は⟨*service*⟩タグ,⟨*function*⟩タグ,⟨*layout*⟩タグから構成される．以下に，CUIL の概要を示す.

```
<?xml version="1.0" encoding="UTF-8"?>
<cuil>
    <service></service>
    <function></function>
              ⋮
    <function></function>
    <layout></layout>
</cuil>
```

⟨*service*⟩タグはサービスに固有な情報を記述する領域であり，⟨*function*⟩タグはサービスが提供する機能を記述する領域である．⟨*function*⟩タグは一つのユーザインタフェース記述に複数含まれ，GUI コンポーネントおよび実行コマンドの記述は⟨*function*⟩タグの内部に記述する．⟨*function*⟩タグについては，4.2.3 項で詳細に述べる．また，⟨*layout*⟩タグは個々の機能を実行するために必要なコンポーネントを配置するための情報を記述する領域である．⟨*layout*⟩については，4.2.4 項で詳細に述べる．

### 4.2.2 サービスの記述

⟨*service*⟩タグは，サービスに固有な情報を記述する領域である．サービスに固有な情報としては，1) サービスの固有名称，2) サービスに対する説明，3) サービスの型，4) 開発ベンダ名，5) シリアルナンバー，6) バージョン情報が挙げられる．CUICs が対象とするホームネットワークにおいて，サービスは複数の分散コンポーネントから構成されていても，同一の計算機上に存在することが多い．そのため，ユーザの制御対象はそのまま物理的なデバイスである場合が多い．CUICs では，カード型インタフェースマネージャを用いてサービスを選択することにより，合成可能な分散コンポーネントの組を提示するモデルを採用している．したがって，ユーザは複数のサービスを選択することになる．そのため，サービス固有な情報をユーザインタフェース記述に含めることにより，ユーザのサービス選択時の利便性を向上できる．以下に⟨*service*⟩タグの記述例を示す.

```
<service>
    <name>SampleService</name>
    <desc>This is sample service.</desc>
    <type>GenericService</type>
    <vendor>FooCompany</vendor>
    <serialNumber>xxxxxx</serialNumber>
    <version>xxxxxx</version>
</service>
```

### 4.2.3 機能の記述

CUIL において，⟨*function*⟩ タグがサービスを構成する分散コンポーネントにあたる．⟨*function*⟩ タグは，⟨*meta*⟩ を含む．⟨*meta*⟩ タグでは，分散コンポーネントの Role Type，Input Type，Input Type 等の機能のメタ情報を定義する．⟨*subComponent*⟩ タグ，⟨*mainComponent*⟩ タグには GUI コンポーネントを記述する．

```
<function id="arbitrary ID">
    <meta type="Amplipher">
        <role>VolumeControl</role>
        <inputType>null</inputType>
        <outputType>null</outputType>
    </meta>
    <subComponent></subComponent>
                    ⋮
    <subComponent></subComponent>
    <mainComponent></mainComponent>
    <command></command>
</function>
```

#### GUI コンポーネントの記述

GUI コンポーネントの階層化について述べる．3.3 で挙げたエアコンの例のように，一つの機能に複数の GUI コンポーネントがある場合がある．GUI コンポーネントは ⟨*mainComponent*⟩ と ⟨*subComponent*⟩ に分かれる．⟨*mainComponent*⟩ は実際にコマンドを発行するコンポーネントであり，⟨*subComponent*⟩ はコマンドを発行するために必要な情報を受け付けるコンポーネントである．一つの機能には一つの ⟨*mainComponent*⟩ が必要であるが，⟨*subComponent*⟩ はいくつあっても良い．以下に，GUI コンポーネントの記述例を示す．

```
<mainComponent id="VolumeExecute">
    <type>boolean</type>
    <label>VolumeExecute</label>
</mainComponent>
<subComponent id="VolumeChange">]
    <type max="63" min="0" default="20">int</type>
    <label>VolumeChange</label>
</subComponent>
```

⟨*subComponent*⟩ タグ，⟨*mainComponent*⟩ タグ内の ⟨*type*⟩ は，個々の GUI コンポーネントによってユーザが入力可能な値の例を示す．CUIC では，⟨*type*⟩ 内で記述された値に従って実際の GUI コンポーネントを生成する．表 4.1 に CUIL で利用可能な入力値の一覧を示す．入力値によっては，最小値と最大値が必要な場合がある．最大値と最小値を取る入力値としては int，float，double が挙げられる．また，あらかじめ複数の値を定義しておき，それに当てはまる値を選択させるような入力値も考えられる．これには，enumerate が挙げられる．

表 4.1: CUIL タグ一覧

| 入力値 | 説明 |
|---|---|
| boolean | true or false を入力可能な GUI コンポーネント e.g. JButton |
| int | 整数を入力可能な GUI コンポーネント e.g. JSlidebar |
| float | 小数値を入力可能な GUI コンポーネント e.g. Slider |
| double | 倍精度の小数値を入力可能な GUI コンポーネント e.g. JSlider |
| string | 文字列を入力可能な GUI コンポーネント e.g. JTextfield |
| enumerated | 選択肢から選ぶことができる GUI コンポーネント e.g. RadioButton |

#### 実行コマンドの記述

CUICs では，分散コンポーネントへのコマンドの発行には遠隔メソッド呼び出しを用いる．遠隔メソッド呼び出しに必要な情報は，特定のミドルウェア上でのサービスの識別子，およびメソッド名，引数，スタブオブジェクトの場所である．サービスの識別子は CUICs が内部的に管理するため，⟨*command*⟩ タグが含む情報はメソッド名と引数の情報である．⟨*param*⟩ タグには，⟨*subComponent*⟩ の id を記述する．

```
<command>
    <method>volume</method>
    <parameter>VolumeChange</parameter>
</command>
```

### 4.2.4 レイアウトの記述

CUIL では，ユーザインタフェース合成を考慮して，GUI コンポーネントの再配置が簡易なグリッドを用いたレイアウト記述を採用する．GUI コンポーネントは，カードに張った格子状の枠の中に配置される．⟨*layout*⟩ タグでは，まず全体のグリッドの数を定め，個々の GUI コンポーネントの配置は ⟨*cell*⟩ タグの中で行う．⟨*cell*⟩ タグの値としては，グリッド内での行列，幅，高さである．このように GUI コンポーネントを抽象的に記述することにより，画面表示領域が異なる制御用端末上でも同様なユーザインタフェースを実現出来る．

```
<layout row="8" column="6">
    <cell row="1" column="4" width="2" height="1">on</cell>
                              ⋮
    <cell row="7" column="1" width="6" height="1">VolumeChange</cell>
</layout>
```

## 4.3 STDL 概要

コンポーネントの Input Type，Output Type，Role Type はコンポーネントを説明するメタ情報である．コンポーネント間の関係はメタ情報を使って定義される．しかし，メタ情報の定義がベンダによって異なると，コンポーネント間のインタオペラビリティが損なわれる．そのため，異なるサービスを構成するコンポーネントであっても，同様な機能を提供するコンポーネントは標準的な形式にしたがって定義される必要がある．これらのメタ情報に関する標準は確立されていないが，EchoNet[1] ではサービスが提供すべき機能を策定しており，Microsoft 社が提唱する UPnP(Universal Plug and Play)[23] でも，UPnP A/V や UPnP Remote 等のサービスを制御するためのアプリケーションプロトコルの策定をはじめている．アプリケーションプロトコルは，サービスが提供すべき機能を定義するという意味で本論文におけるメタ情報と同義である．ホームネットワークにおけるサービスを考慮すると，サービスを制御する際はサービスが最低限提供すべき機能の標準化が行われれば良い．それ以外の個々のサービスに固有な分散コンポーネントのメタ情報は，個々のベンダ内で統一されていれば，特定のベンダの製品内でのインタオペラビリティは確保される．本論文では，既存の標準を考慮した上で，拡張可能なメタ情報定義フォーマット STDL(Standard Type Definition Language) を提案する．

STDL は CUIL で用いるメタ情報である Input Type，Output Type, Role Type を一般的なサービスに分類して記述する．⟨*genericService*⟩ タグが一般的なサービスを表す．⟨*genericService*⟩ 内の ⟨*function*⟩ タグが一般的にこのサービスが提供すべき機能を定義する．⟨*function*⟩ タグ内に記述する内容は，CUIL における ⟨*meta*⟩ タグと同様である．

```
<?xml version="1.0" encoding="UTF-8"?>
<stdl>
    <genericService name="DigitalVideoCamera">
        <function>
            <role>Capture</role>
            <inputType>NULL</inputType>
            <outputType>MPG2 Format</outputType>
        </function>
        <function>
            <role>SnapShot</role>
            <inputType>NULL</inputType>
            <outputType>JPG Format</outputType>
        </function>
                              ⋮
    </genericService>
</stdl>
```

## 4.4　本章のまとめ

本章では，サービスが提供する単位でユーザインタフェースを記述する CUIL の仕様について述べた．CUIL は既存の言語に比べ，機能単位でのメタ情報の記述性，GUI コンポーネントの階層化，GUI コンポーネントの動的再配置を実現するレイアウト記述の点で異なる．CUIL を用いることにより，ユーザインタフェースを動的に合成可能になる．

# 第 5 章

# カードのメタファを用いたユーザインタフェース合成機構 CUICs の設計

本章では，ユーザインタフェースの動的合成機構 CUICs の設計について述べる．CUICs は，ユーザインタフェース記述言語 CUIL やメタ情報記述言語 STDL からカード型インタフェースを生成するユーザインタフェース生成部，ユーザインタフェースをカードとして表示してサービス間の関係をユーザが動的に変更可能にするための視覚的ツールであるカード型インタフェースマネージャ部，カードの重なりを判定してユーザインタフェースを合成する合成制御部，実際にサービスに対して送信されるコマンドを管理するコマンド制御部，利用可能なサービスを下位のサービス合成機構非依存に管理するサービス管理部の 6 つのソフトウェアモジュールから構成される．はじめにシステム全体の設計および設計指針を述べ，個々のモジュールについて詳細な説明を行う．

## 5.1 設計指針

本論文では，カードのメタファを用いたユーザインタフェースの動的合成機構 **CUICs (Card-oriented User Interface Composition System)** を提案する．CUICs の目的は，ホームネットワークにおいて，(1) ユーザがより簡易に多様なサービスを構成する分散コンポーネント間の関係を定義し，(2) 協調動作を行う分散コンポーネントを制御可能なユーザインタフェースをユーザに提示することである．以上の目的を達成するため，CUICs では動的変更性，簡易性，隠蔽性，非依存性，永続性を設計指針とする．

### 動的性

ホームネットワークは多様なサービスと多様な制御用端末から構成されるため，静的な分散コンポーネント間の関係定義や静的な GUI コンポーネント間の合成規則の定義は現実的でない．そのため，動的に分散コンポーネント間の関係や GUI コンポーネント間の関係をシステムが自動的に定義可能な機構を組み込む必要がある．

