修士論文　　2005年度 (平成17年度)

ホームネットワークにおける多様なユーザのためのサービス連携支援システム

慶應義塾大学　政策・メディア研究科
氏名：高橋 元

修士論文要旨　　2005 年度 (平成 17 年度)

ホームネットワークにおける多様なユーザのためのサービス連携支援システム

論文要旨

ホームオートメーションを構成するサービスの連携定義は，新規デバイスの参加，転居などによる人の入れ替わりなどによりしばしば変更要求が起こる．連携定義を居住者が行うことで，プログラマなどを介在させず，要求に応じたホームオートメーションを即興的に実現できる．また，ホームネットワークでは，多様な記述能力の居住者が多様なユーザインタフェースから連携定義を行うことが想定される．この際，多様なユーザインタフェースを用いて連携を定義する場合，各ユーザインタフェースが生成する連携定義の変換が必要となり，盲目の人が健常者の人が定義した連携を定義・確認することは困難となる．修士研究では，居住者がサービス連携定義を行うために，居住者のメンタルモデルに合わせた連携モデルを採用する連携定義言語 ISPL の設計・実装を行い，ISPL をもとにしたホームオートメーションを実現するモデル駆動型アーキテクチャの設計・実装を行う．また，実際に，ISPL を生成するユーザインタフェースの設計・実装を行い，さらに，実際に構築したホームネットワーク環境 Smart Living Room について述べる．

キーワード：

1 サービス連携　2 ホームネットワーク　3 エンドユーザプログラミング　4 分散コンピューティング

慶應義塾大学　政策・メディア研究科

高橋 元

Abstract of Master's Thesis

A Research of Service Coordination Platform considering Heterogeinity of User in Home Network

Academic Year 2005

Summary

The definitions of composition constructing home automation services are to be changed frequently, according to the entry of new devices or moving of the residents. By enabling the residents to define the compositions, without being aided from any programmer, the desired home automation can be formed instantly. Here, in home network, it is likely that various skilled residents define the compositions using diverse interfaces. To accept this, there is a necessity to convert the definitions which each interfaces generate, and it becomes difficult for the blind people to define or confirm the compositions which people in normal health had defined. In this study, in order to let residents define the compositions of services, the following three steps are implemented. 1. ISPL, a composition model linked defining language 2. A model-linked architecture realizing home automation based on ISPL 3. An user interface processing ISPL Furthermore, based on these, we actually constructed a home network environment " Smart Living Room ".

Keyworkds:

1 Service Composition 2 Home Network 3 Enduser Programming
4 Distributed Computing

Keio University Graduate School of Media and Governance
TAKAHASHI Gen

目 次

図 目 次

表 目 次

第 1 章

序論

1.1 本研究の背景ーホームネットワークにおけるサービス連動アプリケーションー

現在，デスクトップコンピュータ，プリンタ，AV 機器などの情報家電機器等の協調動作を実現するソフトウェア的な接続性が整いつつある．例えば，従来から CORBA[7]，RMI[8]，VNA[5] などの分散オブジェクト技術が研究されてきた．近年，さらに情報家電間の通信を対象とした組込み CORBA[9] などとして発展している．また，無線や超音波を利用した位置情報センサが開発されており，このような技術をホームネットワークに導入することによって，例えば，情報家電の状態に応じて他の情報家電等が連動して動作したり，入退室などのユーザの状況に応じて情報家電等が連動するアプリケーションが可能となる．

これらのアプリケーションにより利用者の生活支援を行うことで，利用者の利便性が向上だけでなく，老人や非健常者にとって，肉体的な負担が強いられない生活の実現が予測される．

1.2 本研究の問題意識

センサ情報から利用者や部屋の状況を取得し，ネットワーク接続性を持つ情報家電等のサービスを制御するサービス連携アプリケーションが可能となった．サービス連携アプリケーションを実現するサービスの連動定義は，新規デバイスの参加，転居などによる人の入れ替わりなどによりしばしば変更要求が起こる．

情報家電等の様々なサービス間の連動を実現するホームネットワークにおいて，サービスの連動の定義が静的であるため，新規デバイスの参加等のユーザの置かれる状況の変更に伴うサービス連動定義の変更要求に柔軟に対処することができない．

このことから，ホームネットワークでは，ユーザの置かれる環境の変化に応じて移り変わるユーザの要求とサービスの連動定義の整合性を維持する必要がある．例えば，電話をとると TV が停止するというホームネットワークにおける連動アプリケーショ

ンを考える．新規に CD プレーヤが参加した場合，停止対象は CD プレーヤも含まれるかもしれない．また，TV は停止するけれども，CD プレーヤに関しては接続先アンプのボリュームを下げるだけかもしれない．

このような環境の変化による連動定義の動的な変更の実現は，サービス連動定義が静的であると実現不可能である．従って，ユーザが自らユーザインタフェースなどを用いてサービス連動を要求に応じて再定義することによって，要求に応じたサービス連動アプリケーションを即興的に実現することが望まれる．

また，ホームネットワークでは，多様な記述能力の居住者が多様なユーザインタフェースから連動定義を行うことが想定される．多様なユーザインタフェースにより連動を定義する場合，各ユーザインタフェースが生成する連動定義の変換が必要となり，盲目の人が健常者の人が定義した連動を定義・確認することは困難となる．

1.3 本研究の目的と概要

ホームオートメーションを構成するサービスの連携定義は，新規デバイスの参加，転居などによる人の入れ替わりなどによりしばしば変更要求が起こる．連携定義を居住者が行うことで，プログラマなどを介在させず，要求に応じたホームオートメーションを即興的に実現できる．また，ホームネットワークでは，多様な記述能力のユーザが多様なユーザインタフェースから連携定義を行うことが想定される．多様なユーザインタフェースにより連携を定義する場合，各ユーザインタフェースが生成する連携定義の変換が必要となり，盲目の人が健常者の人が定義した連動を定義・確認することは困難となる．本研究では，居住者がサービス連携定義を行うために，居住者のメンタルモデルに合わせた連携モデル連携定義言語 ISPL の設計・実装を行い，ISPL をもとにしたホームオートメーションを実現するモデル駆動型アーキテクチャ [9] の設計・実装を行う．また，実際に，ISPL を生成するユーザインタフェースの設計・実装を行い，さらに，実際のホームネットワーク環境を構築する．

1.4 本研究の構成

本論文では，まず 2 章でサービス連動を実現するアプリケーションにおける連動定義に関する考察を行う．3 章では，2 章の考察に基づいた，言語 ISPL の定義を行う．4 章では，ISPL によるモデル駆動アーキテクチャの特徴と設計を述べ，5 章でモデル駆動アーキテクチャのプロトタイプ実装，ISPL を用いたユーザインタフェースの例及び情報家電やセンサが配備された Smart Living Room と Smart Living Room 上のサービス実装例について述べる．6 章で ISPL と他の関連研究との比較を行い，最後に 7 章で本論文をまとめる．

第 2 章

サービス連携に関する考察

本章では，多様なユーザが多様なユーザインタフェースを用いてサービス連動を定義するホームネットワークにおけるサービス支援システムを実現するために，ホームネットワークにおけるサービス連携及びユーザについて述べ，ホームネットワークにおけるサービス支援システムの要件について述べる．

2.1　ホームネットワーク上のサービス

本論文では，ホームネットワークにおける情報家電，ユーザの入退室管理システム，気温・湿度等の環境情報等を取得するセンサ，PC で稼動するアプリケーションなど状態の取得や制御がネットワークから可能なものをサービスと呼ぶ.

2.2　サービス連携アプリケーション

サービス連携アプリケーションとは，前項で述べたサービスが，他のサービスの状態等に応じて，連動して動作するアプリケーションを言う．電話サービスとＡＶ機器であるオーディオサービスの連動の例を挙げる．ユーザは，リビングでオーディオを大音量で聞いていたところ，電話がコールされた．利用者は，オーディオの音量を小さくしたい．このシナリオの場合，電話サービスのコールに応じて，オーディオサービスの音量を制御することによって，ユーザの負担を減らすことができる．

AV ルームの例を挙げる．AV ルームは，利用者 A 及び B により使用されているものとする．情報家電として，DVD プレーヤ，AV アンプ，TV 及び CD プレーヤが置かれている．さらに，無線位置情報センサにより，A 及び B の入室を検知できる．A は主に DVD の視聴を行う．B は主に CD 等のオーディオ機器を利用する．また，A は午後 2 から午後 5 時の間，B は主に午後 11 時以降に AV ルームを利用する．このように，AV ルームは利用者により異なった使われ方をする．従って，利用者の入室から DVD もしくは CD を再生するまでの手続を予め定義し，利用者ごとに自動化することで，AV 機器の操作が容易になる．例えば，A の場合の手続として，A の入室を検知し，DVD プレーヤ，AV アンプ，TV を待機状態から復帰させる．さらに，AV アンプの音量を比較的大き目に設定し，TV の入力切り替えを DVD 入力に変えるといったことが考えられる．また，利用者 B の場合，B の入室を検知し，CD プレーヤ，AV アンプ待機状態から復帰させる．さらに，AV アンプに対して，音量の比較的小さめに設定し，イコライザを予めプリセットされたクラシックに変更する．

2.3　ホームネットワークのユーザ

本節では，本研究におけるホームネットワークユーザの役割及び多様性について述べる.

