JAIST Repository
https://dspace.jaist.ac.jp/

Title	センサを用いたレガシーデバイスホームネットワークの連携サービスシステムに関する研究
Author(s)	出村，哲也
Citation	
Issue Date	2006-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1985
Rights	
Description	Supervisor:丹　康雄，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

センサを用いたレガシーデバイスホームネットワークの連携サービスシステムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

出村 哲也

2006 年 3 月

修 士 論 文

センサを用いたレガシーデバイスホームネットワークの連携サービスシステムに関する研究

指導教官　丹 康雄 助教授

審査委員主査　丹 康雄 助教授
審査委員　篠田 陽一 教授
審査委員　敷田 幹文 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410083 出村 哲也

提出年月: 2006 年 2 月

Copyright © 2006 by Demura Tetsuya

概 要

近年, 各種家電のホームネットワーク規格の制定が行われ, 家電が宅内でネットワークを構成し, 外出先からの制御などが可能となってきている. その方法としては, ホームゲートウェイを介し, 各種規格対応家電に対して, 制御を行うといったもので, 各種規格対応の家電が必要になってくる. そのためにはサービス利用者は各種対応規格家電への買い換えが必要になり, 現状のネットワーク接続機能を有していないレガシーデバイスと呼ばれる家電では, サービスを利用できなかった. そこで, それらの家電に対しても各種ホームネットワーク規格対応の家電と同じようなサービスを提供できるようにレガシーデバイスの制御や状態取得に関する研究が行われている [1][2].

本研究では, それらも含めた家電のホームネットワーク化に新たなサービスとして家電同士の連携サービスを提供する手段・方法を検討し, 提案する. 本研究では, 連携サービスを考案し, それらの記述方法が必要になるため, その記述文法を定義し, 連携サービスが形式的に記述できるように試みた. さらに, 記述の字句解析, 構文解析を行う連携サービスパーザを作成した. 評価において, 連携サービスパーザの動作確認や, システムの動作確認を行い, 本研究の成果についての考察を行った.

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

近年、ECHONET や HAVi といった各種ホームネットワーク規格が制定され、規格対応家電が世の中に出始め、宅内の家電がホームネットワークを構成している。しかし、その構成の際には従来の家電であるレガシーデバイスから情報家電への買い換えが必要となっているのが現状である。それはホームネットワークのユーザにとって、経済面での負担となり、さらに、サービス内容も遠隔地からの家電の制御やボタンによる簡単な連携サービスといったものであり、現在ホームネットワーク家電の普及は進んでいない。そこで、その一つの解決方法として今後も家庭内に残ると考えられるネットワーク接続機能を持たない従来の家電（レガシーデバイス）を取り入れたホームネットワークの構築の研究が行われている [1][2]。

また、連携サービスとしてはボタンによるサービス提供が主たるサービスとなっており、そのほかのサービス提供手段がないのが現状で、サービス提供機器も情報家電に限られている。

一方、幾つもの通信機能を備えたセンサをネットワークで繋ぎ、人の動きや環境の状態を監視し、そのデータをシステムで利用しサービス提供を行うといったセンサネットワークの分野の研究も数多く行われている。

今後、レガシーデバイスを用いたホームネットワークにおいて、センサを利用し、レガシーデバイスの状態情報や、宅内の環境の情報を利用した連携サービスの提供を行うことにより、ホームネットワークの普及につながると考えられる。

1.2　研究の目的

本研究の目的は既存の家電機器であるレガシーデバイスを使用したホームネットワークや情報家電を使用したホームネットワークにおいて、宅内に配置した各種センサからの情報を利用し、家電の連携サービスを提供するホームネットワークのシステム基盤を構成することである。

本研究では、連携サービスを提供するために、第一に、どのような連携サービスがあるのかを考案し、さらに、それらの連携サービスを提供する上で必要になる連携サービス記述の記述方法について考案・検討を行う。また、記述の動作確認のため、パーザを作成し、レガシーデバイスホームネットワークシステム内に取り込み、連携サービスを提供で

きるシステムを構築し動作確認を行う。

連携サービス記述言語作成により、従来の予め定められた連携サービスだけではなく、ホームネットワークを使用するユーザがその人個人の嗜好に合った連携サービスを記述することが可能になる。

1.3 本論文の構成

本論文は以下の構成となっている。

- 第 1 章 … 本研究の背景・目的について述べる。
- 第 2 章 … 現存するホームネットワークについて、各種規格や、レガシーデバイスホームネットワークについて説明する。
- 第 3 章 … 連携サービスについて説明する。
- 第 4 章 … 連携サービスの記述言語の設計について説明する。
- 第 5 章 … 連携サービスパーザの字句解析、構文解析について説明する。
- 第 6 章 … 連携サービスを提供するシステムについて、その構成や、処理の流れについて説明する
- 第 7 章 … 提案パーザと提案システムの動作例をもとに評価について説明する。
- 第 8 章 … 本研究の今後の課題について述べる。
- 第 9 章 … 本研究における全体のまとめを述べる。

第2章　ホームネットワーク

2.1　ホームネットワークとは

ホームネットワークとは、宅内の家電機器や住宅設備をネットワークで繋ぎ、様々な情報のやりとりを行い、家電機器や住宅設備の連動動作、外出先からの制御を目的としたネットワークである。その実現のために、様々な規格が制定されている。

本章では、現在提案されている各種ホームネットワーク規格について説明する。

2.2　各種ホームネットワーク規格

2.2.1　ECHONET[8]

ECHONET はエコーネットコンソーシアムが使用の策定を行っており、家庭内の様々な白物家電製品や住宅設備を相互接続するためのホームネットワーク規格である。

ECHONET 機器の通信レイヤは大きく、アプリケーションソフトウェア、通信ミドルウェア、下位通信ソフトウェアの 3 階層に分けられる。ECHONET 規格では、通信ミドルウェア及び下位通信ソフトウェアの使用をきていしている。

アプリケーションソフトウェアは大きく、コントローラなどにおいてシステムに接続される機器を遠隔制御するアプリケーションソフトウェアと、エアコンや冷蔵庫などの個別の機器において、その機器のハードウェアとして機能を実現するソフトウェアの 2 つに分類される。

ECHONET 通信ミドルウェアはアプリケーションソフトウェアと下位通信ソフトウェアに挟まれた位置に設けられ、ECHONET 通信プロトコルに沿った通信処理を行うものである。ECHONET としての特徴の主なものはこの ECHONET 通信ミドルウェアによって実現されている。

下位通信ソフトウェアは、電灯線、無線、赤外線などの電送メディア特有の通信プロトコル処理を行うソフトウェアであり、主に OSI 参照モデルのレイヤ 1、2 に相当する通信処理を行う。下位通信ソフトウェアはサポートされる通信プロトコル毎に規定され、ECHONET 規格 Ver2.10 においては、伝送通信プロトコル、無線通信プロトコル、赤外線通信プロトコル、拡張 HBS プロトコル、LonTalk プロトコルを対象とした下位通信ソフトウェアをそれぞれ定義している。

2.2.2 DLNA[9]

DLNA(Digital Living Network Alliance) は、家電・パソコン・モバイルなどの機器間でデジタルコンテンツ（音楽・写真、ビデオなど）を家庭内で簡単に共有するため、業界標準技術に基づいたオープンな相互接続互換性を構築するための技術的な設計ガイドラインを策定することを目的とし、2003 年 6 月に設立された非営利団体である。

DLNA 機器は DMS(Digital Media Server) と DMP(Digital Media Player) び 2 クラスに分かれ、機能が定義されている。また、フレームワークは次のような標準化技術が使用される。

1. 物理・リンク層
   IEEE802.3i/u、IEEE802.11a/b/g
2. ネットワーク層
   IPv4
3. 機器発見
   UPnP(Universal Plug and Play) の Device Architecture Version1.0
4. 機器制御
   UPnP DCP(Device Control Protocol) の AV Version1.0
5. トランスポート層
   HTTP(Hper Text Transfar Protocol)

2.2.3 HAVi[10]

HAVi(Home Audio Video Interoperability) は、ホームネットワークに対応した高速シリアルバス IEEE1394 のインターフェースを備えた AV 機器のための仕様である。HAVi 対応の AV 機器は、高速なネットワークを通して相互に接続され、連携して動作させることができる。HAVi はホームネットワークを分散コンピューティングプラットホームとして捉え、その上で分散アプリケーションが各々強調して動作するための環境、異種メーカーの機器による相互接続・相互運用を実現するため、API のセットとミドルウェアから構成される。ミドルウェアには、ネットワーク上の機器を自動的に検知する機能や、異なるメーカーの機器同士を相互接続する機能などが備わっている。