### 簡易性

第 2 章で述べたトップダウンな分散コンポーネント間の関係定義モデルであるルール生成モデルでは，ユーザが特定のサービスを指定する必要がある．その際，ユーザが簡易にサービスを指定可能なフロントエンドを提供する必要がある．

### 隠蔽性

既存のサービス合成のアプローチは，エンドユーザを対象としたものというより，ソフトウェアの開発効率や再利用性を向上を主眼としている．CUICs が対象とするホームネットワークでは，専門知識を持たないユーザが直接サービスを指定し，協調動作させる分散コンポーネント間の関係を定義する．そのため，CUICs ではトップダウンな分散コンポーネント間の関係定義モデルを採用し，ユーザが任意に指定するサービスから動的にサービスを構成する分散コンポーネント間の関係を求める．そのため，分散コンポーネント間の関係定義のプロセスはユーザに対して隠蔽すべきである．

### 非依存性

CUICs 自身は下位のサービス合成機構に対して非依存である．サービス合成機構は指定された分散コンポーネント間の通信パスを動的に生成するが，その方法は機構によって異なる．そのため，CUIC を特定のサービス合成機構に依存して設計すると，多様なサービス合成機構の上で動作できない．CUIC では，個々のサービス合成機構に対するアダプタを用意し，アダプタを介して下位機構と通信する．

#### 永続性

ユーザがサービスの協調動作する際にユーザインタフェースを合成することはかえってユーザビリティを低下させる可能性がある．そのため，一度合成したカードは保存され，再利用可能である必要がある．

## 5.2 システムの概要

本節では，CUICs の概要をハードウェア構成，ソフトウェア構成の面から述べる．ソフトウェア構成では，CUICs をモジュールで分割し，個々のモジュールの関連について述べる．最後に，CUICs の全体の動作概要を述べる．

### 5.2.1 ハードウェア構成

CUICs のハードウェア構成は，ホームネットワークに接続された多様なサービス，および多様な制御用端末である．サービスとしては，ネットワーク接続性を持った白物家電機器，DVD プレーヤやスピーカ等の AV 機器が挙げられる．制御用端末としては，PC やワークステーション，PDA，携帯電話が挙げられる．CUICs は任意の制御用端末上で動作する．

### 5.2.2 ソフトウェア構成

CUICs は，カード型インタフェースマネージャ部 (以下 CIM)，ユーザインタフェース生成部，ユーザインタフェース合成部，コマンド実行部，サービスマネージャ部，ミドルウェアアダプタ部の 6 つのモジュールから構成される．図 5.1 に CUICs のソフトウェア構成の概念図を示す.

#### カード型インタフェースマネージャ部

CIM は，CUICs のフロントエンドである．ユーザは CIM を上に表示されるカードを用いてサービス間の関係を定義する．個々のサービスを制御するユーザインタフェースはカードとして表示され，ユーザはカードを重ねることによってユーザインタフェースの合成を行う．CIM の機能要件として，ユーザインタフェースの表示，カードの重なりの判定，GUI コンポーネントのイベントハンドリング，利用可能なサービスの一覧表示が挙げられる．

#### ユーザインタフェース生成部

ユーザインタフェース生成部は，サービスからダウンロードした CUIL を CUICs 内部で使用するデータ構造にパースし，カードを生成するモジュールである．パースされた CUIL は，個々のプラットフォームに依存した GUI ツールキットと GUI コンポー

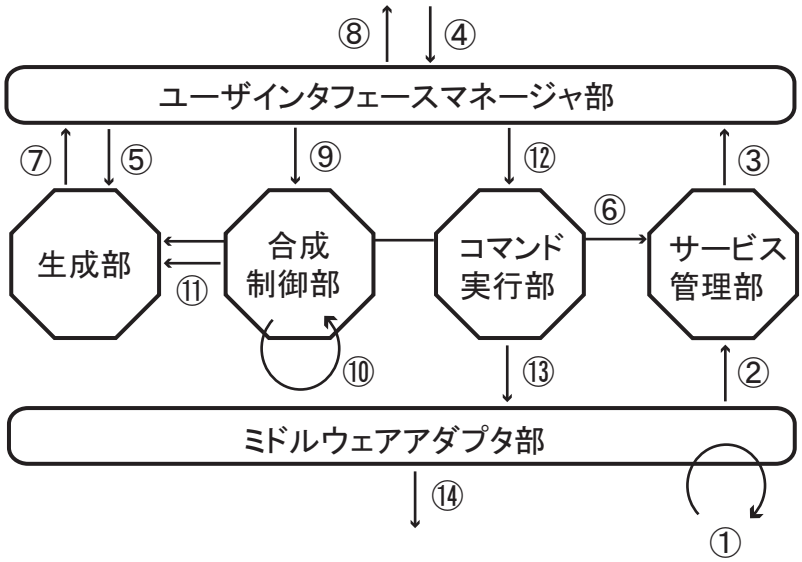



図 5.1: ソフトウェア構成概念図

ネントのマッピングに従ってカードとして CIM 上に表示される．ユーザインタフェース生成部の機能要件として，CUIL の構文解析，GUI コンポーネントと利用可能な GUI ツールキットとのマッピング，ユーザインタフェースの生成が挙げられる．

### 合成制御部

合成制御部は，CUIC システムの基幹モジュールである．合成制御部では，ルール生成モデルに基づく分散コンポーネント間の関係定義，および GUI コンポーネントの合成を行う．合成したユーザインタフェースは内部で保存され，再利用可能である．合成制御部の機能要件としては，分散コンポーネント間の関係定義，GUI コンポーネントの合成，合成済みユーザインタフェースの永続化が挙げられる．

### コマンド実行部

コマンド実行部は，ユーザインタフェースが発行する実行コマンドを制御するモジュールである．3.5 節で述べたように，合成された GUI コンポーネントは内部的に関連付けられている．コマンド実行部の機能要件としては，実行コマンドの分岐，コマンド引数と GUI コンポーネントの対応が挙げられる．

### サービス管理部

サービス管理部は，ミドルウェアアダプタ部を介してサービスの管理を行うモジュールである．サービスの状態の変化をもとに，CIM 上に表示されるサービスの情報を変更する．また，新しいサービスが見つかると，サービス管理部は CUIL をサービスから

動的にダウンロードする．これらのサービスの管理は下位のサービス合成機構非依存に行われる．サービス管理部の機能要件としては，サービスの追加と削除，CUIL の動的ダウンロードと保存，CIM 上のサービスの情報の変更が挙げられる．

#### ミドルウェアアダプタ部

CIM がユーザに対するフロントエンドであるのに対して，ミドルウェアアダプタ部はサービス合成機構に対するインタフェースである．CUICs をサービス合成機構に対して非依存にするため，ミドルウェアアダプタ部は Jini, CORBA, VNA 等のサービス合成機構に対するアダプタを持つ．アダプタの機能要件として，サービス合成機構が提供するディレクトリサービスとの通信，分散コンポーネントとの通信が挙げられる．

### 5.2.3 動作手順

CUIC を構成するモジュール群の動作手順の概要を述べる．CUICs の動作は，(1) 利用可能なサービスの更新と表示，(2) ユーザインタフェースの合成と表示，(3) コマンドの実行に分類できる．(1) は継続的に動作するフローであり，(2)，(3)，(4) はユーザからの入力を受けて始まる手順である．図 5.1 でのモジュール間の矢印に振られた数字が動作手順にあたる．以下に，個々の動作の概要を説明する．

#### (1) 利用可能なサービスの表示，更新

1. ミドルウェアアダプタ部は，サービス合成機構が提供するディレクトリサービスに対して現在利用可能なサービスの一覧を更新する．この動作は一定間隔で行う．

2. 利用可能なサービスの一覧に変更があった場合，ミドルウェアアダプタ部はサービス管理部に対して変更があったサービスを通知し，サービス管理部はそれを処理する．
3. サービス管理部はユーザインタフェースマネージャに対してサービス一覧に変更があったことを通知し，CIM 上のサービス一覧を変更する．

#### (2) ユーザインタフェースの生成

4. ユーザが CIM 上でサービスを選択する．
5. CIM はユーザインタフェース生成部に対してユーザインタフェース生成要求を出す．
6. ユーザインタフェース生成部は，サービス管理部に対して CUIL を要求する．
7. 受け取った CUIL からユーザインタフェースを生成する．
8. CIM に生成したユーザインタフェースを渡し，処理を終了する．

**(3) ユーザインタフェースの合成**

4. ユーザが CIM 上でカードを重ねる.
9. CIM から合成制御部に対し，重ねられたサービスの ID に優先度をつけて合成制御部に渡す.
10. 合成制御部は同様な合成の結果がないかをチェックする．該当する合成結果が見つかった場合，11 の処理を行う．見つからなかった場合は，生成部よりパースされた CUIL を取得し，合成規則を適用する.
11. 合成規則の結果をユーザインタフェース生成部に渡す．生成部は (2) の 7，8 の手順を行い，処理が終了する.

**(4) コマンドの実行**

4. ユーザが CIM 上に表示されたユーザインタフェースを操作する.
12. CIM はコマンド実行部に対して GUI コンポーネントに対応するサービスの ID, MainComponent に記述されたメソッド名，関連付けられている SubComponent の引数の値を渡す.
13. コマンド実行部はサービスの ID とメソッド名をミドルウェアアダプタ部に対してコマンドを発行する.
14. ミドルウェアアダプタはサービスに対してコマンドを発行する.

## 5.3 カード型インタフェースマネージャ部

カード型インタフェースマネージャ部は，以下に示す 4 つのサブモジュール群から構成される．サービス表示モジュールはサービス管理部からの通知を受けて利用可能なサービス一覧を表示する GUI コンポーネント群である．ユーザインタフェース表示モジュールは，ユーザインタフェース生成部から受け取る GUI コンポーネントのセットを表示する GUI コンポーネント群である．ユーザインタフェースはカードとして表示され，これらの重なりを判定するモジュールがカード制御モジュールである．イベント制御モジュールは，GUI コンポーネントのイベントハンドラ群である．図 5.2 にカードを用いたユーザインタフェース合成の概念図を示す.