2.3.1　ユーザの役割

第 2.2 節で述べたサービス連携アプリケーションにおける連携定義は，利用者の要求や環境の変化によって，変わる．新しい情報家電の参加などの利用者の環境の変化にともなう連携定義の変更の要求に柔軟に対応するために，利用者自身が利用者に適したユーザインタフェースを用いて，サービス連携を定義することによって，利用者の

要求とサービス連携アプリケーションの整合性を，連携アプリケーションを開発者等に再構築することを依頼することに比べ，低いコストで保つことができる．

2.3.2 ユーザの多様性

ユーザの役割として，ホームネットワークでは，ユーザが連携アプリケーションを定義することが望まれると述べた．ホームネットワークでは，様々なユーザが想定されるため，あるユーザが定義した連携を他のユーザが再定義や確認できることが望まれる．本項では，連携アプリケーションを定義するユーザの多様性について述べる．

認知の多様性

ホームネットワークのユーザは，子供から数学教員も想定される．従って，サービス連携アプリケーションをユーザが観測する際の，認知モデルも多様となる．例えば，入室に応じて，電灯が消灯する連携アプリケーションの場合，子供はユーザの入室と電灯の消灯の因果関係によって，連携アプリケーションを認知するかもしれない．また，数学者は，子供と同様に因果関係によって認知しつつも，入室時刻と t，電灯の電圧を x とし，t 及び x の関数として認知するかもしれない．

モダリティの多様性

連携を定義するためにユーザは，ユーザインタフェースを用いる．しかし，ホームネットワークユーザは，老人や盲目のユーザも想定されるため，ユーザごとに最適な連携定義のユーザインタフェースに求められるモダリティは多様となる．次に，ユーザインタフェースとして考えられるモダリティについて示す．

文字による記述

文字による記述とは，連携を文字による記法を指す．C 言語のような手続き型，Lisp 言語のような関数型，Prolog のような論理型，Smalltalk のようなオブジェクト指向型がある．

視覚的な記述

視覚的な記述とは，アイコンなどの視覚的な表現を用いるものを指す．実際の記述系として，ToonTalk[11] のような例示による記述，Prograph やペトリネットのようなデータの流れに着目するデータフロー型などがある．

例示による記述

例示による記述とは，ユーザが実際に操作することで，マクロを記述する形で連携を記述する形態を指す．

音声による記述

音声による記述とは，ユーザが発話することで，連携を記述する形態を指す.

実世界指向記述

実世界指向記述とは，[16] で述べられいるように例示による記述を実世界に発展させたもので，ユビキタス環境におけるサービス間の連動を定義するにあたり，ユーザが実際にデバイスの操作を行うことで連携を記述する形態を指す.

2.4 サービス連携定義におけるユーザの認知過程

本節では，ノーマン [6] のユーザのシステム使用時における一連の認知過程について示す.

- 目的・意図の設定目標及び意図の設定とは，仕事から帰宅した時に暗いと寂しいので部屋の電気とテレビがついていてほしいと思う段階である.

- システムに対する行為系列の特定
  システムに対する行為系列の特定とは，実際のシステムをどのように操作するか計画を立てることである.

- 実行
  実行とは，特定した行為系列を元にシステムに働きかけることを言う.

- 知覚
  知覚とは，働きかけた結果，システムの動きを知覚する段階である.

- 状態解釈・評価
  状態の解釈・評価とは，以前立てた目標と意図が満たされているかを評価する段階である.

上記のモデルをサービス連携定義を行うユーザにあてはめると，目的・意図の設定は，サービス連携モデルの構築にあたり，システムに対する行為系列の特定以降は，ユーザインタフェースによる連携定義に相当する.

2.5 認知発達モデル

ジャンピアジェによる認知発達モデル [15] は，感覚－運動期（誕生－ 2 才），前操作期（2-7 才），具体的操作期（7-12 才）形式的操作期（12 才～）に分けれら，具体手的操作期を経た人間は，具体的な直接観測可能な対象に対してのみ，原因ー結果の関係を理解、論理的な分析能力を持つとされている.

2.6　サービス連携モデル

ホームネットワークにおける情報家電や利用者の位置情報を利用したアプリケーションとして AV ルームの例を見た．まず，このような振る舞いから利用者や情報家電等の実世界における空間的状態及び情報家電を表すサービスのソフトウェア的状態を抽出できる．さらに，これらの状態を用いサービス連携をモデル化する場合，離散時間としてのモデル化が適している場合と離散事象としてのモデル化が適している場合がある．以下にモデル化に際する各々の特徴を示す．

ホームネットワークでは，ユーザが連携を定義することが望まれるため，ユーザにとって認知の容易な連携モデルによる連携定義が適している．

離散時間的な連携モデル

状態変数や代数方程式などによりモデル化を行い，状態の変化が連続的な系の挙動のモデル化に適している．

離散事象的な連携モデル

空間的状態及びソフトウェア的状態の集合及びその遷移規則で表現されるため，離散状態の集合からなり，また系並行的な挙動が見られる系のモデル化に適している．

2.7　サービス支援システムの要件

本節では，サービス支援システムの要件について述べる．

ホームネットワークでは，様々なユーザが想定されるため，あるユーザが定義した連携を他のユーザが再定義や確認できることが望まれると述べた．ユーザに適したモダリティは文字，視覚，例示，実世界指向，音声のように様々であるため，連携を定義するユーザインタフェースは，多様なものが求められる．また，ユーザの認知は様々である．ユーザの様々な認知に対応するため，認知モデルごとに様々なユーザインタフェースを用いて連携を定義すると，あるユーザが定義した連携を他のユーザが再定義や確認する際に，各ユーザインタフェースが生成する連動定義の変換が必要となり，盲目の人が健常者の人が定義した連動を定義・確認することは困難となる．

これらから，ホームネットワークでは，多様なユーザインタフェースに共通する一貫した連携定義モデル及びそのモデルを表現する連携定義言語によりサービス制御を実現するサービス支援システムが必要となる．これによって，ホームネットワークでは，様々なユーザが様々なインタフェースから一貫してサービス連携アプリケーションの定義・確認が可能となる．

第 2.4 節でシステム使用時における一連の認知過程を見た．これをサービス連携定義を行う際のユーザの行為に当てはめると，最初の行為は目的・意図の設定となる．従っ

て，容易なサービス連携定義を実現するには，システムに対する行為系列の特定以降のユーザインタフェースと同様目的・意図の設定を支援する連携モデルが重要となる．

第 2.5 節では，ピアジェによる発達心理モデルを見た．この理論から，連携定義モデルとして，原因ー結果の関係を採用し，連携を情報家電などのサービスとその状態及び時間や場所などのコンテクストの因果関係により定義することで，より多様な居住者にとって，ユーザが連携を定義する際の連携モデルの学習コストが低くなると考えられる．

2.8 本章のまとめ

本章では，多様なユーザが多様なユーザインタフェースを用いてサービス連動を定義するサービス支援システムを実現するために，ホームネットワークにおけるサービス連携，ユーザについて述べ，ホームネットワークで求められるサービス連携モデル及びモデルの要件について述べた．次章では，ISPL の定義について述べる．

第 3 章

ISPL

本章では，第2章で述べたによる多様なユーザによるサービス支援システムを実現するサービス連携モデル及びサービス連携定義言語ISPLについて述べる．ISPLの特徴は，サービス連携定義をユーザが認知可能なサービスの名前や状態の因果関係を用いて定義する．これにより，状態をもとにした協調動作定義は，ユーザが実生活において，事象を認知する際に用いられるため，モデル自体の学習コストが低く，多様な居住者にとって理解が容易である．

3.1 ISPLの概要

本節では第 2 章で述べた連携定義モデル及び連携定義言語 ISPL について述べる．ISPL の特徴は，サービスが持つ状態の因果関係の定義による連携定義を行う．サービスが持つ状態とは，ユーザが直接認知可能なサービスの離散状態である．従って，モデルとユーザが認知する対象が近似的であり，記述対象指向である．ISPL では，サービスの制御は，サービスの状態遷移として表現される．図 3.1 で見るような手続き型

```
HANDLE light;//電灯を表すサービス

HANDLE radio;//ラジオを表すサービス
HANDLE television;//テレビを表すサービスサービス

while(true){
  if(light.getState().equals("OFF")){//電灯の状態を監視
    ((TV)television).turnOff();//テレビを消すオペレーション
    ((Radio)radio).turnOff();//ラジオを消すオペレーション
  }
}
```

図 3.1: 手続き指向言語による擬似コード

言語では，メソッド呼び出しや制御構文によって状態の監視やサービスの制御を行い，連動を定義するが，ISPL では，連動定義を状態及び状態の因果関係により表現する．

ジャンピアジェによる認知発達モデルは，感覚－運動期（誕生－ 2 才），前操作期（2-7 才），具体的操作期（7-12 才）形式的操作期（12 才～）に分けれら，具体手的操作期を経た人間は，具体的な直接観測可能な対象に対してのみ，原因ー結果の関係を理解、論理的な分析能力を持つとされている．従って，連携定義言語のモデルとして，原因ー結果の関係を採用し，連携を情報家電などのサービスとその状態及び時間や場所などのコンテクストの因果関係により定義することで，より多様なユーザにとって，ユーザが連動を定義する際の連携モデルの認知が容易になると考えられる．