HAVi では、機器を機器全体と機器に含まれる昨日要素に区別して取り扱い、機器の抽象化を行っている。機器の制御は、コントローラ上のアプリケーションが対象となる機器に対して何らかの方法で制御命令を送信することで行われる。しかし、このコマンドに対応していない機器やメーカ独自の機能を組み込みたい場合には標準のコマンドを使うことが不可能である。そこで HAVi では、機器固有の制御方法をカプセル化し、機器と昨日要素のモデル化を行うことで問題の解決を図っている。

2.2.4 X-10[11]

X-10 は電送媒体として電灯線を使用してネットワークに接続された X-10 対応機器の電源を操作するための規格である。

X-10 では、機器の電源操作や照明機器の明かりの調整を主な目的としており、機器の持つ機能の中で、電源に関する部分と照明器具の照度に関する部分のみを抽象化している。

機器の内部機能は

- 電源制御機能
- 照度制御機能
- 通信制御機能

の 3 種類としている。

また、機器の制御は

- 電源操作
- 照明機器の明るさ調整
- 機器の状態確認

といった各内部機能を実行させるコマンドを通知することで行われる。

そのコマンドを表 2.1 に示す。

2.3 レガシーデバイスホームネットワークとは

レガシーデバイス（Legacy Device）ホームネットワークとは、既存の家電機器であるレガシーデバイスを用いて作成されたホームネットワークで、家電をホームネットワーク規格対応の家電に買い換えなくても、それらのサービスを提供できるシステムである。

2.4 現状のレガシーデバイスホームネットワーク

現状のレガシーデバイスホームネットワークシステムは、レガシーデバイスを複数の部分状態機械からなる複合状態機械として捉え、状態機械に対する入力である赤外線信号を捕捉するセンサを設置し、レガシーデバイスの内部の状態情報を補足し、機器の状態情報を管理する。また、その制御は学習リモコンを用いて制御コマンドを送信し機器の制御を行う。

現状のレガシーデバイスホームネットワークシステムの今後の課題として、状態取得方法の検討、例外や障害の発生時の対応、各種連携サービスの提供などがある。

表 2.1: X-10 コマンド表

コマンド	Data Code(D1 D2 D4 D8)	Function Code(D16)
ALL Unit OFF	0000	1
ALL Light ON	0001	1
ON	0010	1
OFF	0011	1
Dim	0100	1
Bright	0101	1
ALL Light OFF	0110	1
Extended Code	0111	1
Hail Request	1000	1
Hail Acknowledge	1001	1
Pre-Set Dim1	1010	1
Pre-set Dim2	1011	1
Extended Data Transfar	1100	1
Status ON	1101	1
Status OFF	1110	1
Status Request	1111	1

第3章　連携サービス

現在、宅内環境において家電機器の現状は、家電機器のホームネットワーク化により、遠隔からの制御、ホームネットワークシステム内で家電機器の現在の状態の管理を行うことが可能である。それを利用し、複数の家電を一括操作、連携動作させる連携サービスの提供がサービスとして注目されている。

本章では、本研究で取り扱う連携サービスとはどのような定義に基づくサービスか、連携サービスとしてどのような連携サービスが存在するかについて説明する。

3.1　連携サービスとは

普段、ユーザは複数の機器を操作して、ユーザの住みやすい環境に整えている。連携サービスとは、このような一連の操作を自動で行ってくれるサービスのことである。その提供手段として、一連の操作のきっかけとなるものが必要となっている。そのきっかけのことをトリガーと呼ぶ。現状のトリガーは、ボタンによるものである。

本研究で取り扱う連携サービスは、トリガーとして以下の種類のものを考えている。

- 機器の状態変化
- 連携サービス提供ボタン
- イベントタイマー
- スケジュールタイマー

これらのトリガーにより連携サービスが動作する。

機器の状態変化をトリガーにしたものは、ユーザが操作したある機器の状態の変化によって、またはセンサからの環境情報の変化によって連携サービスを提供するものである。このトリガーを用いた連携サービスは現存していない。

連携サービス提供ボタンをトリガーにしたものは、各種連携サービスを定義したボタンを押すことによって連携サービスを提供するものである。このトリガーを用いた連携サービスとしては、東芝のネットワーク家電 FEMIITY[12] など一般的に使われているサービス提供方法であるが、本研究では、提供機器を情報家電に限らず、レガシーデバイスにも対応した連携サービスの提供を行う。

イベントタイマーをトリガーにしたものは、ユーザの機器操作が行われることで、連携サービス定義として定められた時間でタイマーを開始し、そのタイマーが定められた時間を経過することで連携サービスを提供するものである。このトリガーを用いたサービスは、一般にTVなどのおやすみタイマーで使用されている。本研究では、TVが持つ内部タイマー情報は使用せず、ホームネットワークシステム内にてタイマー情報を管理し、サービス提供を行う。そうすることで、内部にタイマー情報を持たない家電機器に対してもタイマーでのサービスを提供することができる。

スケジュールタイマーをトリガーにしたものは、実際の時間（何時何分）によって連携サービスを提供するものである。このトリガーを用いたサービスは、機器の操作はないものの携帯電話の目覚ましなど現在時間（時計)を持つ機器で使用されている。本研究では、目覚ましなどの音を鳴らす部分がTVの電源操作などの機器の操作・制御となる。

3.2 連携サービスの考案

3.2.1 使用想定機器の決定

連携サービスを考案する上で、連携サービスに使用することが想定される機器を挙げ、その機器毎にどのような連携サービスが提供できるかを検討する。

連携サービスの提供機器は、主な分類として、映像や音声に関するAV機器、センサからの情報である環境、一般に白物家電と呼ばれる電化製品、その他のものに分けられる。

その使用想定機器一覧を以下に示す。

- AV機器
  VTR、TV、スピーカー
- 環境
  人の位置、内の環境、外の環境、扉（入退室)、時計、照明
- 電化製品
  エアコン、冷蔵庫、洗濯機、電気ポッド、レンジ
- その他
  扇風機、窓、カーテン、携帯

3.2.2 網羅性の考慮方法

前節3.2.1で挙げられた機器で、その各々の機器間の網羅性の考慮のため、その中から2つの機器を取り出し、その全組み合わせで連携サービスがあるかどうかを人手により確認を行う。

また、単に2つの機器を選択し確認を行うだけでなく、その2つの機器間の方向性として、動作元機器と動作先機器として区別する。その方向性を持つ2つの機器の総当たり表にて、連携サービスの抜けがないかをチェックし、表中の連携サービスのない機器などの不要な部分を削除し、連携サービスの存在する必要な部分のみ抜き出し、表を集約した。

3.2.3 連携サービス表

前述までの使用想定機器の決定、網羅性の考慮方法によって作成された表を図3.1に示す。

表の縦軸は動作元の機器、横軸は動作先の機器を表し、表中の → の前は動作元の機器（トリガー）の現在の状態、→ の先が動作先の機器の変化後の状態情報を表している。

2つの機器の連携								
動作元→ ＼ →動作先	VTR	TV	スピーカー	照明	エアコン	扇風機	窓	カーテン
VTR		ON→ON	ON→ON	番組の種類→照度の調整				番組の種類→OPEN/CLOSE
TV	OFF→OFF		ON→ON OFF→OFF	番組の種類→照度の調整				番組の種類→OPEN/CLOSE
扉		入室→ON 退室→OFF	入室→ON 退室→OFF	入室→ON 退室→OFF	入室→ON 退室→OFF			
時計	時刻→時刻調整	時刻→ON/OFF		時刻→ON/OFF	時刻→ON/OFF			時刻→OPEN/CLOSE
人の位置		位置によって→距離・向き調整	位置によって→向き調整	位置によって→ON/OFF	位置によって→風向き調整	位置によって→風向き調整		
外の環境				暗い→ON 明るい→OFF	温度→運転の切替			暗い→CLOSE
内の環境				暗い→ON 明るい→OFF	温度→ON/OFF・風量調整			
窓					OPEN→OFF			
エアコン					風向きの調整	風向き→位置調整	ON→CLOSE	

図 3.1: 連携サービス表

TV と VTR では、TV が動作元で VTR が動作先の場合、TV の電源が OFF によって、VTR の電源が OFF といった省エネサービスや、VTR が動作元で TV が動作先の場合、VTR の電源が ON によって TV の電源を ON といった個別機器操作省略サービスがある。

扉の入退室検知センサを利用したサービスとして、入室検知で TV の電源 ON、退室検知で TV の電源 OFF といったサービスがある。また、TV に限らず、スピーカーや照明、

エアコン等にもこのサービスは利用できる。

外の環境の情報を利用したサービスとして、外の環境が暗かったら照明を ON といったサービスや、気温によって、運転モードの自動切り替えなどのサービスがあり、内の環境の情報を利用したサービスは、明るかったら照明 OFF といったサービスや、温度によってエアコンの電源操作などがある。

3.2.4 その他の連携サービス

前節 3.2.3 の表はトリガーを機器の状態変化とした 2 つの機器間の連携サービスである。しかし、連携サービスは状態変化をトリガーとした 2 連携のサービスだけではなく、その他の連携サービスも存在する。その他の連携サービスとして、3 連携のサービスやボタンやタイマーをトリガーとした連携サービスがある。