- **サービス表示モジュール**
- **ユーザインタフェース表示モジュール**
- **カード制御モジュール**
- **イベント制御モジュール**

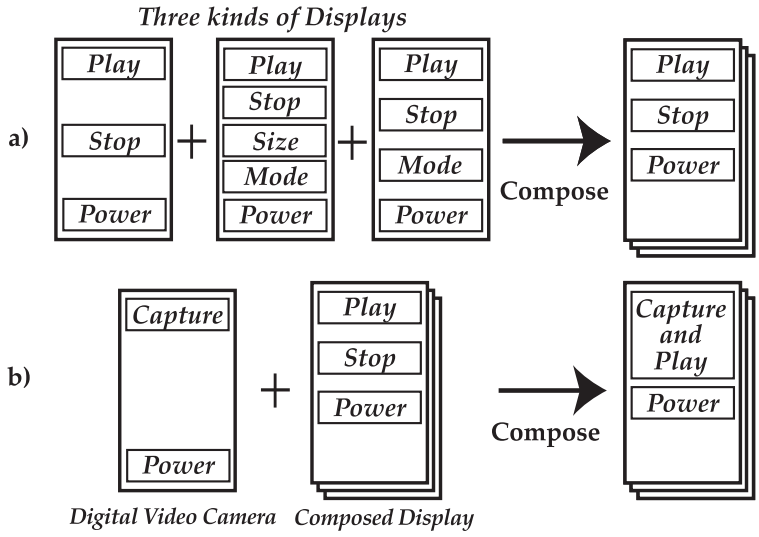



図 5.2: カード型インタフェースマネージャの概念図

CUICs では，分散コンポーネント間の関係定義モデルとして，第 2 章で述べたルール生成モデルを用いる．ルール生成モデルは，複数の分散コンポーネントの実体が与えられたとき，合成可能な分散コンポーネント間の解集合を求め，自動的に個々の分散コンポーネント間の関係を定義するモデルである．しかし，利用可能な全てのサービスに対してルール生成モデルを用いて協調動作可能な分散コンポーネントの解集合を求めると，提案される分散コンポーネントの組み合わせが増大する．そのため，ルールベースモデルに与える分散コンポーネントの集合をあらかじめ限定する必要がある．2.3.2 項で述べたように，CUICs ではユーザにルール生成モデルに与える分散コンポーネントの集合を簡易に指定可能なカード型インタフェースマネージャを提案する．これをフロントエンドとして用いることにより，より限定的で意味のある分散コンポーネントの関係をユーザに提示できる．

CIM において，ユーザは**カードを重ねる**という簡易な操作によって分散コンポーネント間の関係を定義し，ユーザインタフェースを合成できる．CIM を用いたユーザインタフェース合成の概念図を図 5.2 に示す．図 5.2(a) は集約合成規則の適用例を示しており，3 つのディスプレイを同時に制御可能なユーザインタフェースを生成している．また図 5.2(b) は接続合成規則と集約合成規則の適用例を示しており，(a) で生成されたカードをデジタルビデオカメラのカードに重ねることにより，キャプチャ機能の画像出力とプレイ機能の画像入力機能の接続による協調動作を制御可能なユーザインタフェースを生成している．ここでは，ユーザはカードを重ねる操作をインクリメンタルに行うことによって，複雑な協調動作を制御可能なユーザインタフェースを生成できる．

## 5.4 合成制御部

合成制御部は，以下に示す３つのモジュール群から構成される．分散コンポーネント関係定義モジュールは，カード型インタフェースマネージャからの要求に従って，ルールベースモデルのアルゴリズムを用いて分散コンポーネント間の関係を定義する．また，GUI コンポーネント合成モジュールは，分散コンポーネント関係定義モジュールで定義される分散コンポーネント間の関係に従って GUI コンポーネントを合成する．永続化モジュールは，以上のモジュールによって合成されたユーザインタフェースを保存しておき，合成済みのカードを再利用するためのモジュールである．

### 5.4.1 分散コンポーネント関係定義モジュール

CUICs では，複数の分散コンポーネントが与えられたとき，それらの合成ルールを第 2 章で述べたルール生成モデルを用いて求める．複数の分散コンポーネントとは，ユーザが指定するサービスが提供する機能の集合である．ルール生成モデルでは，エキスパートシステムや知識ベースシステムで用いられる前向き推論アルゴリズムを用いてメタ情報のマッチングを行う．前向き推論アルゴリズムとは，if-then ルールを用いて条件の真偽を判定するための推論ロジックである．if 以降を前段節といい，then 以降を後段節と呼ぶ．前段節が真である場合，後段節を CUICs の合成ルールに追加することにより，分散コンポーネント間の合成を行う．たとえば，ある分散コンポーネントが GIF フォーマットの画像を出力し，ある分散コンポーネントが GIF フォーマットの画像を入力として受け取る場合，接続合成規則は真になる．

CONNECT: IF (OUTPUT TYPE = GIF FORMAT)
AND (INPUT TYPE = GIF FORMAT)
THEN (COMPOSITION RULE = CONNECT)

しかし，この分散コンポーネント間の関係として，集約合成規則が真になる場合，この分散コンポーネント間には二つの合成規則が同時に成り立つ．たとえば，画像を変換する分散コンポーネントを合成する場合などが挙げられる．

AGGREGATE: IF (ROLE TYPE = CONVERSION)
AND (ROLE TYPE = CONVERSION)
THEN (COMPOSITION RULE = AGGREGATE)

このような前段節が真であるルールが複数存在する場合，真であるルールの集合を競合セットと呼ぶ．第 3 章で述べたように，CUIC では集約合成規則，接続合成規則の 2 種類の合成規則を用いる．そのため，一対の分散コンポーネント間には最大で 2 つの合成規則が成り立つ可能性がある．CUIC では，このような競合セットの中から適用する規則を選ぶために次の方針を用いる．

- **合成規則に優先順位を設定する**
- **規則の適用最大値を設けて可能な限り合成規則を適用する**
- **適用されなかった合成規則の優先度を上げる**

合成規則の優先順位は，動的制約と静的制約によって求められる．動的制約はユーザが実行時に決定するものであり，最もユーザの意思を反映しやすいことから優先度を高く設定する．反対に，静的制約はあらかじめ設定されているものであるため，ユーザの意思の介入が少ないことから優先度を低く設定する．

1. **順列合成規則**
2. **集約合成規則，接続合成規則**

集約合成規則，接続合成規則は共に静的制約であるため，優先順位が同列である．そのため，これらの合成規則を取捨選択できない．この場合，CUIC はできる限り全ての合成規則を適用する．できる限りとは，全ての合成規則を適用することによって生成される GUI コンポーネントの数がレイアウトの範囲を超えない範囲で合成規則を適用させることを指す．ここで適用されなかった合成規則は，一時的に CUIC の記憶領域に保存され，次回の合成時には優先順位が高く設定される．これにより，複数回の合成においてユーザに全ての可能性を提示できる．

### 5.4.2 GUI コンポーネント合成モジュール

5.4.1 項で述べた前向き推論アルゴリズムは，機能のマッチングだけでなく，機能を制御するために必要な GUI コンポーネント間の関係を求めるためにも利用する．機能間の合成規則が決定し，分散コンポーネントの集合が与えられたとき，次の段階として GUI コンポーネントの合成規則について考慮する必要がある．第 3 章で述べたように，CUICs では分散コンポーネントを制御する GUI コンポーネントを静的に定義しない．分散コンポーネントの入出力型および役割の型は外部 STDL にて定義するが，それを実行するための GUI コンポーネントの記述はそれぞれ異なることを許す．そのため，本節では集約，接続の合成規則に対する GUI コンポーネントの合成アルゴリズムについて述べる．はじめに mainComponent のみの合成アルゴリズムを説明し，次に subComponent を含む GUI 間の合成アルゴリズムを説明する．

集約合成における GUI コンポーネントの合成アルゴリズムについて述べる．たとえば，Amplifier A と Amplifier B があるとする．これらのアンプが持つ音量制御を行う GUI コンポーネントは以下の二通りの記述が可能である．Amplifier A はボリュームを一度設定した後で実行ボタンを押す GUI コンポーネント構成であり，Amplifier B はボリュームを設定する毎にコマンドを発行する GUI コンポーネント構成である．これらの GUI を CUIL で記述すると以下のようになる．なお，サービスは CIM によって合成されるため，サービスを順序付けられる．

AMPLIFIER A:

```
<function id="Volume">
    <meta type="Amplifier">
        <role>VolumeControl</role>
        <inputType>null</inputType>
        <outputType>null</outputType>
    </meta>
    <mainComponent id="VolumeExecute">
        <type>boolean</type>
        <label>VolumeExecute</label>
    </mainComponent>
    <subComponent id="VolumeChange">
        <type>int</type>
        <max>63</max>
        <min>0</min>
        <default>20</default>
        <label>VolumeChange</label>
    </subComponent>
    <command>
        <method>volumeChange</method>
        <parameter>VolumeChange</parameter>
    </command>
</function>
```

AMPLIPHER B:

```
<function id="Volume">
    <meta type="Amplifier">
        <role>VolumeChange</role>
        <inputType>null</inputType>
        <outputType>null</outputType>
    </meta>
    <mainComponent id="volume">
        <type>int</type>
        <max>30</max>
        <min>0</min>
        <default>15</default>
        <label>VolumeChange</label>
    </mainComponent>
    <command>
        <method>volume</method>
        <parameter>volume</parameter>
    </command>
</function>
```

たとえば，ユーザが Amplifier A を Amplifier B に重ねた場合，Amplifier A の優先度が高くなる．これはユーザが直接操作するサービスの優先度が高いことを示す．Amplifier A の int 型の GUI コンポーネントは，Amplifier B の int 型の GUI コンポーネントと合成

する．Amplifier A の boolean 型の GUI コンポーネントは実際にコマンドを発行するが，Amplifier B はこれを持たないため，GUI コンポーネントを合成できない．この場合，Amplifier B の優先度が低く，Amplifier B の実行は Amplifier A の実行のタイミングに従う．つまり，Amplifier B の実行は Amplifier A の boolean 型の GUI コンポーネントに吸収される．反対に，Amplifier B を Amplifier A に重ねた場合，Amplifier A の boolean 型の GUI コンポーネントは棄却され，int 型の GUI コンポーネントを操作したタイミングでコマンドは実行される．

次に，接続合成の GUI コンポーネントの合成について述べる．デジタルビデオカメラとプリンタがあるとする．デジタルビデオカメラのキャプチャ機能の出力をプリンタに接続する場合，プリンタを制御するカードをデジタルビデオカメラの右側に並べる．左側にあるカードの出力が右側にあるカードの入力に接続される．これらの機能の CUIL 記述例を示す．

DIGITAL VIDEO CAMERA:

```
<function id="capture">
    <meta type="DigitalVideoCamera">
        <role>Capture</role>
        <inputType>null</inputType>
        <outputType>image/jpg</outputType>
    </meta>
    <mainComponent id="capture">
        <type>boolean</type>
        <label>Capture</label>
    </mainComponent>
    <command>
        <method>capture</method>
        <parameter>null</parameter>
    </command>
</function>
```

PRINTER:

```
<function id="printer">
    <meta type="Printer">
        <role>Printout</role>
        <inputType>image/jpg</inputType>
        <outputType>null</outputType>
    </meta>
    <mainComponent id="print_black">
        <type>boolean</type>
        <label>Printout Black/White</label>
    </mainComponent>
    <command>
        <method>printout</method>
        <parameter>null</parameter>
    </command>
```

```
</function>
```

機能と機能のメタ情報より，Digital Video Camera 型の OutputType が JPG Format, Printer 型の Input Type が JPG Format であるため，これらの機能は接続合成可能である．それぞれの機能は一つの mainComponent で構成されるため，これらの GUI コンポーネントを合成する．

### 5.4.3 GUI コンポーネント配置モジュール

GUI コンポーネント配置モジュールでは，合成後の GUI コンポーネントの自動レイアウトを行う．CUIL では GUI コンポーネントはカードに設定されたグリッドに配置される．GUI コンポーネントを合成する際には，カードの重なりによって決定される優先順位に従い，優先順位が高い方のレイアウトに，優先順位が低いほうのレイアウトが吸収される．優先順位とは，カードの重なりである．上に重なるカードの優先順位が高く，下のカードの優先順位が低い．複数のカードが重なる場合，最も上のカードのレイアウトが優先される．図 5.3 に GUI コンポーネント自動配置モジュールの動作を示す.