3.2 ISPLの設計

本節では，ISPL の設計について述べる．ISPL では，サービスの状態の因果関係を定義することで，サービスの連携を定義する．状態の因果関係を定義するために，サービスの状態の定義及び状態の因果定義の 2 つを行う．

3.2.1　サービスの定義

状態の定義

ISPL は，サービスをサービスが取り得る状態の集合として定義する．サービスの状態とは，例えば，ユーザが観測可能な状態を指す．状態とは，サービスが不変条件を保つ間の状況を表し，他のサービスとの関係に依存せずに決定する状態である．不変条件として，静的な状況を表現してもよいし，動的な状況を表現してもよい．静的な状況とは,「ＤＶＤプレーヤの停止状態」などのように，ユーザからの入力や遠隔からの制御を待つような状況である．また，動的な状況とは，なんからの活動を実行をしているサービス，例えば,「ＤＶＤプレーヤの再生中状態」のような状況である．サービスはこのような状態の集合として定義される．

ユーザが観測可能な状態とは，ＤＶＤプレーヤやエアコンなどの情報家電の場合は,「ＤＶＤの再生状態／停止状態」や「エアコンの停止状態／稼動状態」などのように,内部変数が直接状態として表現される．また，0.0〜1.0 の間で 64bit 表現される温度センサなどのように，ユーザから見て内部状態が非常に細かい場合は，温度センサの値をサービスの状態として表現せず，図 3.2 に示すように「かなり寒い」「やや寒い」「ちょうどよい」「やや暑い」「かなり暑い」を表現するメンバシップ関数を用いて，センサの値を５つの状態に変換するサービスを温度センササービスとして容易することによって，ユーザが直接観測可能な状態として，温度センサを扱うことができる．

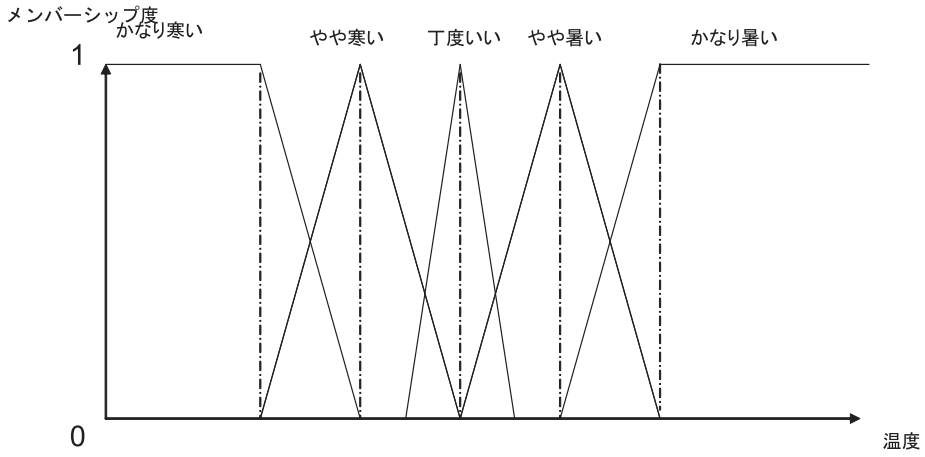

図 3.2: メンバーシップ関数による状態への変換

状態の抽象化

前項では，サービスの状態について述べた．サービスの状態の例として，次のような空間の状態が挙げられる．部屋Ａには複数人がいる，かつ，プロジェクトがＯｎの状態にある場合「部屋Ａは会議中という状態」や部屋Ａには人がいる，かつ，部屋Ａ

の扉がしまっている，かつ，ＤＶＤが再生されている，かつ，電灯が消灯している場合,「部屋Ａはシアターとして使用中である状態」といった部屋を状態と表現したい場合は，各サービスの状態をその都度定義すると，定義が煩雑であり，理解も困難となる．この場合，状態の集合をさらに抽象化された状態として定義することで，一度抽象状態を定義てしまえば，より簡便な表現を用いて，連携を定義できる．

3.2.2 因果関係の定義

因果関係の定義とは，上記のサービスの状態間の遷移を決定する．これによってサービスの連動を定義する．因果関係は，トリガ状態及び遷移先状態により構成される．以下にそれぞれについて述べる．

トリガ

トリガは，状態を遷移させる条件を指定する．条件として指定できるものは，時刻 t におけるサービスの状態である．

遷移先状態

遷移先状態は，トリガに指定される状態が成立した時に，それと連動して成立するべきサービスの状態を定義する．

3.3 ISPL の構文

ISPL は，xXtensible Markup Language(XML)[13] による構造化文書として記述される．本節では，XML にる実際の ISPL 記述について述べる．

3.3.1 ISPL の宣言

ISPL によって，状態や因果関係の定義を行う際は，図 3.3 のように<ISPL>タグを記述する．

```
<?xml version="1.0" ?>
<ISPL></ISPL>
```

図 3.3: ISPL 宣言

3.3.2 サービスの定義

本項では，ISPL におけるサービスの定義について述べる．

サービスの名前の定義

サービスの名前の定義は，図 3.4 に示すように,<service>タグ及び name 属性によって定義する．

```
<ISPL>
  <service name="helloService">
  </service>
</ISPL>
```

図 3.4: サービス名定義

サービスの状態の定義

サービスの状態は, 図 3.5 のように<state>タグを用いる．状態には，名前が付けられ，name 属性で指定する．

```
<ISPL>
  <service name="helloService">
   <state name="stateName"/>
  </service>
</ISPL>
```

図 3.5: サービス名定義

抽象状態の定義

抽象状態の定義は，<abstract_state>タグを用いる．<abstract_state>タグには，抽象状態を成立させるためのサービスの状態を指定する．この状態の指定には，指定された状態が同時になりっている，もしくは，指定された状態のうちどれかが成り立っている場合に，抽象状態が成立するかのかを，guanxi 属性を用いて図 3.6 に示すように指定する．図 3.6 におけるサービス RoomA では，指定した状態が両方成立していれば，

stateName1 の状態が成り立ち，指定した状態がどちらか成立していれば，stateName2 の状態が成り立つ．

```
<service name="RoomA">
  <abstract_state guanxi="coincidentally" name="stateName1">
      <service name="dvd" state="playing">
      <service name="lightA" state="on">
  </abstract_state>
  <abstract_state guanxi="eitherway" name="stateName2">
      <service name="dvd" state="playing">
      <service name="lightA" state="on">
  </abstract_state>
</service>
```

図 3.6: 抽象状態

3.3.3 因果関係によるサービス連動の定義

本項では，ISPL におけるサービス連携の定義について述べる．連携の定義は，図 3.7 のように<behavior>タグを用いる．属性として，連携を識別できる名前を記述する．

```
<ISPL>
  <behavior name="name">
  </behavior>
</ISPL>
```

図 3.7: 連携定義

連携は，状態の因果関係の集合として定義される．因果関係の定義は，図 3.8 のように<rule>タグを用いる．

因果関係は，原因となる状態及び遷移先状態からなる．

原因状態

原因状態は，図 3.9 のように<trigger>タグを用いる．トリガには，原因となる状態の集合を指定する．状態の指定には，図 3.10 のように<service>タグを用いる．

```
<ISPL>
  <behavior name="name">
    <rule></rule>
  </behavior>
</ISPL>
```

図 3.8: 因果関係

```
<ISPL>
  <behavior name="name">
    <rule>
        <trigger>
        </trigger>
    </rule>
  </behavior>
</ISPL>
```

図 3.9: 因果関係

また，同時に成立する複数の状態が原因状態となる場合は，図 3.11 のように，guanxi 属性によって，coincidentally を指定し，

指定する状態のどれかが原因状態となる場合は，図 3.12 のように，guanxi 属性によって，eitherway を指定する．

結果状態

結果となる状態の定義には，図 3.13 のように`<result>`タグを用い，本タグ内に，`<service>`タグを用いて，結果となる状態を指定する．

3.4 本論文のまとめ

本章では，サービス連携アプリケーションを定義する連携定義言語の意味，統語について述べた．次章では，本言語を用いて実際にサービスを制御するためのモデル駆動型アーキテクチャについて述べる．

```
<ISPL>
  <behavior name="name">
    <rule>
        <trigger>
         <service name="dvd" state="playing">
        </trigger>
    </rule>
  </behavior>
</ISPL>
```

図 3.10: 原因状態 1

```
<ISPL>
  <behavior name="name">
    <rule>
        <trigger guanxi="coincidentally">
         <service name="dvd1" state="playing">
         <service name="dvd2" state="playing">
        </trigger>
    </rule>
  </behavior>
</ISPL>
```

図 3.11: 原因状態 2

```
<ISPL>
  <behavior name="name">
    <rule>
        <trigger guanxi="eitherway">
         <service name="dvd1" state="playing">
         <service name="dvd2" state="playing">
        </trigger>
    </rule>
  </behavior>
</ISPL>
```

図 3.12: 原因状態 3

```
<ISPL>
  <behavior name="name">
    <rule>
        <result>
         <service name="dvd1" state="playing">
         <service name="dvd2" state="playing">
        </result>
    </rule>
  </behavior>
</ISPL>
```

図 3.13: 結果状態

第 4 章

サービス支援を実現するモデル駆動型アーキテクチャの設計

本章では，多様なユーザが多様なユーザインタフェースを用いてサービス連携を定義するサービス支援システムの設計を述べる．本実行環境は，ISPL によるサービス連携定義を元に，実際にサービス制御を行うであるモデル駆動型アーキテクチャである．