3 連携のサービスの取り扱いは、2 連携のサービスの組合わせ、ボタンによる一括操作によって実現できる。そのための手段としては、2 つの機器の連携サービスの提供結果により生じる機器の状態変化を検知し、再度 2 つの機器間で連携サービスを行うことにより、結果的には、3 連携以上の連携サービスを提供するといった方法や、ボタン操作によって、2 つ以上の機器を制御するといった方法により可能となる。

トリガーが異なるものとしては、以下のサービスシナリオ例にて例を挙げる。

その他の連携サービスでの表作成は、数が膨大なため困難なので、表作成ではなく、別々に１つずつ挙げていく方法をとる。

3.3 連携サービスのシナリオ例

以下に連携サービスの具体的なシナリオ例をトリガー毎にいくつか挙げる。

この連携サービスのシナリオ例は、動作後の機器の状態が全く同じサービスであってもトリガーの違いによって別のサービスとして分類されている。

- 機器の状態変化
  機器の状態変化を検知して連携サービスを提供する
  - 映画鑑賞
    DVD の電源を入れることで、TV の電源を入れ、チャネルを VIDEO にし、スピーカーを 5.1ch に設定し、部屋のカーテンを閉め、照明を暗くする。
  - おかえり
    外出（扉）により、照明、エアコン、TV の電源を切る。
  - おでかけ
    帰宅（扉）により、照明、エアコン、TV の電源を入れる。

  - TV と VTR（１）
    VTR の電源が入ると、TV の電源を入れ、チャンネルを VIDEO にする。
  - TV と VTR（２）
    TV の電源を切るかつ VTR の録画予約なしで、VTR の電源を切る。

- 連携サービス提供ボタン
  各種ボタンを設定し、それらを押すことにより連携サービスを提供する。

  - 映画鑑賞
    映画鑑賞ボタン ON により、DVD、TV の電源を入れ、TV のチャネルを VIDEO にして、スピーカーを 5.1ch に設定し、部屋のカーテンを閉め、照明を暗くする。
  - おかえり
    お帰りボタン ON により、照明、エアコン、TV の電源を入れる。
  - おでかけ
    お出かけボタンにより、照明、エアコン、TV の電源を切る。
  - おやすみ
    お休みボタン ON により、TV、照明の電源を切る。

- イベントタイマー
  ある機器が操作されてからの時間、相対時間によって連携サービスを提供する。

  - おやすみ
    照明の電源を切った後、30 分後に TV の電源を切る。
  - 省エネ
    ある特定機器が操作されてから一定時間操作されずに電源 ON の状態の時、その機器の電源を切る。

- スケジュールタイマー
  接待時刻によって連携サービスを提供する。

  - おはよう
    毎朝起床時間になったら、TV の電源を入れ、照明をつけ、エアコンの電源を入れる。
  - おやすみ
    毎晩就寝時間になったら、TV の電源を切り、照明を切る。

3.4　連携サービスの考察・まとめ

考案した連携サービスから、連携サービスは、2 連携でとどまることは少なく、多くのサービスは 3 連携以上で初めて、提供されて嬉しいサービスとなることがわかる。また、

考案した連携サービスの多くはAV機器を利用している。特に電源動作によるサービス提供が多く、これは、使用機器の主な機能を使う際に電源操作が必要であり、電源の入力によりその家電機器が意味を成すことが多いと考えられる。つまり、電源処理に仕様用途が集約していると考える。このことから、連携サービスの大部分は電源操作を行うことが可能となれば、提供可能となる。逆に、電源以外の機器のオプション操作を使用したサービスが少ない。これは、オプションはあくまでもオプションであり、人によりそれを使用するかしないかに分かれるためであると考える。

本章で挙げた連携サービスは人によって不必要なサービスであったり、必要なサービスが入っていないということがある。この点については、人手による連携サービスの考案作業のため、その人個人の嗜好が取り入れられているためで、仕方のないことと考える。この改善は、今後のサービス記述言語の作成により、個人の嗜好を取り入れたサービス提供を行うことが可能となる。

第4章　連携サービス記述言語の設計

第3章の連携サービスを提供する上でそのサービス記述が必要になってくる。
本章では、その連携サービス記述言語の設計について説明する。

4.1　BNF記法

連携サービスの記述の構文は基本的にはBNF記法によって記述される。
言語を作成するためには、その言語の厳密な定義が必要である。ここでは、言語の構文規則を厳密に定義する記述方をしてバッカス記法と構文図式を説明し、記述の元となる文脈自由文法とその言語の形式的な定義を与える。

- バッカス記法
  バッカス記法は構文を明確的に定義するものである。バッカス記法で識別子または名前と呼ばれるものを定義すれば次のようになる。

```
<suuji> ::- 0|1|2|3|4|5|6|7|8|9
<eiji> ::- a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<namae> ::- <eiji>|<namae><eiji>|<namae><suuji>
```

ここで <> で囲まれたものを構文要素と呼ぶ。この記述では、<suuji> は0〜9のどれかである、<eiji> はa〜zどれかであると読むことができる。すなわち、::– の左辺にある構文要素が右辺によって定義されている。| はまたはという意味を表す。<namae> は <eiji>、<namae><eiji>、<namae><suuji> のどれかであると読むことができる。
すなわち、<namae> は <eiji> 一文字、もしくは <eiji> で始まり、その後に <eiji>、<suuji> をつけたものということを定義している。

4.2　言語仕様

記述の構文ルールや字句ルールは第5章で詳しく説明を行う。
本節では、サービス記述する際に使用できる記述語句について説明する。
表4.1-表4.5に記述で使用される各種定義を示す。

表 4.1: 基本記述コマンド定義

def	サービス名の記述
if	条件部分の記述
then	コマンド部分の記述
com	コマンド動作の記述

表 4.2: 条件部、コマンド部記述定義

VAR	変数
EXP	記号
CONST	定数

変数と値

変数は名前の. つながりで表現される。

例）*name.name*

その変数の意味する値としては、登録されているデータベースの値を参照する。

例）*tv.power* → AP_TYPE が TV で、POWER の状態を参照

定数は STATE から始まる文字列で表現される。
それ以外の文字列や数字との組み合わせは名前 (name) として扱う。

サービス記述

サービス記述は **def** *name* **if** *condition* **then** *command* で表現される。
この部分は、大きく定義部分とサービス部分に分けられる。

- 定義部分

  例）**def** *name*

  ここではサービス名を定義する。
  システム側で現在動作しているサービスがどれかを判断する際にこのサービス名が使用される。

- サービス部分

例）**if** *condition* **then** *command*

ここでは、サービスの実行条件やその実行動作を定義する。

サービス実行条件である *condition* 部分は
<VAR> <EXP> <VAR> または <VAR> <EXP> <CONST>
の形を取り、
サービス実行動作である *command* 部分は
com (<VAR> , <CONST>)
の形を取る。
VAR は変数を表し、EXP は条件式、CONST は定数を表す。
VAR、EXP、CONST の一覧は表 4.3、表 4.4、表 4.5 に示す。
記号を用いた記述例

- 条件式
  トリガー条件の記述は==で表す。
  例） tv.power == STATE_ON
  その他条件の記述は=で表す。
  例） vtr.power = STATE_OFF
  含有条件の記述は $\ni$ で表す。
  例） tv.connect $\ni$ vtr.apid
  条件不一致の記述は $\neq$ で表す。
  例） vtr.playrec $\neq$ STATE_REC
- 接続式
  接続式は *condition* や *command* に対して用いられる。
  例） *condition* & *condition*
  command $\bigwedge$ *command*
  command $\bigvee$ *command*

区切り文字

サービス記述における区切り文字はスペースで表す。

表 4.3: 変数内使用記述定義

tv	TV の指定
vtr	VTR の指定
speaker	スピーカーの指定
light	照明の指定
aircon	エアコンの指定
fan	扇風機の指定
window	窓の指定
curtain	カーテンの指定
door	扉の指定
clock	時計の指定
mobilephone	電話の指定
apid	家電の ID の指定
apnick	家電のニックネームの指定
room(id)	部屋の指定
power	電源状態の指定
chanel	チャネル状態の指定
vol	音量状態の指定
playrec	再生・停止などの状態の指定
rec	録画予約の状態の指定
connect	接続条件の指定
temp	温度センサの指定
humid	湿度センサの指定
bright	照度センサの指定
value	センサ値の指定
timer(id)	タイマー ID の指定
hogeb	各種ボタンの指定
state	ボタンの状態の指定

表 4.4: 記号種類一覧

条件式	==	トリガーとなる機器の状態条件
	=	トリガー以外の機器の状態条件
	≠	条件不一致
	∋	含有条件
接続式	&,∧	アンド条件
	∨	または、オア条件