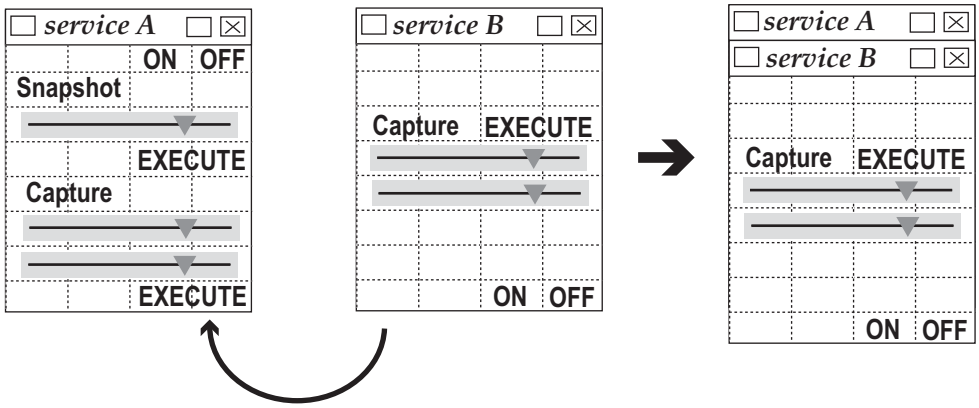



図 5.3: GUI コンポーネント配置モジュールの概念図

*serviceA* と *serviceB* を制御するカードがある．個々のカード内の点線がグリッドである．GUI コンポーネントはこのグリッド内に収められる．*serviceA* のカードの上に *serviceB* のカードを重ねると，電源 ON 機能，電源 OFF 機能，キャプチャ機能が同様な Role Type を持つため，集約合成規則をそれぞれの GUI コンポーネントのセットに対して適用する．合成されたカード上には，*serviceB* のレイアウト情報に基づき，*serviceA* の GUI コンポーネントが吸収されている．

## 5.5　ユーザインタフェース生成部

本節では，ユーザインタフェース生成部について述べる．ユーザインタフェース生成部は，ユーザからの要求に従って CUIL から実際のユーザインタフェースを生成する．実際に CUIL をパースしてユーザインタフェースを生成するためには，CUIC 自身が動作するプラットフォームとの GUI コンポーネントのマッピングが必要である．

### 5.5.1　GUI コンポーネントのマッピング

CUIL では，プラットフォーム非依存にユーザインタフェースを記述するため，特定の GUI ツールキット依存名 GUI コンポーネントの名称を用いない．たとえば，javax.swing.JButton と記述せず,「boolean 型の入力値を持つ GUI コンポーネント」と記述する．そのため，個々の制御用端末上のプラットフォームと CUIL における抽象 GUI コンポーネント型のマッピングを行う必要がある．GUI コンポーネントのマッピングは，個々のプラットフォームに対して記述された PROPERTY.XML に記述する．

PROPERTY.XML

```
<?xml version="1.0" encoding="Shift_JIS"?>
<property>
  <mapping toolkit="Swing">
    <component type="boolean" concrete="JButton" />
    <component type="int" concrete="JSlider" />
    <component type="double" concrete="JSlider" />
              ⋮
  <mapping>
</property>
```

以上の property.xml の記述では，boolean 型の値を入力可能な GUI コンポーネントを Swing の JButton，int 型の値を Swing の JSlider，double 型の値を Swing の JSlider に割り当てている．

## 5.6　コマンド実行部

コマンド実行部は，CIM のイベント制御部から通知を受け，実際に分散コンポーネントに送信するためのコマンドを生成するモジュールである．

### 5.6.1　コマンドオブジェクト

CUIL の記述と特定のサービス合成機構に対する呼び出し方法を抽象化するため，ここでは CIM からのイベントを受け取り，個々の分散コンポーネントの実装に対するコマンドオブジェクトを生成する．コマンドオブジェクトには以下の情報が含まれる．

- 分散コンポーネントの実装プラットフォーム
- 各サービス合成機構におけるサービスの ID
- 呼び出し手続き名
- メソッド呼び出しの引数

分散コンポーネントの実装プラットフォームとは，実際に分散コンポーネントが実装されているサービス合成機構の名称である．コマンド実行部は，各サービス合成機構に対応するコマンドオブジェクトを生成し，ミドルウェアアダプタ部にコマンドオブジェクトを渡す.

### 5.6.2 アプリケーションプロトコル

CUIC ではステートレスな遠隔手続き呼び出しのみをサポートし，サービスとユーザインタフェース間のアプリケーションプロトコルのセッションを考慮しない．ステートレスな遠隔手続き呼び出しとは，分散コンポーネントからの応答を考慮しない遠隔手続き呼び出しである．これに対して，ステートフルな遠隔手続き呼び出しとは，分散コンポーネントからの応答を考慮し，分散コンポーネントからの応答によって次の動作を変更できる遠隔手続き呼び出しである．ホームネットワークの性質上，多くの分散コンポーネントはステートレスに制御可能である.

## 5.7 サービス管理部

サービス管理部は現在利用可能なサービスの管理を行う．サービス管理部では，ミドルウェアアダプタ部から通知を受けて現在利用可能なサービスのテーブルを更新する．新規にサービスが発見された場合，サービス管理部はサービスに対して CUIL 取得要求を発行し，各サービスの CUIL をダウンロードする．あらかじめダウンロードしておくことにより，ユーザインタフェースの応答性を向上できる．また，登録されているサービスが利用不可能になったことをミドルウェアアダプタ部から通知されると，現在保持しているサービスのテーブルから該当するサービス，および CUIL を削除する.

## 5.8 ミドルウェアアダプタ部

本節では，ミドルウェアアダプタ部について述べる．ミドルウェアアダプタ部は，サービス検索機能およびコマンド実行機能を持つ．アダプタは CUICs がサポートするサービス合成機構のそれぞれに対して必要であり，ミドルウェアアダプタを CUIC とサービス合成機構の間にはさむことにより，CUICs がサービス合成機構に対する非依存性を実現出来る.

### 5.8.1　サービス検索機能

サービス検索機能は，サービス合成機構が提供するディレクトリサービスに対して現在利用可能なサービスの一覧を取得，およびサービス管理部に対してサービスの状態の変更を通知する．サービスの状態の変更とは，新規サービスの追加，サービスの削除がある．サービス検索機能は設定された時間ごとに個々のディレクトリサービスに対してサービスの状態を問い合わせ，前回のサービス一覧と差異が生じた場合，サービス管理部に状態の変化を通知する．

### 5.8.2　コマンドの実行

CUICs では，分散コンポーネントへのコマンドの発行には遠隔メソッド呼び出しを用いて行う．そのため，ミドルウェアアダプタ部は，個々のサービスに対するスタブオブジェクトを保持する必要がある．各アダプタは，現在利用可能なサービスに対してコマンドを実行するため，新規サービスが見つかった場合に，HTTP プロトコルを用いて各分散コンポーネントのスタブオブジェクトをダウンロードする．CUICs では，各サービスが内部的に WEB サーバを持っているか，別の WEB サーバ上にスタブオブジェクトを持つことが必要となる．

また，サービス検索機能が一定の時間を置いて利用可能なサービスを変更するため，検索周期とサービス合成機構からサービスが削除されたタイミングによって，コマンドが実行不可能な場合がある．その場合，エラーをコマンド実行部に通知する．

## 5.9　本章のまとめ

本章では，カードのメタファを用いたユーザインタフェース合成機構 CUICs の設計について述べた．はじめに CUICs 全体の構成と動作概要を示し，その後各モジュールの機能要件について述べた．特に，ルールベースモデルを用いた合成アルゴリズム，および GUI コンポーネントの合成アルゴリズム，GUI コンポーネントの自動配置について詳細に述べた．また，CUICs はサービス合成機構に対する非依存性を実現するために，各サービス合成機構に対するアダプタを持つ．CUICs はアダプタを介して分散コンポーネント間の通信を行う．現時点での CUICs の分散コンポーネントに対するコマンドの発行には遠隔手続き呼び出しを用いるため，スタブオブジェクトを動的に HTTP プロトコルを用いてダウンロードする．

# 第 6 章

# カードのメタファを用いたユーザインタフェース合成機構 CUICs の実装

本章では，カードのメタファを用いたユーザインタフェース合成機構 CUICs のプロトタイプシステムの実装について詳細に述べる．はじめに実装概要を述べ，第 5 章で述べた各モジュールの実装について詳細に述べる．

## 6.1 実装概要

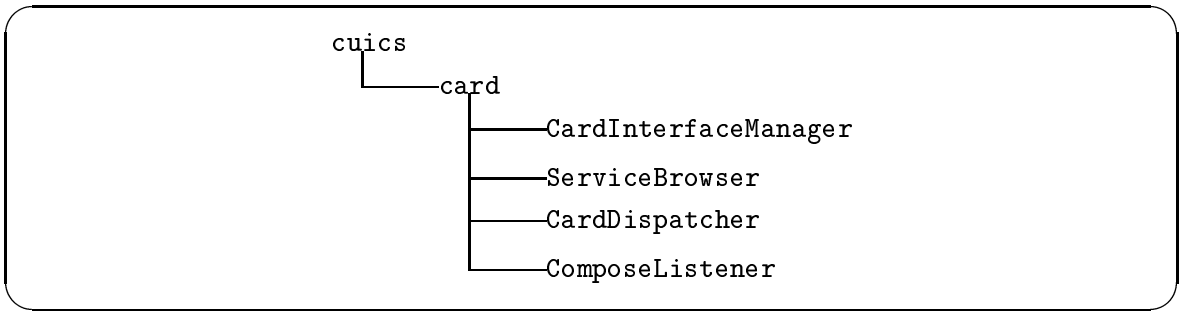
CUICs の実装には Java 言語を用いた．Java 言語を用いた理由は，Java 言語のプラットフォーム非依存性が挙げられる．Java 言語で記述したソフトウェアは JVM(Java Virtual Machine) が動作するプラットフォームであれば，ソフトウェアを再コンパイルせずに動作できる．また，サービス合成機構としては Jini を用いた．Jini はホームネットワークを対象とする分散ミドルウェアであり，CUICs の対象と合致する．現在，Jini における通信機構は多く提案されているが，CUICs では Jini が標準でサポートする RMI[18] を用いた RPC 機構を採用した．また，ディレクトリサービスとしては，Jini が提供する JiniLookupService[20] を用いた．表 6.1 に CUICs の実装環境を示す．