4.1 設計方針

本節では，モデル駆動型アーキテクチャ[9] の設計方針と機能要件について述べる．

4.1.1 モデル駆動アーキテクチャ

モデル駆動型アーキテクチャとは，アプリケーション開発を行う際に，問題領域のモデルを表現する言語によりアプリケーションの機能や動作を定義し，モデル情報を元に，実際のアプリケーションのソースコード等を自動生成するアーキテクチャである．本研究におけるホームネットワークにおける多様なユーザが多様なユーザインタフェースを用いてサービス連携を定義するサービス支援システムは，ISPL によるサービス連携定義情報を元に，実際の情報家電等のサービスの制御を行うモデル駆動型アーキテクチャである．ISPL 連携定義言語によりサービス連携を表現することで，様々なユーザインタフェースから一貫してサービス連携の定義・確認ができる．

4.1.2 低いシステム管理コスト

ホームネットワークにおいて，専門的な管理者を想定したシステムは，金銭的コストがかかるなど，現実的ではない．名前サーバ及び連携定義評価サーバを単一のホストに集中させると，サーバのアドレスの管理や，これらサーバの継続的な動作の保障が必要となる．本システムでは，サービスの名前管理は，各サービスが稼動するランタイム内のネーミング機能が個別に管理し，ランタイム内のネーミング機能間で同期を取る．さらに，各サービス自体がサービスが稼動するランタイム内で連携定義や状態定義を保持し，連携定義評価エンジン自体を持つことで連携定義評価サーバを介さずサービス自信が図 4.3 のように通信することで制御を実現する．また，サービスの名前管理も各サービスが稼動するランタイムが管理する．これにより，ホームネットワークのような管理者の想定が困難な環境でも，サーバのアドレスの設定が不要となり，さらに，システムに必要な名前サーバ及び連携定義評価サーバを継続的に動作させる保障がなくなる．居住者は各サービスの動作のみを保障することで，ホームネットワークの運用コストが軽減される．

4.2 機能要件

4.2.1 通信機能

連携定義言語 ISPL に基づいて，サービスを制御を行うため，制御側プログラムは，非制御側のサービスを制御する際に，非制御サービスが提供するコードやプロトコル情報などを必要とすることなく，サービス制御を行える通信機能が必要となる．

4.2.2 サービス動的配置機能

ホームネットワークでは，家電の電源が頻繁に切れたり，機器が移動されたりする．これにより，サービスが常に稼動している状況は想定にしくい．従って，起動されたサービスが他のサービスを再起動することなしに，他のサービスが発見できる必要がある．

4.2.3 連携定義評価機能

連携定義言語 ISPL を解析し，サービスの状態を元にサービスの制御を行う機能が必要となる．

4.2.4 サービスフレームワーク

連携定義評価機能によって，サービスの状態を元にサービスの制御を行う．このため，サービスは，各々の状態変化を常にネットワークに通知する．また，状態遷移要求を受け取ると状態を実際にサービスの内部状態に反映させる．このため，サービス支援システムにおけるサービスは状態の通知機能及び状態遷移要求への応答機能を持つ．これらをフレームワーク化することで，ホームネットワークにおけるサービス開発者の負担を軽減する．

4.3 設計

本節では，第 4.1 節で述べた指針に従って，モデル駆動型アーキテクチャの設計を示す．

4.3.1 アーキテクチャ

本システムにおけるモデル駆動アーキテクチャの設計は図 4.1 に示されるように 3 層に区分される．第 1 の通信機能を持つ通信ミドルウェアによって，センサや，情報家電やソフトウェアコンポーネントが相互に発見し通信する層である．第 2 の層は，状態の因果関係をもとにしたサービス制御層である．この層では，通信機能を使用し，状態の変更通知及び状態変更要求の状態メッセージをサービス間で送受信し，状態メッセージに応じて実際のサービスの内部状態を変更することによりサービス連動を実現する．第 3 の層は，ユーザによる状態の定義・連動定義を行う．

4.4 Container アーキテクチャ

コンテナアーキテクチャは，通信機能及びサービス動的配置機能をサポートする．本節では，Container アーキテクチャにおける通信機能及びサービスの動的配置機能について説明する．

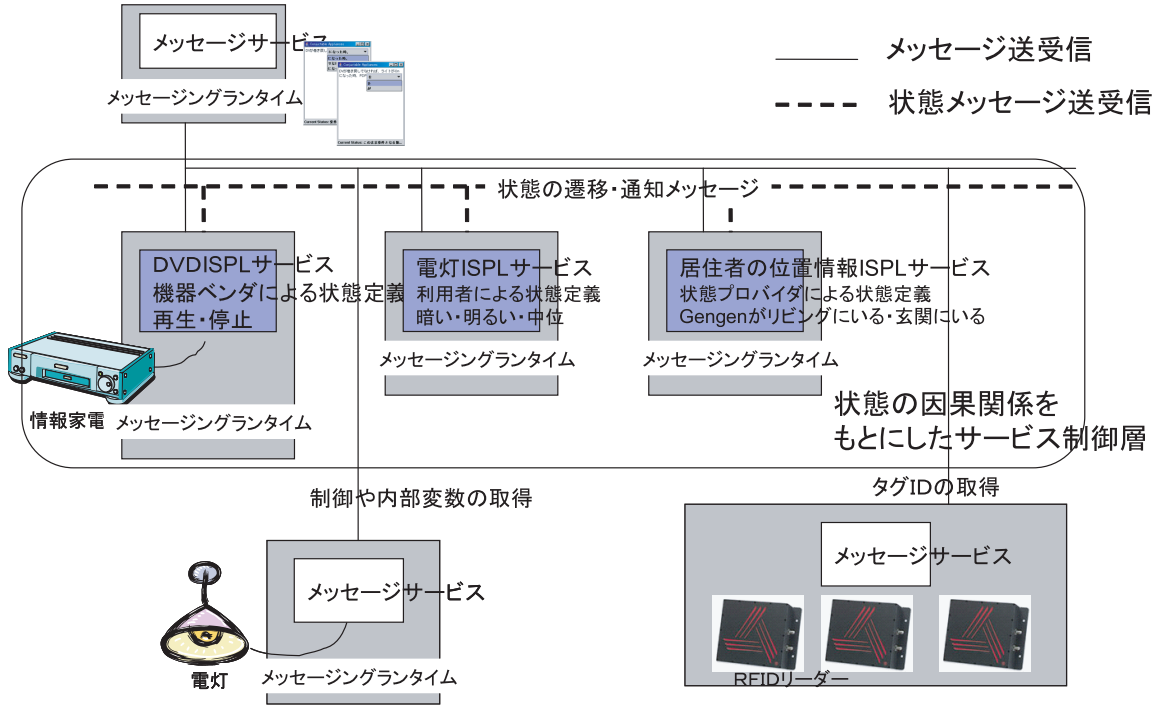

図 4.1: アーキテクチャ

4.4.1 Container

Container は，図次項で述べる通信機能を実現する上で必要な下位の機能を表現する Facility やライブラリ及びメッセージ通信の主体である Component をコンテナが許す動作以外は実行できないよう管理する．これにより，悪意のあるコードにより，システムが不正な動作をするリスクを軽減する．低レベルなメッセージ通信を行う Channel を持つ．さらに，コンテナは Facility 及び Component の起動及び終了のライフサイクルの管理を行う．

4.4.2 Component

Component は，コンテナからスレッドが割り当てられ，自律的に動作するアクティブオブジェクト及びメッセージ受信に応じて動作するパッシブオブジェクトとして実装される．例えば，センサ情報を定期的に送信する場合などは，アクティブオブジェクトとして Component となる．また，情報家電のようにメッセージに応じた動作しか行わない場合は，パッシブオブジェクトとなる．

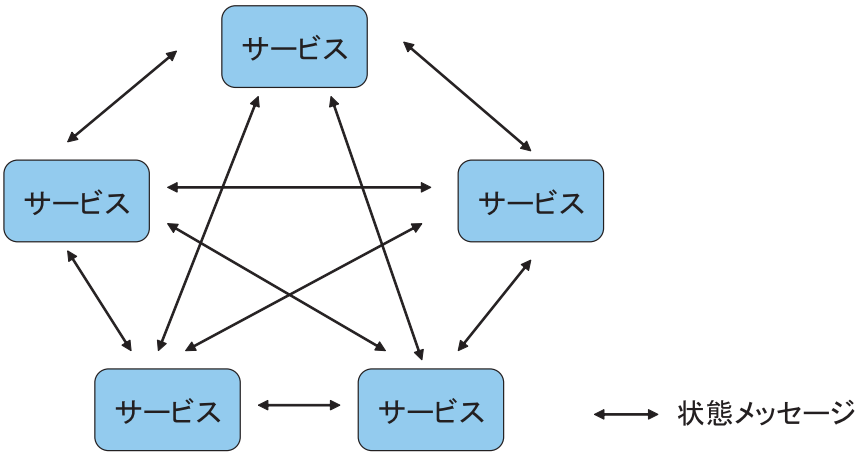

図 4.2: 状態メッセージ

4.4.3 Channel

Channel は，Facility から利用され，ネットワーク上の他のコンテナの Facility とメッセージ送受信を行う際に使用される．Channel へ送受信されるメッセージは，バイト列として表現され，バイト列の順序は保障するが，メッセージの到着順は保障しない．