表 4.5: 状態情報定数一覧

STATE_ON	機器の状態が電源 ON の状態
STATE_OFF	機器の状態が電源 OFF の状態
STATE_CUT	機器の状態が電源カットの状態
STATE_CHx	機器のチャネルが x の状態
STATE_VOLx	機器の音量が x の状態
STATE_LINEx	機器のライン入力が x の状態
STATE_MONO	機器の音声がモノラルタイプの状態
STATE_ST	機器の音声がステレオタイプの状態
STATE_5.1	機器の音声が 5.1ch タイプの状態
STATE_PLAY	機器が再生状態
STATE_STOP	機器が停止状態
STATE_FF	機器が早送り状態
STATE_REW	機器が巻き戻し状態
STATE_PLFF	機器が再生早送り状態
STATE_PLREW	機器が再生巻き戻し状態
STATE_REC	機器が録画状態
STATE_PAUSE	機器が一時停止状態
STATE_REC	機器が録画状態
STATE_LOW	照度が低い（暗い）状態
STATE_HIGH	照度が高い（明るい）状態
STATE_OPEN	窓などの状態が OPEN の状態
STATE_CLOSE	窓などの状態が CLOSE の状態
STATE_IN	扉の状態が IN の状態
STATE_OUT	扉の状態が OUT の状態
STATE_COOL	機器の運転状態が冷房の状態
STATE_HEAT	機器の運転状態が暖房の状態
STATE_DRY	機器の運転状態が除湿の状態
STATE_AUTO	機器の運転状態が自動の状態
STATE_RIGHT90	機器の向き状態が右向き 90 度の状態
STATE_RIGHT45	機器の向き状態が右向き 45 度の状態
STATE_CENTER	機器の向き状態が真正面の状態
STATE_LEFT45	機器の向き状態が左向き 45 度の状態
STATE_LEFT90	機器の向き状態が左向き 90 度の状態
STATE_LOW	機器の風量が弱の状態
STATE_MID	機器の風量が中の状態
STATE_HIGH	機器の風量が強の状態
STATE_TIMEx	タイマーの時間が x の状態 (ランダム時間の記述は x に RANDAM を記述)

4.3 サービス例の記述例

作成した言語仕様を用いて記述したサービス例をいくつか挙げる。

機器の状態の変化をトリガーとしたものの記述として TV と VTR のサービス (1)、部屋の環境により照明をつけるサービス (4)、連携サービス提供ボタンを用いた記述例として、おかえりサービス (2)、イベントタイマーを用いた記述例として、お休みサービス (3)、スケジュールタイマーを用いた記述例として、おはようサービス (5) を記述する。

1. def service1 if room1.vtr.power == STATE_ON & room1.tv.power = STATE_OFF & room1.vtr.connect ∋ room1.tv.apid then com (room1.tv.power , STATE_ON) & com (room1.tv.ch , STATE_CHVIDEO)

2. def service2 if ghb.state == STATE_ON then com (room1.tv.power , STATE_ON) & com (room1.light.power , STATE_ON) & com (room1.aircon.power , STATE_ON)

3. def service3 if room1.light.power == STATE_OFF & timer1.value = UNUSED then com (timer1.value , STATE_TIME30) if timer1.value == STATE_TIME0 then com (room1.tv.power , STATE_OFF)

4. def service4 if room1.bright.value == STATE_LOW then com (room1.light.power , STATE_ON)

5. def service5 if time.value == STATE_TIME0730 then com (room1.tv.power , STATE_ON) & com (room1.right.power , STATE_ON)

4.4 サービス記述言語の考察

4.4.1 記述方法のメリット・デメリット

記述方法のメリットは、ユーザによってサービスが記述できることである。すなわち、サービス提供が予め決められたサービスのみではなく、ホームネットワークを使用するユーザが実行したい連携サービスをユーザ自身の手で記述することができるので、ユーザ毎に異なる嗜好を持っているが、それに対応したサービスを記述できる。

しかし、この記述を書く際には、現状の記述は十分に一般化されておらず、サービス記述者の側で登録家電の状況や、タイマーの使用用途などを把握しておき、記述の際に取り入れる必要がある。

4.4.2 現状のサービス記述言語の限界

現状のタイマー記述や機器指定の方法には、ユーザ側でどのような機器が登録されているかやその機器に対してはどのような状態があるのかといった機器の情報の全部を把握する必要があるため、サービス記述者の質が問われることになる。それらの情報はシステム側で確認することもできるが、記述の際には、確認作業を行いながら記述するのことは少ないと考えられる。そういった点で、記述の際にその機器の状態などがわからなくなった際にサービス記述をすることができなくなることがある。

今後、記述の際に記述者に対して何らかのアシストの方法が必要であると考えられる。

第5章　連携サービスパーザ

連携サービスパーザでは字句解析、構文解析、サービス記述判断、構文木の作成を行う。本章ではその解析方法や解析ルールについて説明する。

連携サービスパーザの処理は以下の手順で行われる。

1. 記述されたサービスファイルを読み込みを行う。
2. そのサービスファイルの記述に対して、字句解析ルールによって字句解析を行う。
3. 字句解析の結果を受けて構文解析ルールによって構文解析を行う。
4. 構文解析の結果が受理できれば、構文木の作成を行い、構文木を保持しておく。

連携サービスパーザの処理の流れを図 5.1 に示す。

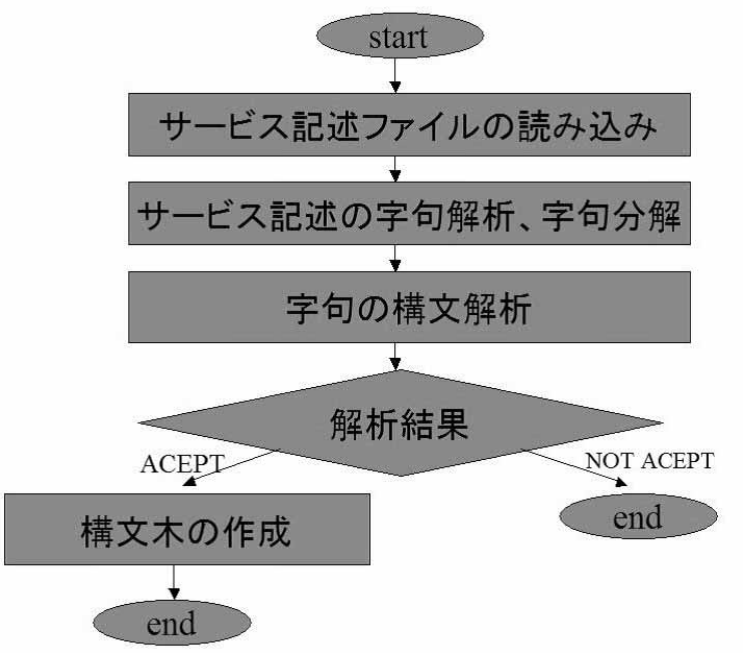

図 5.1: パーザの流れ

5.1 字句解析

記述された連携サービスから、区切り文字であるスペースまでの文字を一つ一つ抜き出し、その文字列の種類を調べ、トークンとして保持する。

5.1.1 字句解析ルール

以下に字句解析のルールを示す。

- DEFINE ::- def
- IF ::- if
- THEN ::- then
- COM ::- com
- ALPHA ::- [a-zA-Z] |- |_
- NUMBER ::- [0-9]
- EXP1 ::- == |= |≠ |∋
- EXP2 ::- ∨ |∧ |&
- NAME ::- ALPHA(ALPHA)*(NAME)*
- VAR ::- NAME(.NAME)*
- CONST ::- STATE_ (NAME)

syntax diagram は付録として添付する。

5.1.2 システム内動作

字句解析は 5.1.1 節のルールによって、サービス記述ファイルに記述されている文字の一つ一つを調べ、区切り文字として定義したスペースまでを一つの字句として扱い解析を行う。その結果を後の構文解析で使用するためトークンとしてシステム内で保持しておく。

字句解析ルーチンを図 5.2 に示す。

トークンの種類を表 5.1 に示す。字句解析により作成されるトークンは表の 10 番台、20 番台のトークンである。30 番台のトークンは構文解析によって作成される。