表 6.1: CUICs の実装環境

| 項目 | 説明 |
|---|---|
| ハードウェア | AMD Athlon XP1900+ |
| OS | Windows2000 |
| 言語 | j2sdk1.4.1 |
| サービス合成機構 | Jini1.2.1 |
| 通信機構 | RMI |

## 6.2 カード型インタフェースマネージャ部の実装

CIM の実装は，cuics.card パッケージ内のクラス群から構成される．以下に cuics.card パッケージ内の主要なクラスを挙げる．




CardInterfaceManager クラスが CUICs 全体のフレームであり，ServiceBrowser は利用可能なサービス一覧を表示するためのパネルである．また，CardDispatcher クラスはカードの重なりを判定するクラスであり，カードが重なるとカードの合成を促すボタンを表示し，ユーザがボタンを押すことによってカードに優先順位を設定して合成制御部に送る．図 6.1 に CIM の実装画面を示す．

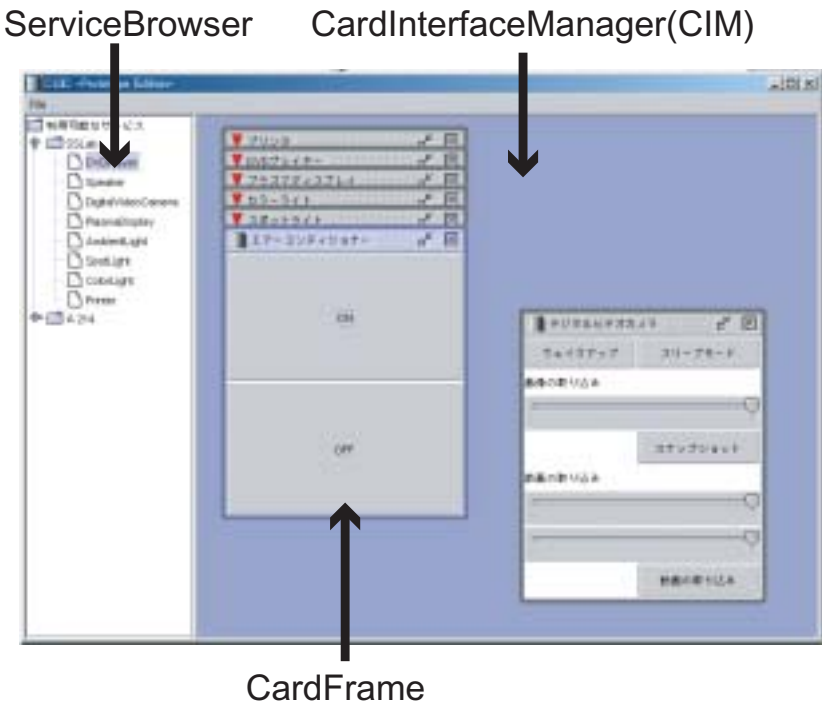



図 6.1: 提案する CIM の画面

### 6.2.1 カードの表示と重なりの判定

CardInterfaceManager クラスは，ユーザインタフェース生成部からの要求によってカードを表示するためのフレームである．CardInterfaceManager は画面上のカードの状態が変更した時に CardDispatcher クラスに対して重なり判定を要求する．重なり判定が出た場合，CardDispatcher は自身を引数にして CIM 上にユーザに合成を促すボタンを表示する．合成ボタンが押された場合，CIM は ComposeListener の dispatch メソッドを呼び出し，カードの合成処理を行う．以下に，以上の処理を行うメソッド群を示す．

```
public class CardInterfaceManager {
  public void addCardFrame(CardFrame card) {--省略--}
  public void addComposeButton(ComposeListener listener) {--省略--}
}
```

```
public class CardDispatcher implements ComposeListener {
  public void check(Vector allCards, CardFrame card) {--省略--}
  public void dispatch() {
    CardComposer composer = CardComposer.getInstance();
    composer.compose(CardFrame srcCard, CardFrame tgtCard);
  }
```

```
}
```

```
public interface ComposeListener {
  public void dispatch();
}
```

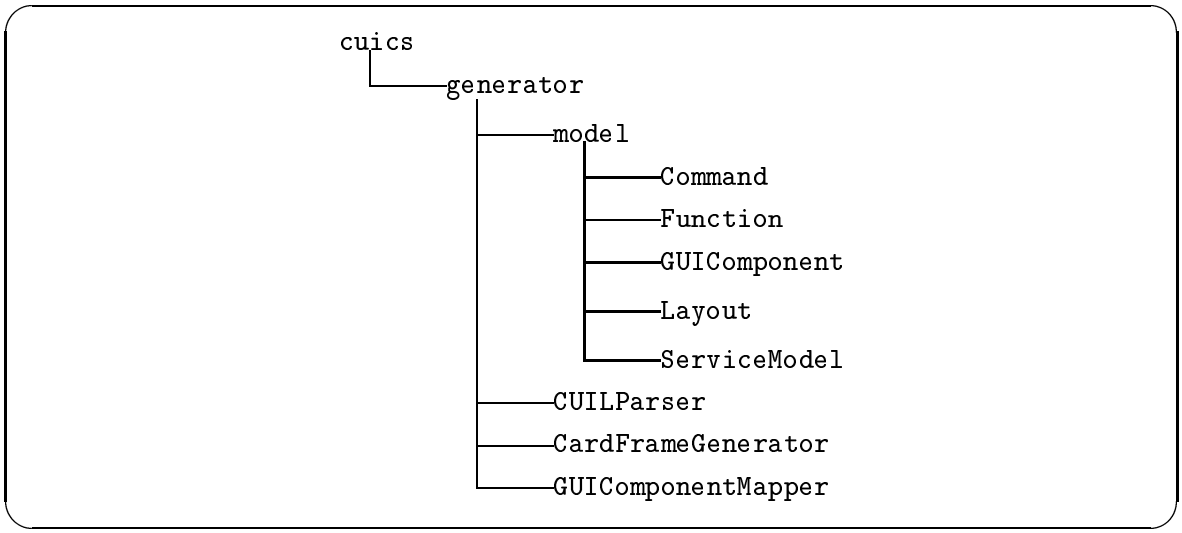
### 6.2.2 利用可能サービスの表示

ServiceBrowser クラスは，サービス管理部からの要求によって自動的に利用可能なサービス一覧を変更する．ServiceBrowser のインタフェースを以下に示す．

```
public interface ServiceBrowser {
  public void addService(ServiceID id, String serviceName);
  public void removeService(ServiceID id);
}
```

## 6.3 ユーザインタフェース生成部

ユーザインタフェース生成部の実装は，cuics.generator パッケージ内のクラス群から構成される．cuics.generator パッケージの概要を以下に示す．cuic.generator パッケージは，cuics.generator.model パッケージを持つ．cuics.generator.model パッケージは，CUICs におけるカードのデータ構造を構成するクラス群で構成される．




CUILParser はサービス制御部から保存してある CUIL を受け取り，ServiceModel を生成するクラスである．ServiceModel は，CUIL から生成されたカードのデータ構造で

ある．CardGenerator クラスでは，生成された ServiceModel をもとに CardFrame を生成し，CardFrame に CardEventHandler を追加する．CardEventHandler は，カードが操作された際にイベントをコマンド実行部に通知するイベントハンドラである．その後，CardGenerator は生成した CardFrame を CardInterfaceManager に登録する．

CardInterfaceGenerator がカードを生成する際には，GUIComponentMapper で実際の GUI コンポーネント名と CUIL 内部での抽象 GUI コンポーネント名のマッピングを参照する．図 6.2 に CardGenerator モジュールによって生成された DigitalVideoCamera のカード，図 6.3 に Airconditioner のカードを示す．


図 6.2: 自動生成された DigitalVideoCamera の UI

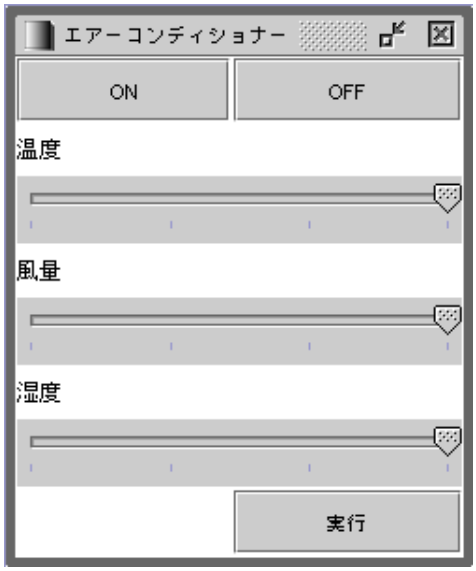
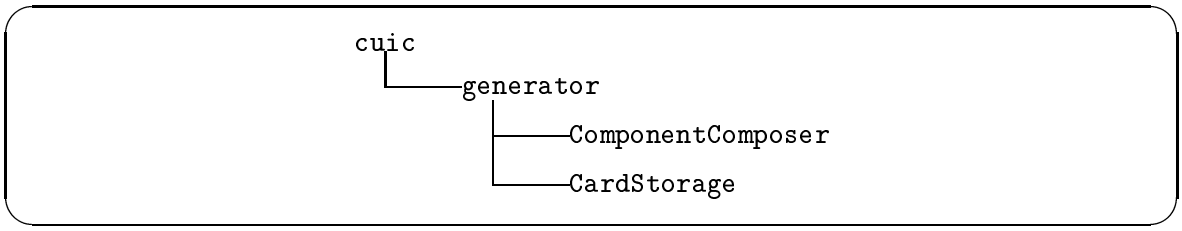



図 6.3: 自動生成された AirConditioner の UI

## 6.4 合成制御部

合成制御部の実装は，cuic.composer パッケージ内のクラス群から構成される．cuics.composer パッケージの概要を以下に示す．




ComponentComposer は，CIM からの合成要求を受け取り，分散コンポーネント間の関係定義，および GUI コンポーネントを合成する CUICs のコアモジュールである．

ComponentComposer により，合成されたカードの cuic.generator.ServiceModel を生成する．生成した ServiceModel は cuics.generator.CardGenerator に渡され，CIM 上に表示される．また，次回生成時の高速化をはかるため，ServiceModel を CardStorage に保存する．これにより，同様な ID を持つ ServiceModel の再合成を要求されたとき，より高速にカードを合成可能である．