4.4.4 Facility

Facility は，コンテナのモジュール化された機能として Component に提供される．Facility としてモジュール化され提供される．これにより，通信機能部における機能の拡張や変更が容易である．Facility は，Component と同様，コンテナからスレッドが割り当てられ，自律的に動作するアクティブオブジェクト及びメッセージ受信に応じて動作するパッシブオブジェクトとして実装される．Facility は，メッセージ通信を行うための MessagingChannelFacility と Component などの名前を管理する NamingFacility がある．

MessagingChannelFacility

メッセージ送受信を行うための機能を持つ．MessagingChannelFacility は，メッセージを Channel へ送受信を行う．Component がメッセージ送受信を行う際は，MessagingChannelFacility を介して，メッセージの送受信を行う．送受信されるメッセージは，オブジェクトであり，送信時にプログラム言語非依存の形式に変換され，受信時にオブジェクトへと復元される．

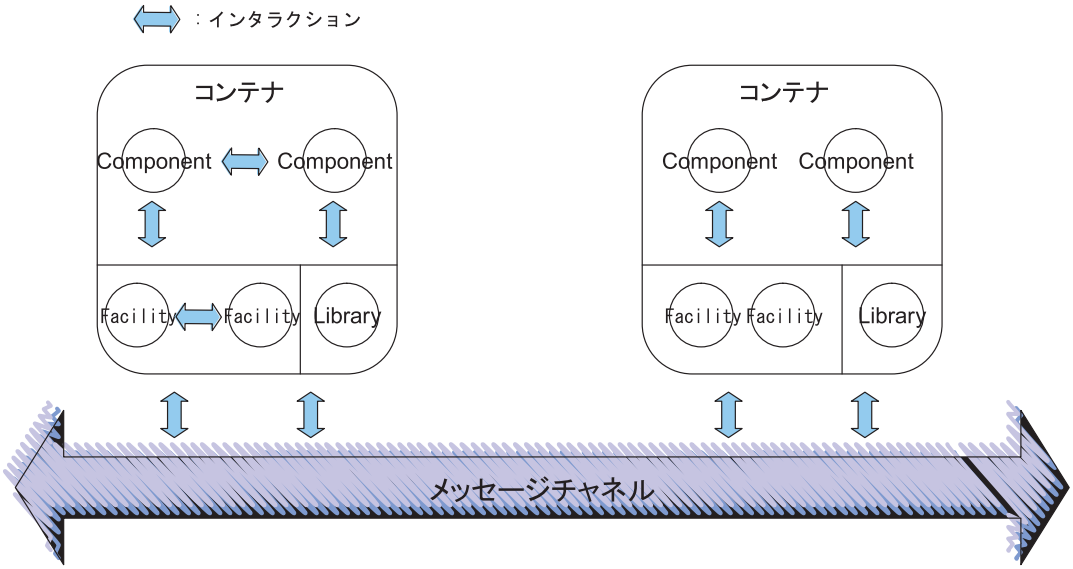

図 4.3: コンテナ

NamingFacility

Component の名前を管理する機能を持つ．NamingFacility は，各コンテナ上で稼動し，稼動した Component の名前を管理する．これによって，サービス間の通信をアドレスではなく，任意の名前によって行うことができる．この機能を実現するために，メッセージ送受信を通じて，同一ネットワーク上のコンテナ上で稼動する NamingFacility に登録されている Component に関する情報の同期を取る機能を持つ．これによって，新たに起動された Component が他のコンテナや Component を再起動することなしに，他の Component が発見できる．

4.4.5 Library

コンテナ内にクラスライブラリなどが配備され，コンテナ外のライブラリを Component 及び Facility は利用できない．これにより，悪意のあるコードにより，システムが不正な動作をするリスクを軽減する．

4.5 連携定義評価機能の設計

連携定義評価機能は，Facility として提供される BehaviorInterpreterFacility とサービスフレームワークにおける状態変更通知機能及び状態変更要求受信機能によって実現される．本節では，サービスフレームワークと BehaviorInterpreterFacility について述べる．

4.5.1 サービスフレームワーク

サービスフレームワークは，ISPL により連携を定義できるサービスに共通して持つ機能を Component を拡張し，フレームワーク化したものである．従って，サービスフレームワークは第 4.4 節で述べたコンテナ内における Component が受ける制約を完全に受ける．

また，共通する機能を Component を基に抽象化し，AbstractStateService としてフレームワーク化することにより，サービス開発者の負担を軽減する．ISPL によって連動可能なサービスを提供したいベンダは，本サービスフレームワークに基づいてサービスを実装する．

状態変更通知機能

状態変更通知機能とは，サービスの状態定義に応じた状態変更通知メッセージを Channel へ送信し，ホームネットワーク上のすべての BehaviorInterpreterFacility へ通知する．

状態変更要求受信機能

状態変更要求受信機能とは，ローカルホストの BehaviorInterreterFacility からくる状態変更要求メッセージを受信する機能を持つ．また，フレームワークに対するアプリケーションから状態変更要求メッセージハンドラの登録された登録を受け付ける．さらに，受信した状態変更要求メッセージに応じて登録されたハンドラを駆動する．

状態定義情報保持機能

状態定義情報保持機能とは，ISPL によって定義された状態定義ファイルを保持し，ユーザインタフェースなどからサービスの状態定義に関する問い合わせに答える．

4.5.2 BehaviorInterreterFacility

BehaviorInterreterFacility は，状態定義及び連携定義を保持し，サービスから受信した状態変化通知メッセージと，状態定義及び連携定義とを比較し，ルールに基づいてローカルホスト上のサービスへ状態変更要求を送信する．次に，BehaviorInterreterFacility を構成する各サブ機能について述べる．

StateChangeConsumer

StateChangeConsumer は，サービスの状態変化通知メッセージを監視し，状態変化通知メッセージを受信した際は StateMessageHeap へメッセージを格納し，その後に BehaviorEvaluator へメッセージの受信を通知する．

BehaviorParser

BehaviorParser は ISPL 言語で記述した連携定義ファイルの字句解析・構文解析・意味解析を行う． Behavior Parser は，意味解析終了時に，連携定義における因果関係の集合を生成し，BehaviorRuleRepository へ格納する．

BehaviorEvaluator

BehaviorEvaluator は，StateMessageHeap に格納される状態変更通知メッセージ及び BehaviorRuleRepository の因果関係を元に，サービスに対する状態変更要求を出す．

状態変更要求機能

BehaviorEvaluator から受けた状態変更要求の要求対象が同一ホスト上のサービスであれば，サービスへ状態変更要求を出す．

BehaviorRuleRepository

BehaviorParser が生成した連携定義を保持する．

AbtractStateRepository

抽象状態の定義を保持する．抽象状態の定義は，ホームネットワーク上のすべての BehaviorInterreterFacility は，抽象状態に定義に関する情報の同期を取る機能を持つ．

StateMessageHeap

連携定義にトリガに複数のサービスの状態の同時成立が定義されている場合，複数の状態変更通知メッセージが時間的に遅れて到着した際も，正しく BehaviorEvaluator が連携定義を評価できるように，状態変更通知メッセージは，一定時刻保持される．

4.6 本章のまとめ

本章では，モデル駆動型アークテクチャの設計指針とその設計について述べた．次章では，多様なユーザが多様なユーザインタフェースを用いてサービス連携を定義するサービス支援システムのプロトタイプ実装について，その設計通信機能，連携定義評価機能，サービスフレームワークについて述べる．また，実際に情報家電やセンサが配置された Smart Living Room ともに，本アーキテクチャ上に構築した Smart Living Room におけるサービスの実装例を見る．

第 5 章

実装

本章では，前章の設計に基づいたサービス支援システムのプロトタイプ実装について述べ，ISPL を元にしたユーザインタフェースの例，情報家電やセンサが配置され，サービス連携が実現されたホームネットワーク Smart Living Room について述べるとともに，Smart Living Room 上のサービスの実装例について見る．

表 5.1: 実装環境

項目	環境
CPU	Intel Pentium M プロセッサ 765
主記憶	1GB
OS	WindowXP Professional
開発言語等	Eclipse/Java 言語/J2SE 1.5

5.1 プロトタイプ実装

本節では，サービス支援システムのプロトタイプ実装について述べる．本プロトタイプ実装を，表 5.1 に示す．Java 言語 [3] は，プラットフォーム非依存であり，また，開発効率も高いため，実装に Java 言語を用いた．

5.2 通信機能の実装

本節では，Java 言語を用いたメッセージ通信を行う通信機能の実装について述べる．

5.2.1 Node

Node は，ネットワークにおけるホストを表現し，main メソッドを持ち，通信機能を実現するために必要な構成情報をファイルシステムから読み込み保持する．Node が読み込む構成情報を次に示す．

5.2.2 Node ディプロイメントディスクリプタ

Node は，XML で記述されたメッセージ送受信のためのチャネルに関する構成情報として，ChannelDescirptor を読み込み，Channel を生成する．また，Container 内で動作する Facility に関する構成情報として図 5.2 に示す FacilityDescriptor を読み込み，Facility を生成する．ここでは，メッセージチャネルが使用するポート番号を名前を指定して，チャネルを初期化する．また，図 5.1 に示すように Component を含むクラスファイルの配置場所が記述された NodeDescriptor を読み込む．