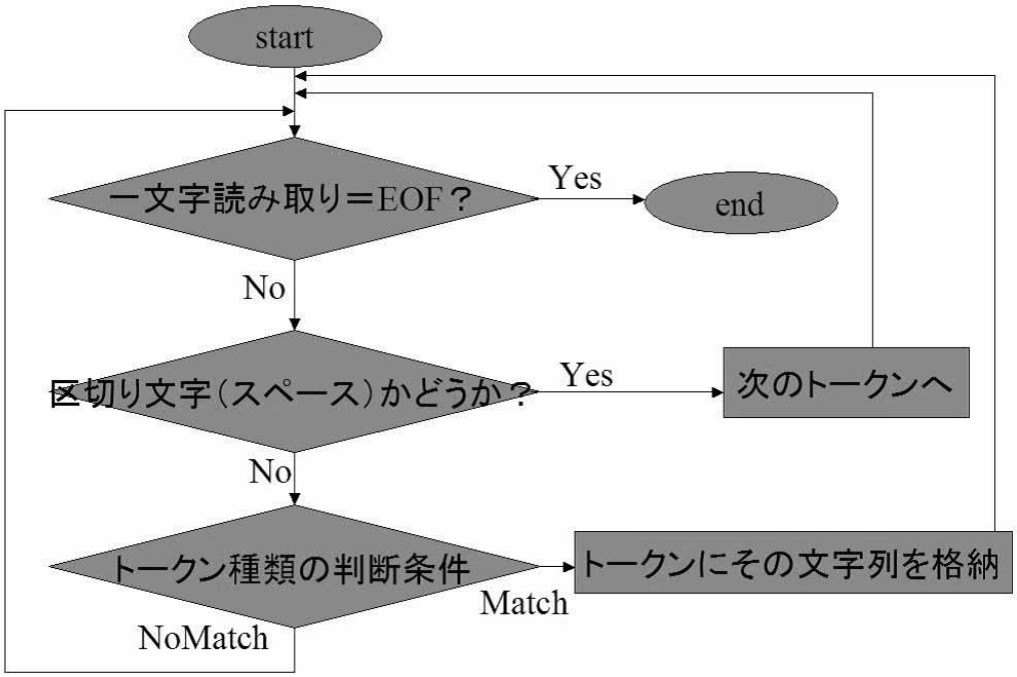

図 5.2: 字句解析の流れ

表 5.1: 字句のタイプ

字句タイプ	番号
DEFINE	10
NAME	11
IF	12
THEN	13
COM	14
VAR	15
EXP1	16
EXP2	17
CONST	18
KAKKO	19
KOKKA	20
KANNMA	21
CONDITION	30
COMMAND	31
DEFINEPART	32
SERVICEPART	33
SERVICEDSCL	34

5.2 構文解析

構文解析では字句解析で得られたトークンをもとに、構文解析のルールによって、記述として受理できる記述か否かを判断する。

5.2.1 構文解析ルール

以下に構文解析のルールを示す。

1. <SERVICEDSCL> ::- <DEFINEPART> <SERVICEPART>
2. <DEFINEPART> ::- <DEFINE> <NAME>
3. <SERVICEPART> ::- <SERVICEPART> <SERVICEPART>
4. <SERVICEPART> ::- <IF> <CONDITION> <THEN> <COMMAND>
5. <CONDITION> ::- <CONDITION> <EXP2> <CONDITION>
6. <CONDITION> ::- <VAR> <EXP1> <VAR>
7. <CONDITION> ::- <VAR> <EXP1> <CONST>
8. <COMMAND> ::- <COMMAND> <EXP2> <COMMAND>
9. <COMMAND> ::- <COM> (<VAR> , <CONST>)

syntax diagram は付録として添付する。

5.2.2 システム内動作

構文解析は 5.2.1 節のルールによって、字句解析の結果を構文的に受理できる記述かどうかの判断を行い、受理可能ならばその構文木を作成する。解析を行う際にはスタック方式を用いて構文解析を行う。

構文解析ルーチンを図 5.3 に示す。

スタック

スタック（stack）は積み重ねという意味で、コンパイラの中ではいろいろな部分で使用される。スタックは後に入れたものほど先に取り出される仕掛けである。

本研究では、構文解析においてスタックを用いて解析を行っている。字句を一つ一つスタックに積み込み、そのスタック内で、構文ルールにあったものがあれば、スタックから取り出し、ルールを適用後に変換後の字句をスタックに積み込む。この操作を繰り返し行い、最終的にはスタックの中にサービス記述を表す SERVICEDSCL のトークンが現れる。

スタックの動作例を図 5.4-図 5.6 に示す。図 5.4 のように、まずスタックが空の状態から始まり、トークンが次々に格納されていく。構文解析ルールに適用できるトークンの並びがきたらその部分をトークンから抜き出し、構文解析ルールを適用し、一つにまとめ置換

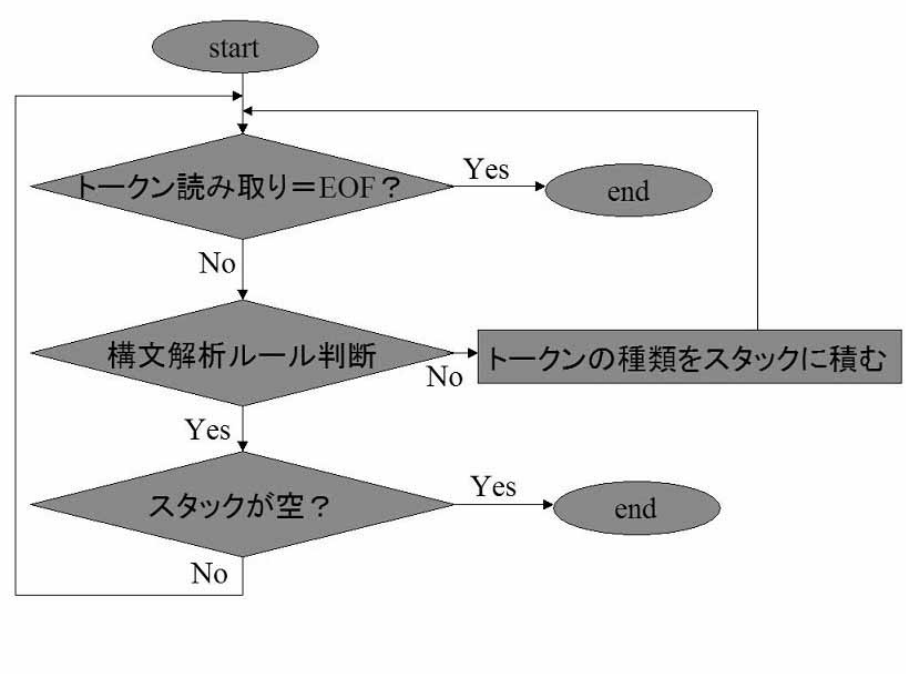

図 5.3: 構文解析の流れ

されたトークンをスタックに積む。その後、さらにトークンを格納し、ルール適用を繰り返す。そうすることで、最終的に受理できる記述の場合は、図 5.6 のような SERVICEDSCL がスタックに残る。

図 5.4: スタック内の動作 1

図 5.5: スタック内の動作 2

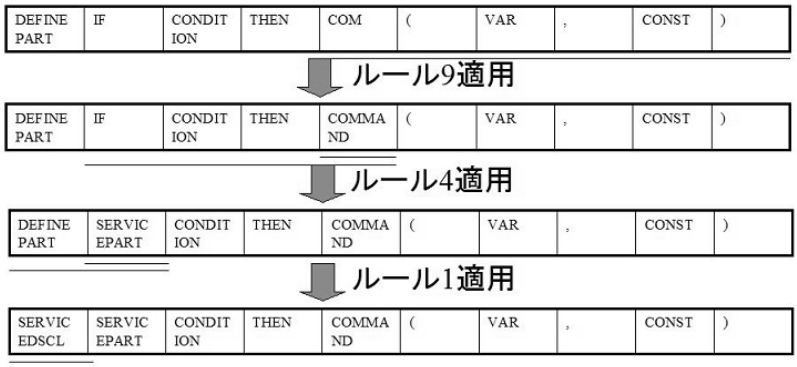

図 5.6: スタック内の動作 3

5.3 構文木の作成方法

構文木の作成は、構文解析の結果、記述が受理された際に行われる。

作成方法は、構文解析中にスタックに積み込み、解析していく作業を木のノードに追加する作業へと置換する。積み込み順番、ルール適用順番に木のノードは作成され、ノード内には、ノード番号、ノードに格納されたトークンタイプ、親子関係を持つ。親子関係はルール適用時にのみ追加され、それ以外の時は、ノード番号、トークンタイプの格納処理を行う。

スタック時の説明で例に挙げたサンプルで構文木を作成したときの初めの部分を以下に説明する。

1. ノード DEFINE をノード番号 1、ノードタイプ 10 で格納
2. ノード NAME をノード番号 2、ノードタイプ 11 で格納
3. ルール 2 の適用により作成されたノード DEFINEPART をノード番号 3、ノードタイプ 32 で格納し、ノード番号 1 と 2 の parent にノード番号 3 を、ノード番号 3 の reft、right に 1 と 2 を格納

この作業を最終ノードである SERVICEDSCL まで続ける。ノードの子として 2 つ以上ある場合には、reft、right に続けてデータとして格納されている。

第6章　連携サービス提供システムの提案

本章では、連携サービスを提供するシステムについて説明する。

6.1　システム構成

本提案システムは既存のレガシーデバイスホームネットワークシステムを部分的に利用して作成する。レガシーデバイスホームネットワークシステムに連携サービスの記述部分や判断部分（連携サービスパーザ部分）を追加し、その記述をもとに既存のシステムで動作する機器を制御する。