### 6.4.1 分散コンポーネント間の関係定義の実装

第 5 章で述べた分散コンポーネント間の関係定義モデルの実装を示す．ここでは，CIM によって設定されたカードの優先順位に従って分散コンポーネント間の関係を定義する．

```
/** src は重ねる方，tgt は重ねられる方のカード */
public void compose(CardFrame src, CardFrame tgt) {
    //協調動作する分散コンポーネントを入れるデータ構造
    Vector connectList = new Vector();
    Vector aggregateList = new Vector();
    //パースされたカードのデータ構造
    ServiceModel srcModel = src.getServiceModel();
    ServiceModel tgtModel = tgt.getServiceModel();
    //サービスが持つ機能の一覧を取得
    Vector srcFunctions = srcModel.getFunctions();
    Vector tgtFunctions = tgtModel.getFuncitons();

    //分散コンポーネント間の関係を定義
    Enumeration srcNum = srcFunctions.elements();
    Enumeration tgtNum = tgtFunctions.elements();
    while(srcNum.hasMoreElements()) {
        Function srcFunc = srcNum.nextElement();
        while(tgtNum.hasMoreElements()) {
            //I/O Type に基づく協調動作の定義
            String srcInput  = srcFunc.getInputType();
            String tgtOutput = tgtFunc.getOutputType();
            if(srcInput.equals(tgtOutput)) {
                Connect connect = new Connect(srcFunc, tgtFunc);
                connectList.add(connect);
            }
            //Role Type に基づく集約動作関係の定義
            String srcRole   = srcFunc.getRoleType();
            String tgtRole   = tgtFunc.getRoleType();
            if(srcRole.equals(tgtRole)) {
                Aggregate aggregate = new Aggregate(srcFunc, tgtFunc);
                aggregateList.add(aggregate);
            }
        }
    }
}
```

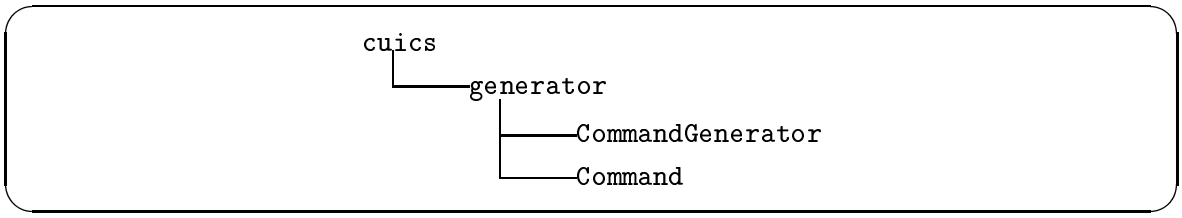
### 6.4.2 GUI コンポーネント間の関係定義の実装

6.4.1 項で求められた関係の集合に基づき，GUI コンポーネントの合成を行う．実装方針としては，機能の集合についてそれぞれ GUI コンポーネントの対応を見る方法を用いる．

ここでは，はじめに重ねる方のカード (srcCard) の GUI コンポーネントと重ねられる方のカード (tgtCard) の GUI コンポーネントの対応で，集約合成規則を適用できる GUI コンポーネントの組を走査する．集約合成規則が適用できる GUI コンポーネントの組が見つかり，tgt 側の GUI コンポーネントが subComponent である場合，該当する tgtCard の mainComponent に対して srcCard 側の GUI コンポーネントの引数をとるように設定する．それが mainComponent である場合は，コマンドの発行条件を srcCard 側の mainComponent にあわせる．集約合成規則の適用後，接続合成可能な GUI コンポーネントの合成規則の適用を行う．接続合成規則では集約合成規則と同様に GUI コンポーネントの組を走査し，両方の分散コンポーネントを制御可能な GUI コンポーネントを生成する．

## 6.5 コマンド実行部

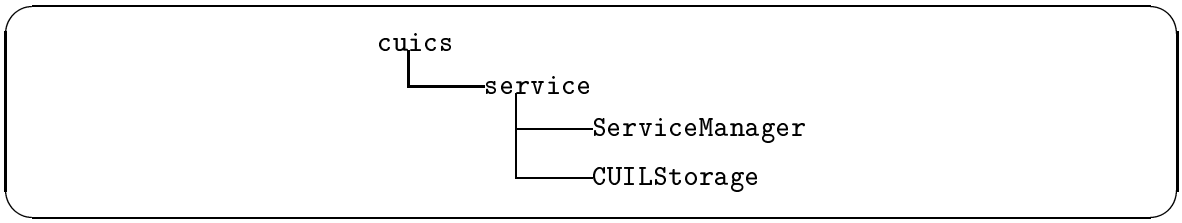
コマンド実行部の実装は，cuics.command パッケージ内のクラス群から構成される．cuics.command パッケージの概要を以下に示す．




CommandGenerator は，cuics.card.CardEventHandler より通知されたイベントをもとに，Command オブジェクトを生成する．Command の生成は，ComponentComposer によって生成された ServiceModel を参照し，Adapter の種類，Service の ID に従って行う．生成した Command オブジェクトを Adapter に渡す．

## 6.6 サービス管理部

サービスマネージャ部の実装は，cuics.service パッケージ内のクラス群から構成される．cuics.service パッケージの概要を以下に示す．

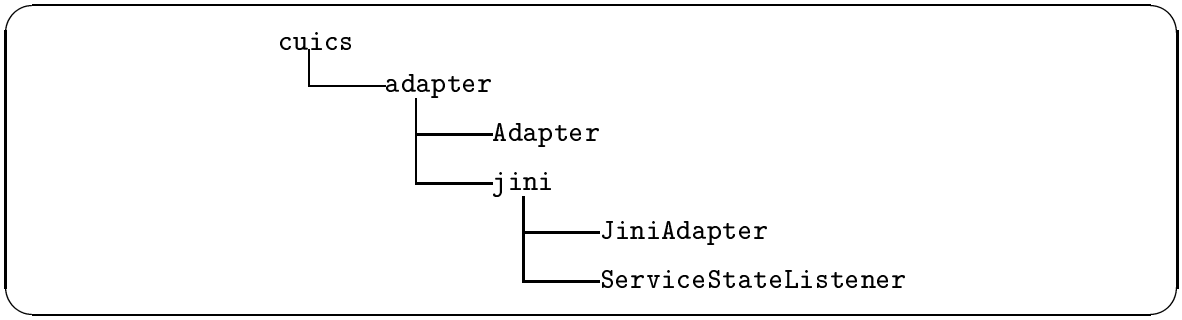



CUILStorage は，JiniAdapter より受け取った CUIL と現在利用可能なサービスの一覧を作成する．利用可能なサービスの一覧は，ServiceBrowser によって参照される．サービスの一覧は Hashtable にサービスの ID を key，CUIL を value として保持する．ServiceManager は，Adapter からのサービス変更通知を受けて Hashtable の内容を変更し，これを ServiceBrowser に通知する．以下に ServiceManager のメソッド一覧を示す.

```
public interface ServiceManager() {
}
```

## 6.7 ミドルウェアアダプタ部の実装

ミドルウェアアダプタ部の実装は，cuics.adapter パッケージ内のクラス群から構成される．cuics.adapter パッケージの概要を以下に示す.




Adapter はインタフェースであり，これを実装するクラスが個々のサービス合成機構に対する実際のアダプタである．以下に Adapter の概要を示す.

```
public interface Adapter() {
  public void execute(Command cmd);
}
```

JiniAdapter は，Jini プラットフォームにおける Adapter の実装クラスである．CUICs のプロトタイプシステムでは，JiniAdapter のみを実装する．JiniAdapter は，JiniLookupService に対してサービスの一覧取得要求を出す．JiniAdapter は JiniLookupService とのト

ランザクションを処理する ServiceStateListener を LookupDiscoveryListener に登録する．LookupDiscoveryListener は Jini が提供するユーティリティクラスである．LookupDiscoveryListener は JiniLookupService から，(1) サービスが新規に追加された，(2) サービスが削除されたことを通知するメッセージを受け取り，ServiceStateListener を呼び出す．JiniAdapter はそれを受けて，ServiceManager を呼び出す．

### 6.7.1 サービスの実装

プロトタイプ実装では，CUICs は Jini に対するアダプタのみを実装する．そのため，CUIC は RMI を用いてサービスを制御する．また，CUICs は RMI を用いてサービスを制御するため，JavaVM が動作する計算機上でサービスが動作することを前提とする．以下にサービスが実装するインタフェースを示す．

```
public interface Service extends Remote {
    public String getCUIL() throws RemoteException;
    public void someCommand(Command cmd) throws RemoteException;
}
```

JiniAdapter は Command オブジェクトに記載された URI に従ってサービスのスタブオブジェクトを HTTP 経由でダウンロードし，取得したスタブオブジェクトを介してサービスとの通信を行う．

## 6.8 本章のまとめ

本章では，カードのメタファを用いたユーザインタフェース合成機構 CUICs のプロトタイプシステムの実装について述べた．はじめに CUIC の実装環境について述べ，その後 CUIC を構成する個々のパッケージについて詳説した．

# 第 7 章

# CUICsの評価

本章では，定量的評価として **CUICs** の基本性能の評価を行いプロトタイプシステムのボトルネックの検証を行う．基本性能の評価としては，サービス一覧取得に必要な時間，ユーザインタフェースの生成および合成時間，コマンド発行時間の検証を行う．また，定性的評価として関連研究との比較を行い，本システムの有用性を評価する．

## 7.1 定量的評価

本節では，CUICs のプロトタイプシステムの基本性能の評価について述べる．

### 7.1.1 測定環境

表 7.1 に評価に用いた計算機環境を示す．評価には PC を用いたが，750MHz 程度の CPU を持つ計算機は今後 PDA や携帯電話にも搭載されることが想定可能である．CUICs のプロトタイプシステムは PC 上でのみ動作するが，将来的に PDA や携帯電話上での動作を想定している．

表 7.1: 測定に用いた計算機環境

| CPU | Memory | VRAM | OS |
|---|---|---|---|
| 750MHz | 256MB | 8MB | Vine Linux 2.6 |

### 7.1.2 測定手法

以下に CUICs の基本性能の評価のための 3 つの評価項目を示す．() 内の項目が各評価項目における評価軸である．また，表 7.2 に評価に用いた CUIL の概要を示す．測定には，3 種類のライト，Plasma Display，Airconditioner，WebCamera，DVDPlayer を用いた．個々の項目は左よりファイルサイズ，機能数，GUI コンポーネント数を示している．

(1) **利用可能なサービス一覧の取得 (サービス数，時間)**
CUICs のクライアントが JiniLookupService に対して現在利用可能なサービスの一覧を取得するために必要な時間を測定する．評価軸として利用するサービス数は，JiniLookupService において利用可能なサービス数の総数を示している．

(2) **ユーザインタフェースの生成 (ファイルサイズ，機能数，GUI コンポーネント数，時間)**
CUIL をサービスからダウンロードして CUICs 内部で使用するデータ構造へパースする処理，およびカード生成に必要な時間を測定する．評価軸として，CUIL の機能数，GUI コンポーネント数を用いた．