Node は，XML で記述された図 5.3 に示す構成情報を読み込み，Facility の生成を行う．`<facility>`タグによって Facility の起動クラスを指定し，`<context>`タグによって，facility の実行時パラメータを渡す．

```
<component-descriptor-path>
./jp/ac/keio/sfc/ht/umiddleware/apps
</component-descriptor-path>
<facility-component-descriptor-path>
./jp/ac/keio/sfc/ht/umiddleware/facilities
</facility-component-descriptor-path>
```

図 5.1: NodeDescriptor

```
<channel-set>
    <channel name="hoge-event" port="21111"/>
</channel-set>
```

図 5.2: ChannelDescriptor

5.2.3 Container

Container は，Component 及び Facility の動作環境であり，両者は，Container 内で動作する．Container は，NodeDescriptor から読み込んだ情報をもとに，Facility 及び Component を生成する．生成に際しては，図 5.4 のようにそれぞれの個別に Thread 及びクラスローダを割り当て，稼動させる．

5.2.4 Container インタフェース

Container の主なインタフェースを表 5.2 に示す．このインタフェースを用いて，Component は Facility へアクセスする．

表 5.2: Container インタフェース

public Facility getFacility(String className);
public Map getFacilityRepository();

```
<facility-set>
    <facility class="jp.ac.keio.sfc.
     ht.gengen.ispl.facility.activator.Activator">
    <context key="timingdeffile"
     value="jp/ac/keio/sfc/ht/utexture/conf/activation-timing.xml"/>
</facility>
    <facility class="jp.ac.keio.sfc.ht.gengen.ispl.test.Sample">
    <context key="timingdeffile"
    value="jp/ac/keio/sfc/ht/utexture/conf/activation-timing.xml"/>
    </facility>
</facility-set>
```

図 5.3: FacilityDescriptor

```
ThreadGroup tg = new ThreadGroup(comp.getId());
t = new ComponentKickerThread(tg, comp);
t.setContextClassLoader(comp.getClassLoader());
```

図 5.4: Thread の割り当て

5.2.5 ClassLoader

Container において，Component 及び Facility のクラスファイルが圧縮されたアーカイブとして配備されてた場合にも生成できるように JavaVM のクラスローダではなく jar ファイルを読み込むクラスローダを用いる．

5.2.6 Component

コンポーネントは，表 5.3 に示すように 3 つの抽象メソッドを実装することで定義する．

public abstract void start() メソッドは，アプリケーションが起動される際に行われる処理を定義する．public abstaract void start() メソッドは，この中でコンポーネントのメインループを定義する．public abstract void stop() メソッドは Component が，停止された場合の処理を定義する．

表 5.3: Component インタフェース

```
public abstract void init();
public abstaract void start();
public abstract void stop();
```

Component のディプロイメントディスクリプタ

Component は，Node ディスクリプタで指定したファイルシステムにおかれる Component ディプロイメントディスクリプタの情報をもとに生成される．Component ディプロイメントディスクリプタでは，図 5.5 に示すように表示名，NamingFacility に登録される際の名前クラス名，実行時に与えるパラメータを定義する．ID は UUID[10] 用いており初回の Component 起動時に自動的に付与される．

```
<component-descriptor>
<display-name>sample component</display-name>
<name>sample</name>
<id>195137fc-fdfa-4e7f-baed-b316424e9fb3</id>
<class-name>jp.ac.keio.sfc.ht.t41.test.Sample</class-name>
<doc-base>C:\Documents and Settings\gengen
\My Documents\workspace\loft-in-the-loft
\jp\ac\keio\sfc\ht\umiddleware\apps\sample.jar</doc-base>
<parameter><name>a</name><value>n</value></parameter>
</component-descriptor>
```

図 5.5: Component ディプロイメントディスクリプタ

ActiveComponent の例

Thread が割り当てられたアクティブコンポーネントとして定義する場合は，図 5.6 に示すように，start メソッドをオーバーライドし，ループを記述する．

PassiveComponent の例

Thread が割り当てられずメッセージに応じて動作を行うパッシブ Component として定義する場合は，図 5.7 に示すように，Sink インタフェースを実装し，Component ク

```
public class MyComponent extends Component{
    private volatile boolean running = true;
    public void init() {
//前処理
    }
    public void start() {
        running = true;
        while(running){
            try{
                doSomething();
TestEvent t = new TestEvent();
t.setName("gengen");
super.send(t);
Thread.sleep(3000);
           }catch(InterruptedException e){
           }
}
    }
    public void stop() {
        running = false;
    }
}
```

図 5.6: アクティブ Component

ラスのメソッド addMessagingListener を用いて，メッセージチャネルに接続する．メッセージを受信すると Sink インタフェースの received(Message msg) メソッドが非同期に呼び出される．

5.2.7 MessagingChannelFacility

MessagingChannelFacility は，Component がメッセージ通信を行う際に，Component から利用される．Component から MessagingChannelFacility へのアクセス方法は，Container インタフェース getFacility メソッドを利用するか，もしくは，Component クラスのメソッド send(Message msg) を利用する．

```
public class MyComponent extends Component   implements Sink {
    public void init() {
//前処理
super.addMessagingListener(this);
    }
    public void start() {

    }
    public void received(Message msg){
        if(evt instanceof TestEvent){
            System.out.print("TestEvent Received: ");
            TestEvent hoge = (TestEvent)evt;

System.out.println(System.currentTimeMillis()-hoge.getTime());
    }
    }

    public void stop() {
    }
}
```

図 5.7: パッシブ Component

Message

MessagingChannelFacility を利用して Message クラスを継承したオブジェクトの送受信を行う．表 5.4 にメッセージクラスのインタフェースを示す．送受信できるメッセージは，Message クラスを継承し，継承先クラスの各フィールドは，public アクセス修飾子による getter/setter を付与する．Getter/setter の命名規則は，getName/setName などの命名規則に従う．

Externalizer

MessagingChannelFacility は，Message 型オブジェクトの送受信を行う．ネットワークに送信される際は，メッセージを XML を用いた形式に変換し，受信時にオブジェクトへ戻す．Externalizer は，Message 型オブジェクトのプログラミング言語によらない XML を用いた表現に変換する．この際の XML の形式を図 5.8 に示す．継承先クラス

表 5.4: Message クラス

```
public String getComponentId();
public String getComponentType();
public long getTime() ;
public String getNodeId();
```

の各フィールドは，<member>タグで表現され，属性として，フィールド名，型，値が指定される．String 型オブジェクトの配列については，<element>タグで要素を表す．

```
<externalization class="jp.ac.keio.sfc.ht.gengen.MyTag"
time="123" nodeid="sdf">
  <member name="sdf" type="int" value="11"/>
  <member name="sd" type="int[]" value="01h02h"/>
  <member name="sdf" type="string[]">
    <element>hosdf1</element>
    <element>hosdf2</element>
  </member>
</externalization>
```

図 5.8: 外部化の形式

5.3 連携定義評価機能の実装

本節では，Java 言語を用いた連携定義評価機能について述べる．連携定義評価機能は，第 5.2 で述べた Facility であり，BehaviorInterpreterFacility として実装されている．BehaviorInterpreterFacility を構成する StateChangeConsumer オブジェクト，BehaviorEvaluator オブジェクト，NamingFacility オブジェクト,BehaviorRuleRepository オブジェクトがサービス連携を行う際のシーケンス図を図 5.9 に示す．状態の変更がメッセージによりネットワーク上のすべての StateChangeConsumer へ伝播し，StateChangeConsumer は，BehaviorEvaluator へ状態変更メッセージを通知する．BehaviorEvaluator は，BehaviorRuleRepository へ連携定義を問い合わせ，状態変更メッセージが連携定義におけるトリガに合致していれば，NamingFacility へ状態変更要求を出すべきサービスの場所を問い合わせる．サービスが同一ホスト上にあれば，サービス

表 5.5: AbstractStateService クラス

protected void notifyStateChanged(StateChangeMessage msg);throw IllegalStateException
public abstract void requestChangeStateMessageReceived(StateChangeRequestMessage msg);
public ServiceProfile getProfile();

へ状態変更要求を送信し処理が完了する．また，サービスが異なるホスト上にあれば，サービスへ状態変更要求を送信することなく処理は完了する．

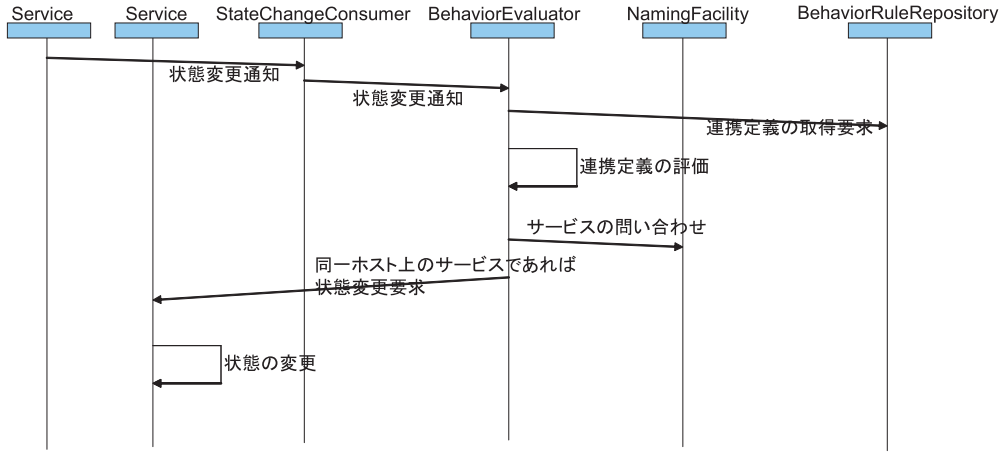

図 5.9: 連携定義評価機能シーケンス図

図ネットワーク上に分散する BehaviorInterpreterFacility が図 5.9 に示す動作を行うことで，サーバを介さずサービス連携定義の評価を行える．