そのシステム構成図を図 6.1 に示す。

宅内におけるシステムの配置イメージを図 6.2 に示す。

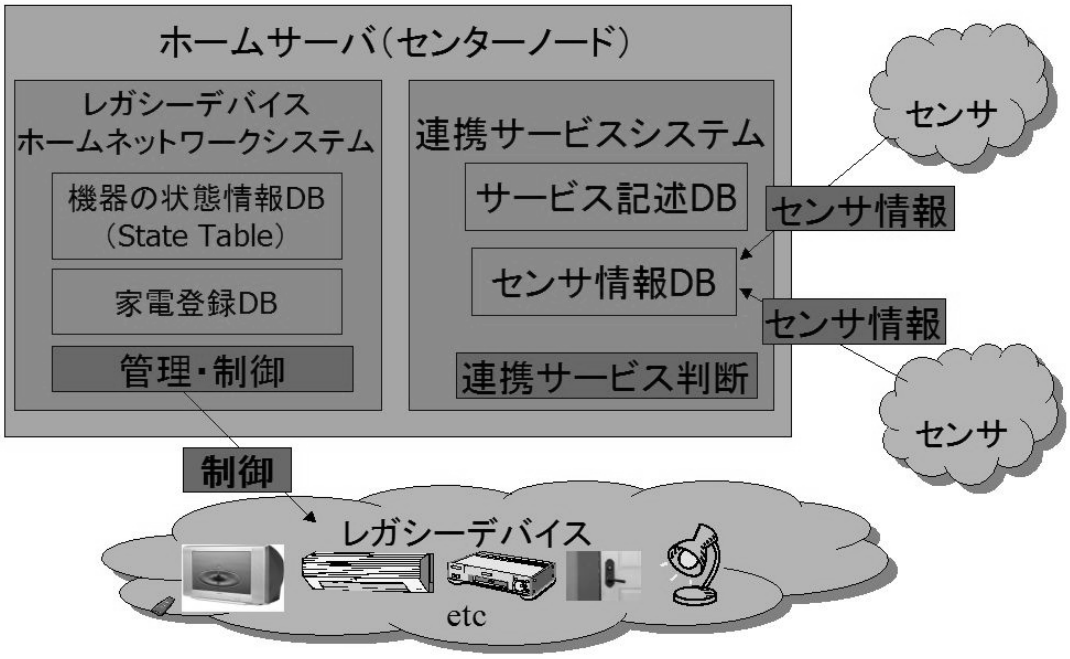

図 6.1: システム構成

本システムは LINUX 開発環境で、構成されている。そのため、LINUX 環境を構築できるものであれば、とりあえずはサーバとして動作する。実際に使用するとなると、他に

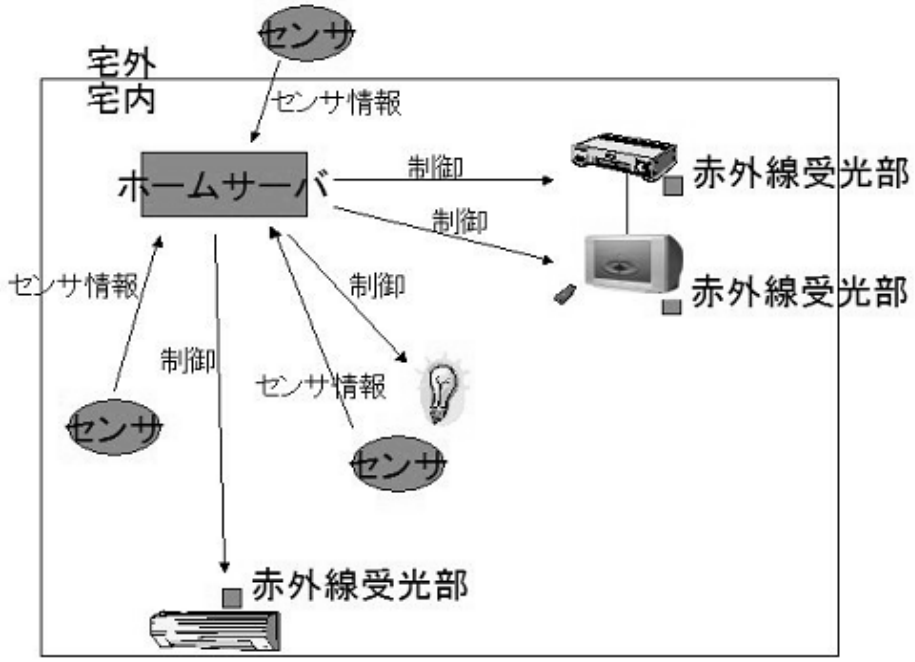

図 6.2: 宅内におけるシステム配置

接続端子類が必要であるなどの条件がある。

ホームサーバシステムのユーザインターフェースは、コマンドラインとなっており、0-9までのメニュー選択により、動作させる。

そのメニューの一覧を表 6.1 に示す。

以下、各メニュー項目の詳細を 1 つずつ説明する。

6.1.1 家電の登録

宅内にある家電をシステムのデータベースに登録する。
登録項目は、家電の ID、タイプ、メーカ、モデル、ニックネーム、存在する部屋 ID、接続状況、その家電に付随するセンサ情報（センサ ID、タイプ）である。

家電の登録はシステム動作時の初期設定として必ず必要な項目となっている。そのため、毎回同じ登録作業を繰り返す手間を省くために、一度行った登録作業をファイルに書き出し、次回からはその登録ファイルの読み込みによって、家電の登録を行うことを可能としている。

登録時の初期状態は STATE_OFF、STATE_CH1、STATE_VOL0、STAE_LINE0、STATE_AUTO、STATE_CENTER、STATE_MID である。したがって、機器の登録時に実際の機器をその状態にして登録を行うか、6.1.3 節の状態遷移の初期化により登録後に初期化を行うかが必要になる。

登録処理の流れを図 6.3 に示す。

表 6.1: システムのメニュー一覧

番号	動作内容
0	システムの終了
1	家電の登録
2	家電の削除
3	状態遷移管理の初期化
4	状態遷移の動作確認
5	連携サービス記述
6	連携サービスパーザの動作
7	現在の登録家電の状態確認
8	連携サービスの定義（初期登録サービス）
9	連携サービススタート

6.1.2 家電の削除

登録されている家電をシステムのデータベースから削除する。
削除には家電 ID を指定して行う。
削除された家電 ID の部分に新たな家電を登録することが可能である。
削除処理を行う流れを図 6.4 に示す。

6.1.3 状態遷移の初期化

何らかの異常が起こった際にシステム内部の状態を初期状態にリセットするための手段を提供する。

初期化される状態は登録時の初期化状態と同じ状態に初期化される。これはシステム内で管理している内部情報の初期化作業なので、実際の機器をこの初期化作業によって、制御し、初期化を行うことはできない。実際の機器はこの初期化作業後に、人手によって初期化する必要がある。