(3) **ユーザインタフェースの合成 (機能数，GUI コンポーネント数，時間)**
パースされたデータ構造を第 3 章で述べた合成規則を適用するために必要な時間を測定する．評価軸として，(2) と同様に処理に必要な機能数，および GUI コンポーネント数を用いた．

表 7.2: 測定に用いた CUIL の概要

| サービス名 | ファイルサイズ | 機能数 | GUI 数 |
|---|---|---|---|
| Light(Ambient,Spot,Color) | 1.27KB | 2 | 2 |
| PlasmaDisplay | 2.57KB | 4 | 5 |
| Airconditioner | 3.38KB | 5 | 8 |
| WebCamera | 3.31KB | 4 | 9 |
| DVDPlayer | 5.17KB | 9 | 12 |

### 7.1.3 利用可能なサービスの一覧取得時間

図 7.1 にサービス一覧の取得に必要な処理時間を示す．横軸が利用可能なサービス数を表し，縦軸が処理時間を示す．処理時間は 1000 回計測した値の平均値を示す．CUICs では，新規にサービスが追加される度にサービスへの参照を取得するため，これらの処理がボトルネックになる可能性がある．しかし，図からもわかるように，クライアント起動時に行う利用可能なサービスの取得要求には 15ms 程度の時間を要するが，単一のサービスの参照の取得に必要な処理時間は 6ms 程度である．そのため，バックグラウンドでサービスへの参照を更新しつづける処理はボトルネックにならないことが分かる．

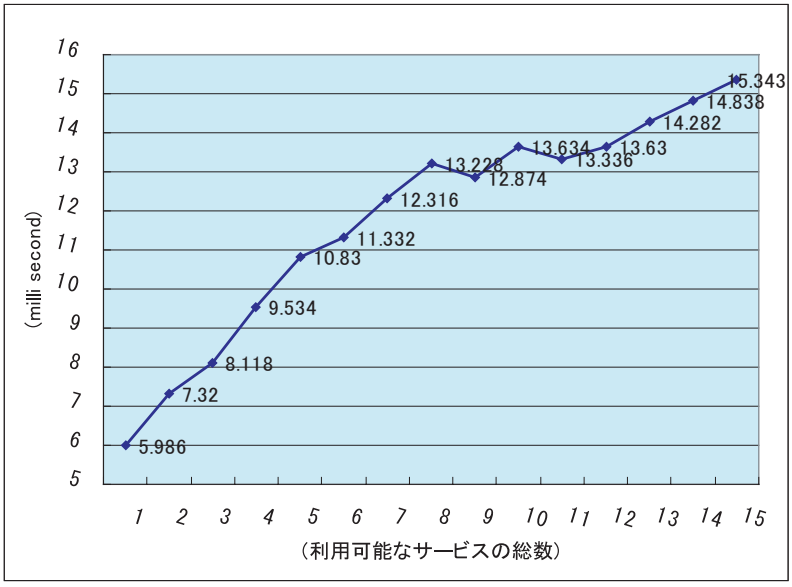



図 7.1: 利用可能なサービスの一覧取得時間

### 7.1.4　ユーザインタフェース生成時間の測定

表 7.3 に表 7.2 で述べたサービスに対するユーザインタフェースの生成時間の測定結果を示す．カードの生成時間は，(1) サービスから CUIL をダウンロードする時間，(2)CUIL を CUICs 内部で利用するデータ構造にパースするための処理時間，(3) カードの生成時間，および (4) カードへの表示処理の 4 つの処理に分類できる．表 7.3 では以上の処理をそれぞれ 1000 回づつ測定し，平均値を求めたものである．

これらの平均処理時間より，(1)，(2) の処理時間はサービスの機能数に関係なく処理が行われ，(3)，(4) の処理時間はカードを構成する GUI コンポーネント数に依存することが分かる．特にカードを表示する処理は GUI コンポーネントの総数と強く関連している．また，(1) の処理は RMI を用いて行うため，処理時間の増加が予想されたが，測定結果より 10ms 以内に処理が終了していることがわかる．

以上の測定結果より，CUICs のプロトタイプシステムにおいて，ユーザがサービス一覧より特定のサービスを選択してからカードは概ね 50ms 前後で表示される．したがって，プロトタイプシステムとしてユーザインタフェースに求められる応答性を満たす処理時間でユーザインタフェースを生成していると言える．しかし，高速な処理時間の理由としては，プロトタイプシステムでは GUI コンポーネントのラベルに画像を用いない点や，測定に用いた CUIL の機能数および GUI コンポーネント数が少ないことがあり，今後これらの機能追加によって処理時間が大幅に増加する可能性がある．

表 7.3: カード生成時間の測定結果 (単位:milli second)

| サービス名 | DL 時間 | パース時間 | 生成時間 | 表示時間 | 総時間 |
|---|---|---|---|---|---|
| Light(Ambient,Spot,Color) | 4.40ms | 12.16ms | 16.52ms | 6.30ms | 39.38ms |
| Plasma Display | 4.23ms | 9.14ms | 15.31ms | 10.94ms | 39.62ms |
| Airconditioner | 5.25ms | 7.19ms | 18.15ms | 29.00ms | 59.59ms |
| DVDPlayer | 9.61ms | 7.14ms | 18.48ms | 25.49ms | 60.72ms |
| WebCamera | 5.17ms | 7.64ms | 24.70ms | 30.21ms | 67.72ms |

### 7.1.5　ユーザインタフェース合成時間の測定

表 7.4 に合成規則適用に必要な処理時間の測定結果を示す．合成規則として，同一の機能を構成する GUI コンポーネントを集約する合成規則と，機能間の入出力を接続するための GUI コンポーネントを生成する合成規則がある．これらの合成規則は一対のカードに対して適用され，集約合成規則，接続合成規則は同時に適用される．また，これらの合成規則の適用には，合成対象となるサービスが持つ機能数，および合成対象となる機能数に関係する．そのため，表 7.4 には合成対象となる機能数を同様に表示した．左より，合成対象となるサービスの組み合わせ，対象となる機能数，合成規則適用時間，カード表示時間，総時間を示している．サービスの組み合わせとしては，集

約合成規則のみが適用される Spot Light と Ambient Light の組み合わせ，および合成されたカードと Color Light の組み合わせ，Web Camera と Plasma Display の組み合わせを評価した．なお，表の測定結果は 7.1.4 項と同様に 1000 回の測定結果の平均である．

測定結果より，カード間の合成規則適用にかかる処理時間は 20ms 前後で終了しており，全体としても 40ms 前後で処理が終了している．前項のユーザインタフェース生成時間の測定結果とほぼ同様な数値が出た理由として，カード生成に必要な CUIL のダウンロードやパース処理が必要ない点が考えられる．これらの処理と合成規則適用に必要なコストはほぼ等価である．これにより，CUICs のプロトタイプシステムにおいて，ユーザインタフェースの合成処理は高速に行われることが分かる．

表 7.4: カード合成時間の測定結果 (単位:milli second)

| 組み合わせ | GUI 数 | 合成時間 | 表示時間 | 総時間 |
|---|---|---|---|---|
| **(1)** Spot Light + Ambient Light | 4 → 2 | 16.70ms | 22.11ms | 38.81ms |
| **(2)** (1) + Color Light | 4 → 2 | 14.38ms | 21.76ms | 36.14ms |
| **(3)** WebCamera + Plasma Display | 14 → 3 | 26.23ms | 22.63ms | 48.86ms |

### 7.1.6 測定結果の考察

本節では，CUICs のプロトタイプシステムの基本性能を評価するため，(1) サービス一覧取得処理，(2) カード生成処理，(3) カード合成処理の平均処理時間の測定を行った．(1) の測定結果として，一つのサービスが利用可能になった際に必要な処理時間は平均 6ms であることがわかった．また，(2)，(3) に関しても平均して約 50ms 前後で処理が終了することが分かった．以上の結果より，CUICs は高速に動作することが分かる．

## 7.2 定性的評価

本節では，Document-based Framework[6]，ICrafter[15]，Pebbles[7] の 3 つ関連研究との性質面での比較を行う．こららの関連研究は，4.1 節で述べた ISL，SDL，ASL 等のサービスが提供する機能単位でユーザインタフェース記述を行う言語を提案する研究である．以下にそれぞれの研究の概要を示し，CUICs との相違点を述べる．

### 7.2.1 Document-based Framework

Document-based Framework は，ISL (Interface Specification Language) を用いてユーザインタフェースを記述し，制御用端末非依存にユーザインタフェースを生成する機構である．各制御用端末上に ISL のインタープリタが配置され，ISL 内で記述された抽象的な GUI コンポーネント型と，特定の GUI ツールキット依存な GUI コンポーネント

間の一致を行う．GUI ツールキットとしては，Tcl/Tk および独自の GUI ツールキットである MASH ツールキットを用いている．

ISL の特徴としては，(1)ISL から動的に生成されるユーザインタフェースにプログラミング言語を用いて記述されたユーザインタフェースをプラグイン可能である点，(2) 同様な機能を制御するユーザインタフェースの集約が挙げられる．(1) は抽象的に記述されたユーザインタフェース記述から動的にユーザインタフェースを生成するより，プログラミング言語で記述されたユーザインタフェースの記述の方がより詳細にユーザインタフェースを定義可能であるための機構である．この機能は CUICs のプロトタイプシステムではサポートしない．この機能をサポートしない理由として，CUICs では GUI コンポーネント自身をコマンドを発行する MainComponent と引数を設定する SubComponent に分離するため，特定の機能に対する GUI コンポーネントの設定を判断できないためである．Document-based Framework では GUI コンポーネントの集合自体をあらかじめ定義するため，プログラミング言語で記述されたユーザインタフェースと動的に生成されるユーザインタフェース間の入れ替えが可能である．CUICs のように特定の機能に対する GUI コンポーネントの集合を静的に定義しないことにより，個々のサービスに対するユーザインタフェースの多様性を実現できる．同様に，(2) も Document-based Framework では特定の機能に対する GUI コンポーネントの集合を静的に記述するのに対し，CUICs では MainComponent と SubComponent の差異を考慮している点で異なる．

### 7.2.2 ICrafter

ICrafter は，Document-based Framework と同様に機能単位でユーザインタフェースを記述し，制御用端末非依存にユーザインタフェースを生成する機構である．ICrafter では，SDL(Service Description Language) を用いてユーザインタフェースを記述する．SDL は Java 言語で記述されたサービスのバイトコードからリフレクション API[17] を用いて動的に生成される．

ICrafter の特徴として，(1) 独自のイベントモデルに基づくフレームワーク上での動作，(2) プログラミング言語で記述されたユーザインタフェースとの代替性，(3) 複数のサービスを制御可能なユーザインタフェース生成のためのユーザインタフェース合成機構が挙げられる．(1) による利点はサービス間の関係をより簡易に定義可能であることや利用可能なサービスを単一のディレクトリサービスを利用して管理可能な点があるが，同時に多様なサービス合成機構上で動作できない問題点がある．これに対し，CUICs はサービス間の関係をサービス合成機構非依存に定義するため，利用可能なサービスの情報を個々の制御用端末で集約するオーバーヘッドがあるが，多様なサービス合成機構上で動作可能な利点がある．