5.4 サービスフレームワークの実装

サービスフレームワークは，ISPL により連携を定義できるサービスに共通して持つ機能を Component クラスを継承し，AbstractStateService クラスとしてフレームワーク化する．フレームワークでは，状態変更通知機能，状態変更要求受信機能，状態定義情報保持機能を提供する．これらの機能群と第 5.3 節で述べた連携定義評価機能の関係を図

表 5.5 に AbstractStateService の持つメソッドを示す．

サービス開発者は，サービスの実装において，サービスの内部状態の変化した際に，notifyStateChanged を呼ぶことで，ネットワーク上のすべての BehaviroInterpreterFacility へ通知される．また，BehaviroInterpreterFacility から状態変更要求を受けた際に，

実際のサービスの内部状態を変化させる処理を requestChangeStateMessageReceived メソッドに実装する．StateMessage は，状態変更を通知するためのメッセージであり，StateChangeRequestMessage は，状態変更要求のメッセージある．それぞれ，Message クラスを継承する．

5.5 ISPL をもとにしたユーザインタフェースの例

図 5.10 に示すようなサービス連携を「選択」と「決定」という簡易なインタラクションで定義する ISPL を元にしたユーザインタフェースのプロトタイプを実装した．サービスの名前及び状態を順次選択し，条件や結果を表現する日本語を選択していくことで ISPL を生成する．このようなインタフェースは，操作性には，すぐれないものの，自然言語で直感的に連携を定義していくためより幅広い利用者にとって，学習コストが低いと予測される．

図 5.10: ISPL のユーザインタフェースの例

5.6 アプリケーション例-Smart Living Room-

本節では，情報家電やセンサが配備された Smart Living Room を紹介し，Smart Living Room におけるサービス例及び連携アプリケーション例を紹介する．

5.6.1 Smart Living Room

Smart Living Room の概観を図 5.11 に示す．Smart Living Room は，PILA[12] と呼ばれる部材を組み合わせ，屋内の中にリビングルームを構築した．

図 5.11: Smart Living Room

Smart Living Room の構造

Smart Living Room では，センサがフレームに配置され，家電がネットワークから制御可能となっている．家電やセンサを管理する計算機を配置するために図 5.13 で示すように，天井と床下の 2 重構造となっている．

Smart Living Room のセンサ

5.13 に示すように，Smart Living Room には，Mote センサ [14] が配置されており，温度・照度・湿度を取得できる．

Smart Living Room のアクチュエータ

図 5.14 に示すように，Smart Living Room では，ネットワークに接続された家電や電灯，天井のスクリーンなどがアクチュエータとして配備されている．

サービス例

Smart Living Room における状態に基づくサービスとして次のものを実装した．

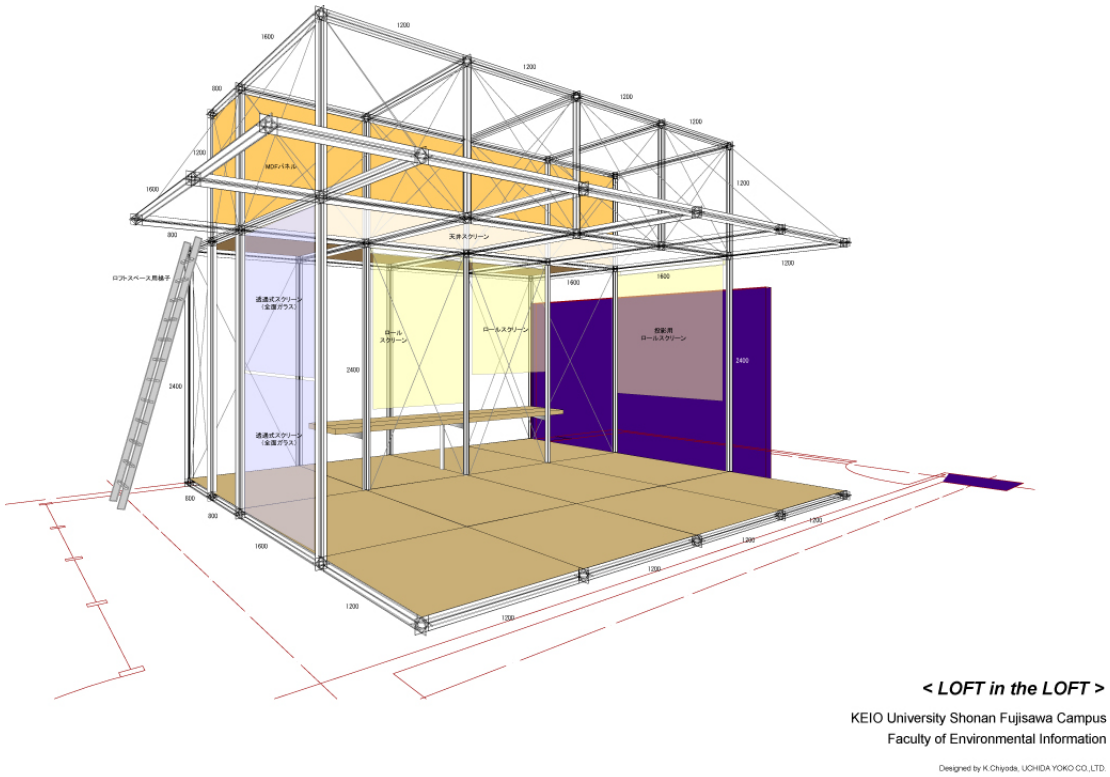

図 5.12: Smart Living Room の構造

電話サービス

このサービスは，H.323[4] プロトコルにおけるセッションを監視することで，電話の電話中，待機中２つの状態を取得する．

ドアサービス

このサービスは，Serial インタフェースをドアコントローラに実装し，PC からドアをシリアル通信で制御によりドアの開状態，閉状態を検知している．

ＤＶＤサービス

このサービスは，シリアル制御赤外線リモコンを利用し，ＤＶＤプレーヤの再生，停止の制御を行う．

TV サービス

このサービスは，シリアル制御赤外線リモコンの利用し，１から１２のチャンネルを制御する．

図 5.13: Smart Living Room のセンサ

電灯サービス

このサービスは，MIDI による調光機を利用し，消灯，点灯の制御をする．

アプリケーション例

Smart Living Room における連携アプリケーションの例を挙げ，ISPL による例を示す．部屋でユーザが映画を見ている．電灯は，消灯されている．ＴＶ電話がかかってきた．連携アプリケーションはＴＶ電話の呼び出しに応じて，ＴＶの入力切替を行い，音量を下げ，電灯を点灯させた．電話の通話が終了するとＴＶの入力を元に戻し，音量を戻して，電灯を消灯させた．

このシナリオでは，登場するサービスは，ＴＶサービス，電話サービス，電灯サービス，アンプサービスであり，それぞれ次のような状態を持つ．

- TV サービス
  ＤＶＤ入力状態
  ＴＶ入力状態
  ＰＣ入力状態
  ＴＶ電話入力状態

- 電話サービス
  コール状態
  切断状態

図 5.14: Smart Living Room のアクチュエータ

- 電灯サービス
  点灯状態
  消灯状態

- アンプサービス
  かなり大音量状態
  やや大音量状態
  普通の音量状態
  やや小音量状態
  かなり小音量状態
  無音

これらサービスの ISPL による状態定義は次のようになる．

5.7 本章のまとめ

本章では，システムの Java 言語による実装について述べ，さらに，Smart Living Room における ISPL アプリケーション例を紹介した．次節では，本システムの基本性能測定について述べる．

```
<ISPL>
<service name="TV">
<state name=""input1>
<display-name>ＤＶＤ入力状態</display-name>
</state>
<state name=""input2>
<display-name>TV 入力状態</display-name>
</state>
<state name=""input3>
<display-name>PC 入力状態</display-name>
</state>
<state name=""input4>
<display-name>TV 電話</display-name>
</state>
</service>
</ISPL>
```

図 5.15: TV サービス

```
<ISPL>
<service name="IP-Phone">
<state name="calling">
<display-name>電話がかかってきた</display-name>
</state>
<state name="disconnect">
<display-name>受話器を置いている</display-name>
</state>
</service>
</ISPL>
```

図 5.16: 電話サービス

```
<ISPL>
<service name="light">
<state name="on">
<display-name>点灯</display-name>
</state>
<state name="off">
<display-name>消灯</display-name>
</state>
</service>
</ISPL>
```