状態遷移の初期化を行う流れを図 6.5 に示す。

6.1.4 状態遷移の動作確認

状態遷移の動作を確認する。

システム内部で管理している状態遷移が正常に動作するかを疑似コマンドの発生により確認する。

状態遷移の動作確認を行う流れを図 6.6 に示す。

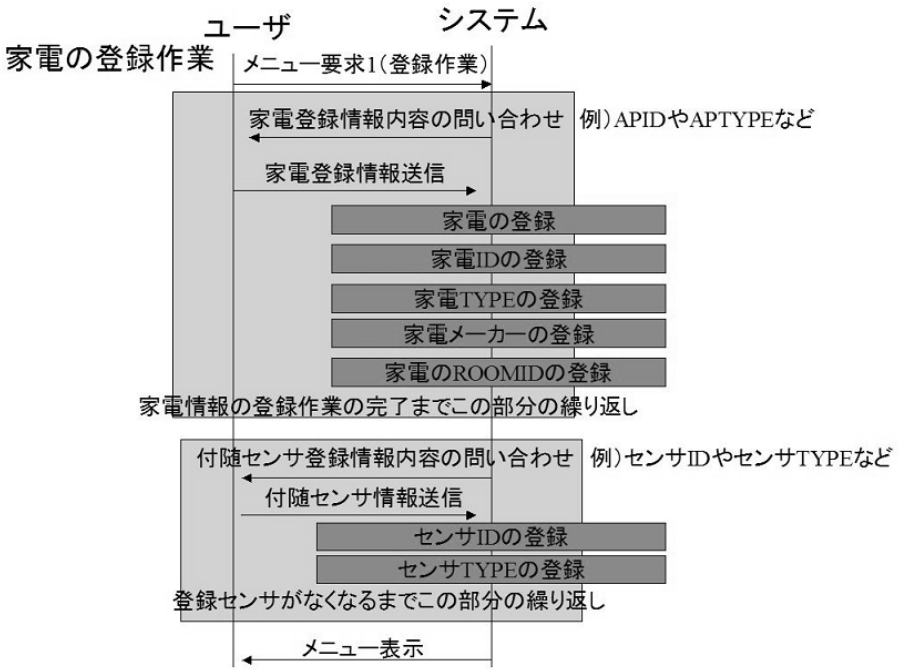

図 6.3: 家電の登録

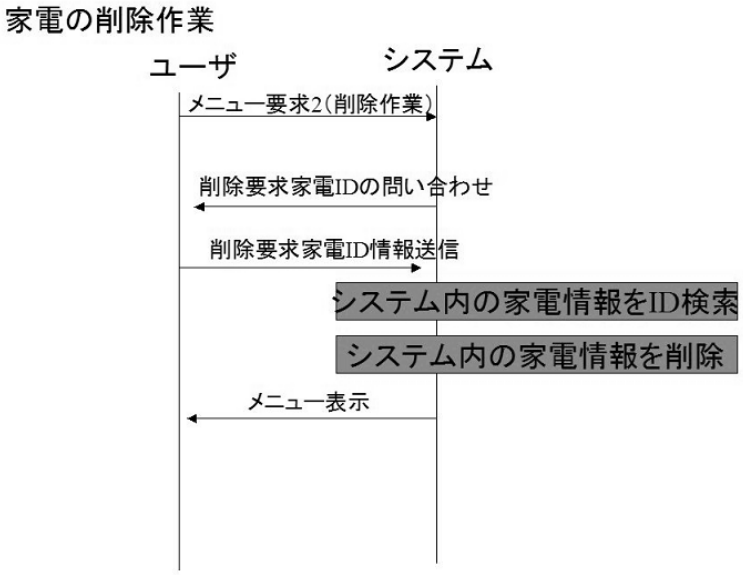

図 6.4: 家電の削除

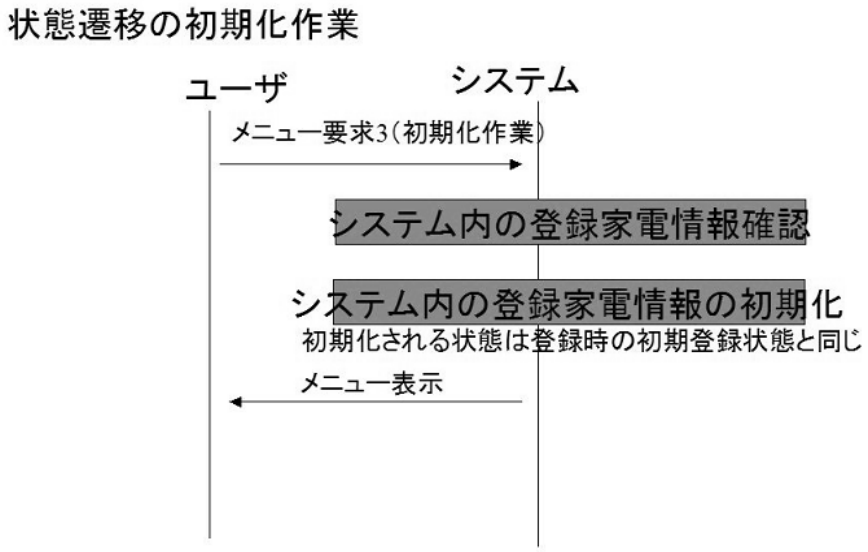

図 6.5: 状態の初期化

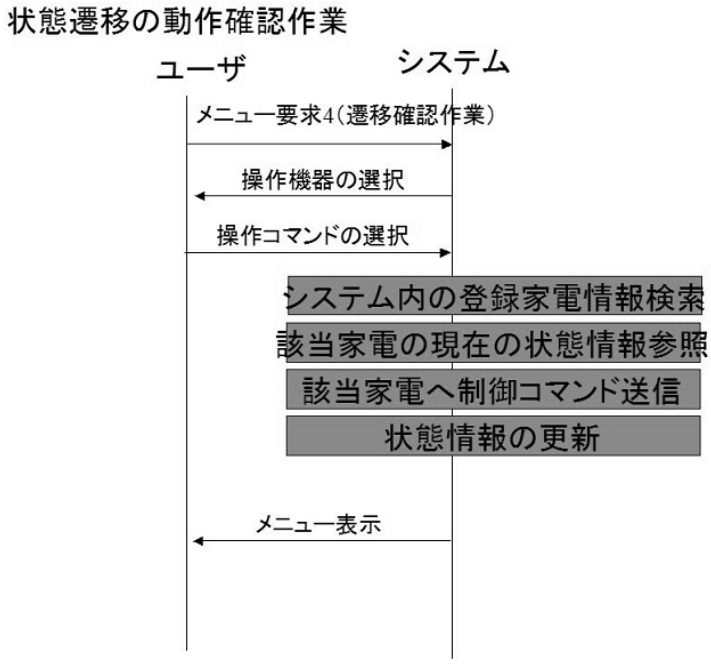

図 6.6: 状態遷移の確認

6.1.5　連携サービス記述

連携サービスを記述してファイルに書き出す。
連携サービス記述を行う流れを図 6.7 に示す。

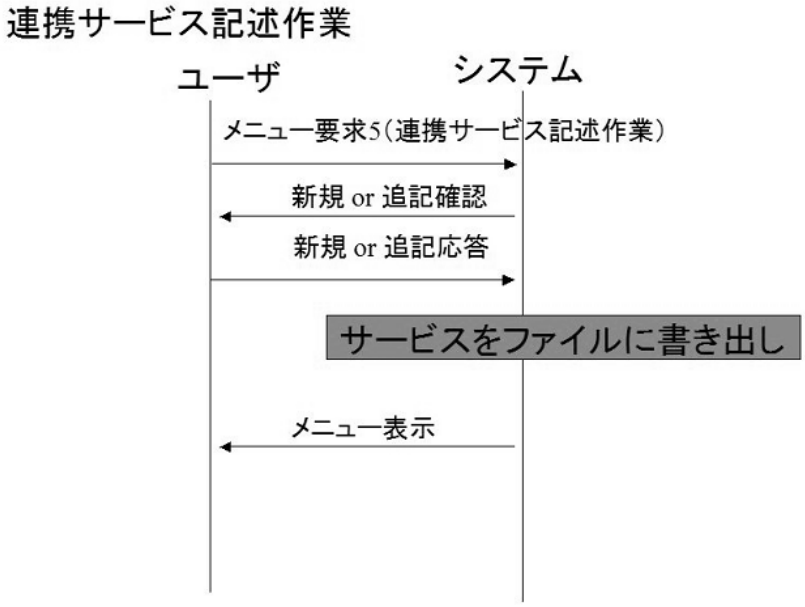

図 6.7: サービス記述

6.1.6　連携サービスパーザの動作

記述された連携サービスを受理できるかどうか調べるため、パーザを動作させる。
その動作の詳細は第 5 章に記述した。
連携サービスパーザの動作の流れを図 6.8 に示す。