また，(3) は CUICs ユーザインタフェース合成機構であるが，以下の 2 つの点で CUICs と異なる．一つは，CUICs がルール生成モデルを用いてサービス間の関係を動的に定義可能である反面，ICrafter ではテンプレート主導モデルを用いるため，あらかじめ静的に定義されたサービスの組み合わせを制御するユーザインタフェースの生成にとど

まる．二つ目は，ICrafter は Document-based Framework と同様に，特定の機能を制御する GUI コンポーネントの集合を静的定義するため，合成アルゴリズムが簡単になる反面，CUICs のように GUI コンポーネント間の多様性を許容できない．CUICs では，同様な機能を持つサービスであっても実装が異なれば GUI コンポーネントの集合も異なることを想定し，同様な機能を制御する GUI コンポーネント間の差異を吸収する機構を持つ．(2) は Document-based Framework で提案された機構と同様である．

### 7.2.3 Pebbles

Pebbles は，Document-based Framework や ICrafter と異なり，PDA 上にユーザインタフェースを動的に構築するするための機構である．ユーザインタフェース記述言語として，サービスが提供する機能単位でユーザインタフェースを記述する ASL (Appliance Specification Language) を用いる．Pebbles の特徴は，(1)GUI だけでなく音声ユーザインタフェースを同様の ASL から生成可能である点，(2) サービスの状態を反映してユーザインタフェースを生成する点が挙げられる．

(1) は，個々の GUI コンポーネントのラベルをテキストで表示する代わりに，音声ファイルを示すタグを追加して実現されている．たとえば，

```
<text-to-speech  recording="speech.au">
```

等のように音声ファイルへの参照を記述する．これにより，PDA 上で GUI を操作した際に，GUI が持つ意味を音声を用いてユーザに通知可能である．しかし，音声は GUI コンポーネントのラベルとして機能するため，画面表示能力の低い制御用端末では動作できない問題点がある．音声のみのユーザインタフェースを実現するためには，現在 GUI コンポーネントとして表示されている選択肢をユーザに通知するための変換機能が必要である．Pebbles では，そこまでの機能はサポートされていない．CUICs では，カードのメタファを用いることにより，音声等のマルチモーダリティを実現しない．GUI コンポーネントに特化することにより，より簡易に制御可能なユーザインタフェースの構築を主眼としている．また，(2) に関しては，CUICs のプロトタイプシステムでは考慮されていない．サービスの状態を反映することにより，表示する GUI コンポーネントをより詳細にシステムが管理可能である．そのため，CUICs の今後の拡張としてサービスの状態を管理する機構を組み込む必要がある．

## 7.3 本章のまとめ

本章では，CUICs のプロトタイプシステムの定量的評価，および定性的評価を行った．定量的評価では CUICs を構成する個々のモジュールの処理時間を測定することにより，システムのボトルネックの検証を行った．検証の結果，大きなボトルネックは検証されず，プロトタイプシステムはユーザインタフェースの応答性の要求を満たしている．また，定性的評価では，関連する研究との性質面での比較を行い，関連研究に対する CUICs の優位性，および改善点などを考察した．

# 第 8 章

# 結論

## 8.1 まとめ

本論文では，カードのメタファを用いたユーザインタフェースの動的合成機構である，**CUICs (Card-oriented User Interface Composition System)** を提案し，システムの設計およびプロトタイプシステムの実装，基本性能の評価を行った．ユーザインタフェースの合成は，カードとして表示されるユーザインタフェースをユーザが実行時に重ねることにより，CUICs が協調動作可能な機能の組み合わせを判別し，個々の分散コンポーネントを制御する GUI コンポーネントの集合を動的に合成する．CUICs を用いることにより，ユーザは簡易に複数のサービスを制御可能なユーザインタフェースを利用可能になる．

既存のサービス合成機構は，分散コンポーネント間の通信パスを動的に生成する機構を提供するが，分散コンポーネント間の協調動作を動的に変更するためのフロントエンドを持たないため，利用者に高度な専門知識を要求する問題点がある．これを解決するため，CUICs ではユーザが実行時にサービスを選択し，選択されたサービスを構成する分散コンポーネント間の関係を動的に求めるトップダウンな関係定義モデルを採用した．本論文では，分散コンポーネントの実体の集合から，動的に分散コンポーネント間の関係を定義する関係定義モデルをルール生成モデルと呼ぶ．トップダウンな関係定義モデルとは，ユーザが指定する複数のサービスを構成する分散コンポーネントの集合から，動的に関係を求める関係定義モデルである．トップダウンに分散コンポーネント間の関係を求めることにより，ユーザはサービスを指定するという簡易な操作によってサービス間の協調動作を定義できる．定義される関係として，CUICs では (1) 集約合成規則，(2) 接続合成規則の 2 つの関係を定義している．CUICs は，以上の分散コンポーネント間の関係に従い，個々の分散コンポーネントを制御する GUI コンポーネントの集合を合成する．GUI コンポーネントの集合を合成することにより，ユーザはユーザは個々の分散コンポーネントに分散したユーザインタフェースを用いずに，単一のユーザインタフェースを用いて協調動作する分散コンポーネントを制御できる．

## 8.2 今後の課題

CUICs の今後の課題としては，以下の 3 点が挙げられる．

### 8.2.1 CUICs の実装の拡張

現在の CUICs は Java 言語を用いて実装される．そのため，現時点では CUICs は JVM が動作しない制御用端末では動作できない．また，Java 言語は中間コードをインタープリタで変換するため，ネイティブコードに比べて動作が遅い問題点がある．今後の課題としては，より計算能力の低いデバイス上でも動作できるように，多様なプラットフォームで動作可能な CUICs の実装を行うことが挙げられる．

### 8.2.2 多様なサービス合成機構への対応

現在，CUICs は下位の通信機構へのアダプタとして JiniAdapter のみを実装する．そのため，サービス合成機構横断的にサービスを制御できない問題点がある．今後の方針として，CORBA[11]，UPnP[23] 等の分散ミドルウェアに対するアダプタを実装する．

### 8.2.3 CUIL の拡充

本システムでは，プラットフォーム非依存で合成可能なユーザインタフェース記述言語として CUIL を提案した．しかし，CUIL は仕様策定段階にあり，これを用いて複雑な機能を持つサービスを制御するユーザインタフェースを記述しにく問題点がある．現段階では，CUICs の想定するサービスはステートレスなプロトコルのみで制御可能であるが，今後ステートフルなプロトコルに対応予定である．

# 謝辞

本研究の機会を与えてくださり，絶えず丁寧なご指導を賜りました，慶應義塾大学環境情報学部教授徳田英幸教授に深く感謝致します.

慶応義塾大学徳田, 村井，楠本，中村，南研究会の諸先輩方にはお忙しい中貴重な助言をいただきました．特に，政策・メディア研究科柳原正氏，松宮健太氏，政策・メディア研究科前期博士課程由良淳一氏には，本論文執筆にあたって多くの励ましとご指導を得ました．この場を借りて深く感謝の念を表します.

最後に，研究の日々を共に過ごした青木俊氏，高橋元氏その他の友人に深く感謝し，謝辞と致します.

平成 15 年 2 月 24 日
門田 昌哉

# 参考文献


[1] ECHONET CONSORTIUM. Echonet consortium, 2002. http://www.echonet.gr.jp.

[2] Microsoft Corporation. Distributed component object model (dcom), 1998. http://www.microsoft.com/com/tech/DCOM.asp.

[3] Microsoft Corporation. Activex controls, 1999. http://www.active-x.com/, http://www.microsoft.com/com/tech/ActiveX.asp.

[4] Jacqueline Floch and Rolv Braek. Towards dynamic composition of hybrid communication services. In *6th International Conference on Intelligence in Networks (Smartnet 2000)*, 2000.

[5] Object Management Group. Idl to java language mapping specification, Aug 2002. http://cgi.omg.org/docs/formal/02-08-05.pdf.

[6] Todd D. Hodes and Randy H. Katz. A document-based framework for internet application control. In *2nd USENIX Symposium on Internet Technologies (USITS'99), Boulder, CO,*, Oct 1999.

[7] Michael Higgins Joseph Hughes Thomas K. Harris Roni Rosenfeld Jeffrey Nichols, Brad A. Myers and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *ACM Symposium on User Interface Software and Technology, 27-30*, Oct 2002. http://www.cs.cmu.edu/ pebbles/puc/.

[8] Richard F. Rashid Michael B. Jones and Mary R. Thompson. An Interface Specification Language for Distributed Processing. In *12th ACM Symposium on Principles of Programming Languages*, Jan 1985.

[9] Jin Nakazawa, Masayuki Iwai, and Hideyuki Tokuda. Dynamic message-path generation for improvised composition of networked appliances services. In *International Workshop on Networked Appliance (IWNA4)*, January 2002.

[10] Jin Nakazawa, Yoshito Tobe, and Hideyuki Tokuda. On dynamic service integration in vna architecture. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 7, No. E84-A, pp. 1610–1623, July 2001.

[11] Object Management Group. The Common Object Request Broker Architecture and Specification 2.2ed CORBA Event Service, February 1998.

[12] Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for building composite web services http://www2002.org/cdrom/alternate/786/. In *The Eleventh International World Wide Web Conference (Web E ngineering Track)*, 2002.

[13] Distinguished Engineer Member IBM Academy of Technology IBM Software Group Prof. Dr. Frank Leymann. Web services flow language (wsfl 1.0), May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modem Approach*. Prentice-Hall, 1995.

[15] Armando Fox Pat Hanrahan Shankar R. Ponnekanti, Brian Lee and Terry Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Ubicomp 2001: Atlanta, GA, USA, 56-75*, 2001.

[16] Mihut Ionescu Sirish Chandrasekaran, Samuel Madden. Ninja paths: An architecture for composing services over wide area networks. http://ninja.cs.berkeley.edu/.

[17] Sun Microsystems Inc. Java core reflection api and specification, 1996.

[18] Sun Microsystems Inc. Remote method invocation specification, 1996.

[19] Sun Microsystems Inc. JavaBeans specification, 1997. http://www.java.sun.com/products/embeddedjava/.

[20] Sun Microsystems Inc. *Jini LookupService Specification*, October 2000. http://www.sun.com/jini/specs/core1_1.pdf.

[21] Sun Microsystems, Inc. Enterprise Java Beans Technology, 2001. http://java.sun.com/products/ejb/.

[22] Satish Thatte. XLANG Web Services for Business Process Design, 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/.

[23] Universal Plug and Play Forum. Universal Plug and Play (UPnP), 1999. http://www.upnp.org.