図 5.17: 電灯サービス

```
<ISPL>
<service name="amp">
<state name="1">
<display-name>かなり大音量状態</display-name>
</state>
<state name="2">
<display-name>やや大音量状態</display-name>
</state>
    <state name="3">
<display-name>普通の音量状態</display-name>
</state>
<state name="4">
<display-name>やや小音量状態</display-name>
</state>
<state name="5">
<display-name>かなり小音量状態</display-name>
</state>
<state name="6">
<display-name>無音</display-name>
</state>
</service>
</ISPL>
```

図 5.18: アンプサービス

```
<ISPL>
  <behavior name="tel">
   <display-name>楽電</diplay-name>
    <rule>
        <trigger>
         <service name="IP-Phone" state="calling"/>
        </trigger>
        <result>
         <service name="light" state="on"/>
         <service name="tv" state="input4"/>
         <service name="amp" state="6"/>
        </result>
    </rule>
    <rule>
        <trigger>
         <service name="IP-Phone" state="disconnect"/>
        </trigger>
        <result>
         <service name="light" state="off"/>
         <service name="tv" state="input1"/>
         <service name="amp" state="4"/>
        </result>
    </rule>
  </behavior>
</ISPL>
```

図 5.19: 連携定義

第 6 章

関連研究との比較

本章では，他の連携定義言語や連携定義を実現するシステムに関する議論を行う．連携定義言語として，CLAM や PICCOLA を上げ，また，ユーザインタフェースとして FieldMouse,MediaCube を挙げる．

6.1 マルチモデルサポート言語との比較

CLAM[5] は，図 6.1 のように手続き型言語によって CORBA オブジェクトの連携を定義する．

```
light_handle = SETUP("LIGHT");//名前サーバに問い合わせる文字列を引数にとる
dvd_handle = SEUP("DVDPlayer");//名前サーバに問い合わせる文字列を引数にとる
WHILE(TRUE){//制御構造
     dvd_state_handle = dvd_handle.INVOKE("GET_STATE");//CORBA のインタフェース呼び出し
    state = dvd_handle.EXTRACT();//結果の取得
    IF(state == "PLAYING"){
      light_handle.INVOKE("OFF");
    }
}
```

図 6.1: CLAM による連携定義

どうようの言語に PICCOLA[1] が挙げられる．これらの特徴について述べる．

特徴

CLAM は手続き指向型言語であり，従って，表現できるモデルが多様である．離散時間モデルも表現が可能であり，離散事象モデルも表現が可能であり，このような言語を本研究では，マルチモデルサポート言語と呼ぶ．マルチモデル言語の利点は，利用者が言語の意味及び統語を習得すれば，様々なな分野で言語を用いて問題を解決が可能な点が特徴である．

マルチモデルサポート言語の 2 点目の特徴は，問題領域をどのようなモデルで捉えるか利用者によって，離散時間モデルであったり，離散事象モデルであったりと様々なモデルを許容する点である．

6.2 考察

手続き指向型言語により CORBA オブジェクトの連携を定義する CLAM や PICCOLA などの連携定義言語を見た．これは，言語が多様なモデルを表現可能なため，多様なユーザインタフェースを想定した連動定義言語には適さない．また，FieldMouse[17]

や MediaCube[2] などの実世界指向インタフェースによる連動定義手法が提案されている．これは，単一の連携定義言語を介在させない．従って，連動定義を他のユーザインタフェースを用いて定義・確認を行うことが困難である．

従って，多様なユーザが想定されるホームネットワークでは，一貫した単一のモデルによる連携定義が必要である．ISPL は，ホームネットワークにおける連携モデルとして，原因ー結果の関係を採用し，連動を情報家電などのサービスとその状態及び時間や場所などのコンテクストの因果関係により定義する．ジャンピアジェによる認知発達モデルは，感覚ー運動期（誕生ー 2 才），前操作期（2-7 才），具体的操作期（7-12 才）形式的操作期（12 才〜）に分けれら，具体手的操作期を経た人間は，具体的な直接観測可能な対象に対してのみ，原因ー結果の関係を理解、論理的な分析能力を持つとされている．このことから，本研究で提案する連携定義モデルは，様々なホームネットワークユーザにとって理解しやすい連携モデルであると言える．

第 7 章

結論

7.1 今後の課題

本節では今後の課題について述べる．

記述インタフェースに関する課題

今回 ISPL を元にしたインタフェースとして，サービスとその状態を「選択」と「決定」することで連携を定義する ISPL を元にしたユーザインタフェースを実装した．

ユーザと計算機のインタラクションは，文字，アイコンなどの視覚を用いたもの，例示を用いたもの，音声を用いたもの実世界指向記述によるものと様々である．今後 ISPL を元にした，多様なインタフェースを設計・実装していくことで，本研究の主眼である多様なユーザを想定するサービス支援システムの実証を進める．

システムに関する課題

サービス支援システムはサービス連携定義言語 ISPL を元にしたモデル駆動型アーキテクチャの設計を行った．モデル駆動型アーキテクチャの仕様策定，普及の活動行う OMG[9] は，モデル駆動型アーキテクチャの利点はミドルウェアに依存しない，包括的，かつ組織化された，将来にわたる相互運用を実現するとしている．本研究では，通信ミドルウェアとして，独自のメッセージ通信による機構を設計・実装し，それを用いて ISPL の処理系を実現した．実際に，SOAP[7],CORBA[8] などのミドルウェア上で ISPL の処理系で実装することで，モデル駆動型アーキテクチャの特徴である「ミドルウェアに依存しない，包括的，かつ組織化された，将来にわたる相互運用」が可能であることを実証する．

7.2 本論文のまとめ

ホームネットワークユーザによる学習や理解が容易なモデルを表現する連携定義言語 ISPL を提案し，ISPL から実際のサービスの制御を行うモデル駆動アーキテクチャの設計について述べ，ISPL を基にしたユーザインタフェース例及び実世界に構築したホームネットワークとホームネットワーク上のサービスについて述べた．

既存の連携定義言語として，手続き指向型言語により CORBA オブジェクトの連動を定義する CLAM や PICCOLA などの連携定義言語が提案されている．これは，言語が多様なモデルを表現可能なため，多様なユーザインタフェースを想定した連動定義言語には適さない．また，FieldMouse などの実世界指向インタフェースによる連動定義手法が提案されている．これは，単一の連携定義言語を介在させない．従って，連動定義を他のユーザインタフェースを用いて定義・確認を行うことが困難である．

本論文で提案した連動定義言語は，サービス連携モデルとして原因ー結果の関係を採用し，連動を情報家電などのサービスとその状態及び時間や場所などのコンテクストの因果関係により定義する．ホームネットワークにおいて，このような一貫したサービス連携モデルを元にモデル駆動型アーキテクチャを実現することで，多様なユーザが様々なユーザインタフェースを用いて一貫してサービス連携を定義・確認することが可能であることを示した．

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．副査を引き受けて下さり，私の研究を暖かく見守って下さった慶應義塾大学環境情報学部服部隆志助教授，慶應義塾大学政策・メディア研究科高汐一紀助教授に感謝致します．また，慶應義塾大学政策・メディア研究科岩井将行講師，慶應義塾大学政策・メディア研究科博士課程由良淳一氏には，本論文の執筆にあたって，ご助言を賜りましたことを感謝致します．また，本論文を書き終えることができたのは，多くの励ましを頂いた本間暁子氏のお陰でもあります．慶應義塾大学政策・メディア研究科ならびに，KMSF 研究グループの方々には大変お世話になりました．ここに，深い感謝の意を表します．

平成17年 7月 5日
高橋 元

参考文献

[1] Franz Achermann and Oscar Nierstrasz. Applications = components + scripts ? a tour of piccola,. *Software Architectures and Component Technology, Mehmet Aksit (Ed.)*, pp. pp. 261–292, 2001.

[2] Alan F. Blackwell and Rob Hague. Autohan: An architecture for programming the home. *In IEEE Symposia on Human- Centric Computing Language and Environments,*, p. pp. 150?157., 2001.

[3] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison Wesley, Reading, Massachusetts, 1996.

[4] IEC. H.323. http://www.iec.org/online/tutorials/h323/.

[5] Laurence Melloul Gio Wiederhold Neal Sample, Drothea Beringer. Clam: Composition language for autonomous megamodules. In *International Conference on Cordination Models and Language*, April 1999.

[6] D.A Norman. *The psychology of Everyday Things*. Basic Book Inc., New York, 1988.

[7] W3C Note. Simple Object Access Protocol (SOAP) 1.1, May 2000. http://www.w3.org/TR/SOAP/.

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

[9] OMG. Model driven arhitecture, 1997. http://www.omg.org/mda/.

[10] UUID org(www.uuid.org). Uuid.

[11] Animated Programms. Toontalk 3,www.toontalk.com.

[12] Uchida. Smart-pao. http://www.uchida.co.jp/.

[13] World Wide Web Consortium. Extensible markup language(xml) 1.0, February 1999.

[14] XBow. Smart dust sensor (mote). http://www.xbow.com/.

[15] J. ピアジェ/[ほか] 著 赤塚徳郎/監訳森楙/監訳. 遊びと発達の心理学. 黎明書房, 2000.

[16] 増井俊之. 実世界指向プログラミング. 第 40 回冬のプログラミングシンポジウム予稿集. 情報処理学会, January 1999.

[17] 増井 俊之 梶尾一郎福地健太郎. Fieldmouse による実世界指向インタフェース. コンピュータソフトウェア, Vol. 18, No. 1, pp. 28–38, 2003.