6.1.7　現在の登録家電の確認

現在データベースに登録されている家電の様々な状態を確認する。
登録家電の確認を行う流れを図 6.9 に示す。

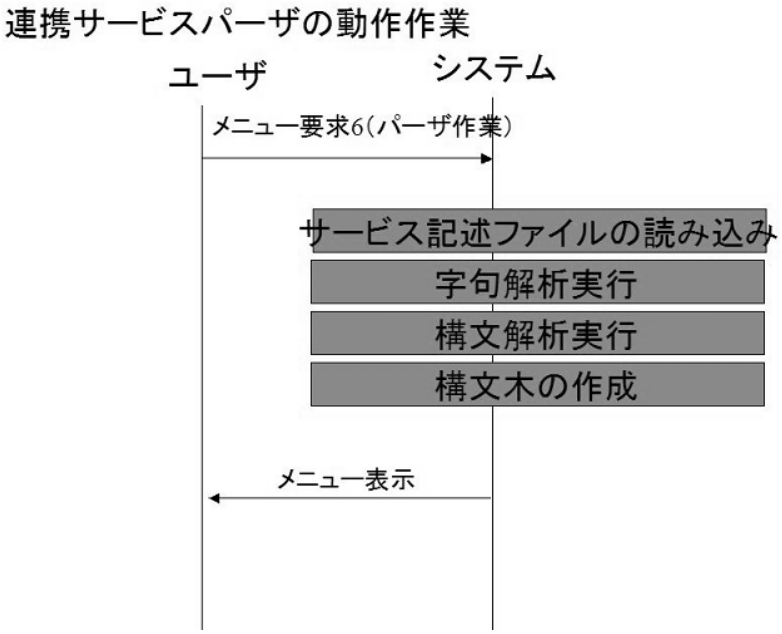

図 6.8: 連携サービスパーザ

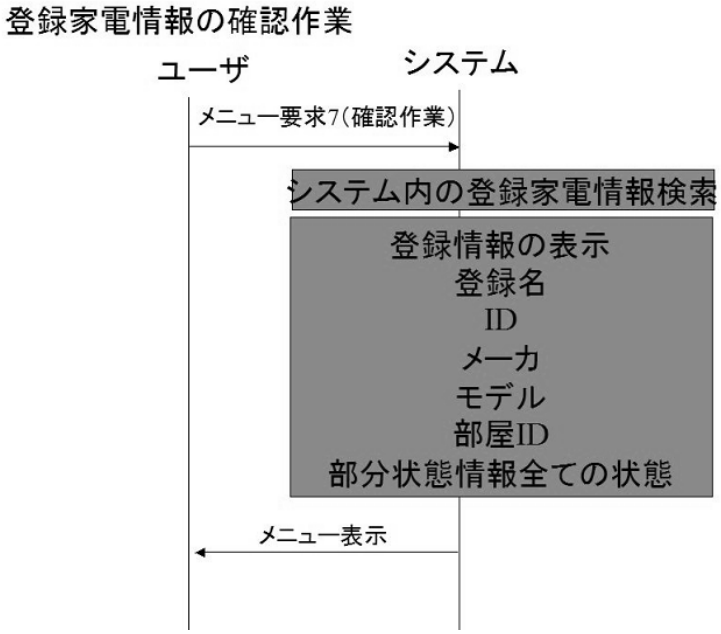

図 6.9: 登録家電情報確認

6.1.8　連携サービスの定義（初期登録サービス）

定義されている連携サービスを動作させる。

ユーザ定義の連携サービス以外に予め初期登録されている連携サービスを動作させる際に使用する。初期登録されている連携サービスとしては、照明、TV、VTR を用いたサービスがある。

初期登録サービス動作の流れを図 6.10 に示す。

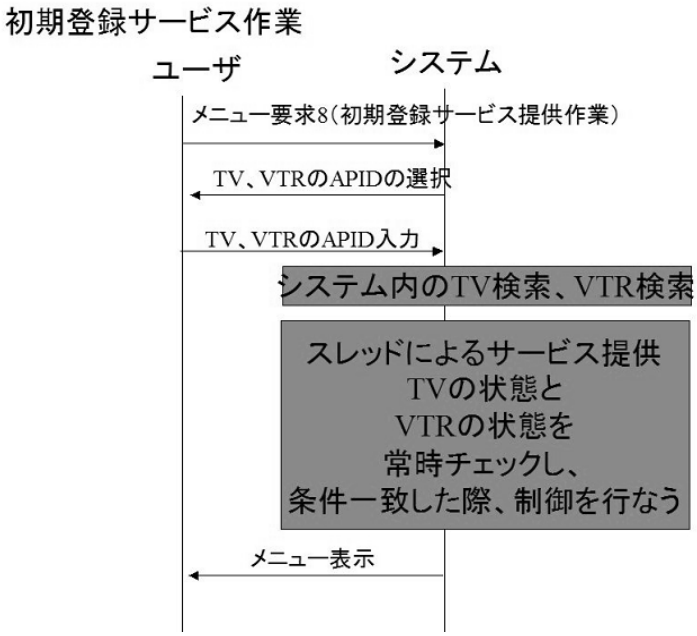

図 6.10: 初期サービス

6.1.9　連携サービススタート

記述した連携サービスを動作させる。

連携サービス開始の流れを図 6.11 に示す。

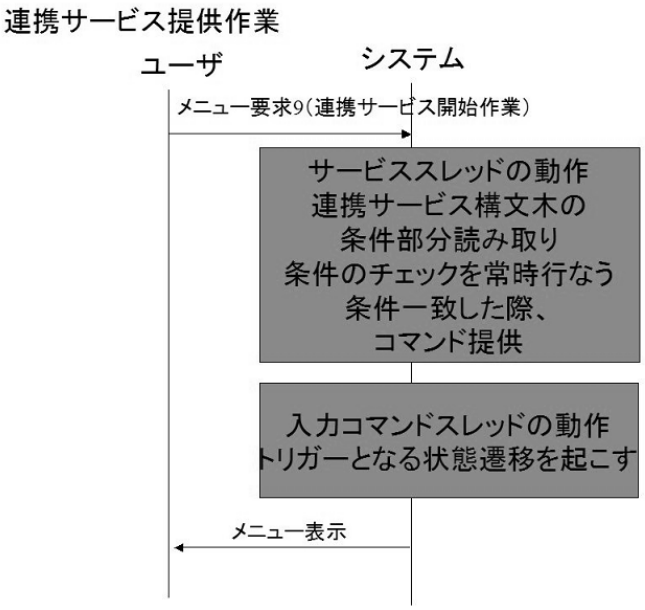

図 6.11: 連携サービス開始

6.2 システム内のテーブル情報

システム内で持つテーブル情報を表6.2-表6.9に示す。

従来の単体サービス時の情報に加えて、機器と機器の接続性や位置関係といった情報も連携サービスを提供する上で必要になってくる。

表 6.2: 家電基本情報

AP_ID	機器固有のID
AP_TYPE	機器のタイプ（ＴＶ、ＶＴＲなど）
AP_MAKER	家電のメーカ
AP_MODEL	機器のモデル(Aquosなど)
AP_NICK	家電のニックネーム（記述の際にわかりやすくするため）
ROOM_ID	家電の存在する部屋のID
LOCATE	家電の位置情報
NUM_COM	接続している家電の数
CONNECT	接続している家電の情報

図 6.12: AP_TYPE 一覧

家電タイプ	番号
TV	1
VTR	2
スピーカー	3
照明	4
エアコン	5
扇風機	6
窓	7
カーテン	8
扉	9
時計	10
携帯電話	11

図 6.13: SENSOR_TYPE 一覧

センサのタイプ	番号
赤外線	1
カメラ	2
温度	3
湿度	4
人感	5
画像	6
照度	7
タグ	8

表 6.3: センサ基本情報

SENSOR_ID	センサの ID
SENSOR_TYPE	センサのタイプ
SENSOR_VALUE	センサの現状値
AP_ID	センサの所属する家電 ID

表 6.4: 環境基本情報

ROOM_ID	部屋の ID
IN_TEMP	室内の温度
OUT_TEMP	室外の温度
IN_HUMID	室内の湿度
OUT_HUMID	室外の温度
IN_BRIGHT	室内の照度
OUT_BRIGHT	室外の照度
USER	ユーザの存在フラグ
USER_LOCATE	ユーザの位置情報

表 6.5: ボタン基本情報

BUTTON_ID	ボタンの ID
BUTTON_VALUE	ボタンの現状値

表 6.6: タイマー基本情報

TIMER_ID	タイマーの ID
TIMER_VALUE	タイマーの現状値 （使用されていない場合は-1(UNUSED)）

表 6.7: 部分状態情報

POWER	電源
CHANEL	チャンネル
VOL	音量
AUX	ライン入力
SOUND	音声タイプ
PLAY_REC	再生状態
REC	録画状態
BRIGHT	照度状態
OPEN_CLOSE	開閉状態
IN_OUT	入退出状態
MODE	運転モード
ANGLE	風向き
WIND	風量

表 6.8: 機器毎の状態情報

家電タイプ	部分状態情報
TV	POWER, CHANEL, VOL, AUX, SOUND
VTR	POWER, CHANEL , VOL, AUX, SOUND, PLAY_REC, REC
スピーカー	POWER, VOL, SOUND
照明	POWER, BRIGHT
エアコン	POWER, MODE, ANGLE, WIND
扇風機	POWER, ANGLE, WIND
窓	OPEN_CLOSE
カーテン	OPEN_CLOSE
扉	OPEN_CLOSE, IN_OUT
時計	POWER
携帯電話	POWER

表 6.9: 部分状態情報の状態一覧

POWER	STATE_ON	機器の状態が電源 ON の状態
	STATE_OFF	機器の状態が電源 OFF の状態
	STATE_CUT	機器の状態が電源カットの状態
CHANEL	STATE_CHx	機器のチャネルが x の状態
VOL	STATE_VOLx	機器の音量が x の状態
AUX	STATE_LINEx	機器のライン入力が x の状態
SOUND	STATE_MONO	機器の音声がモノラルタイプの状態
	STATE_ST	機器の音声がステレオタイプの状態
	STATE_5.1	機器の音声が 5.1ch タイプの状態
PLAY_REC	STATE_PLAY	機器が再生状態
	STATE_STOP	機器が停止状態
	STATE_FF	機器が早送り状態
	STATE_REW	機器が巻き戻し状態
	STATE_PLFF	機器が再生早送り状態
	STATE_PLREW	機器が再生巻き戻し状態
	STATE_REC	機器が録画状態
	STATE_PAUSE	機器が一時停止状態
REC	STATE_REC	機器が録画状態
BRIGHT	STATE_LOW	照度が低い（暗い）状態
	STATE_HIGH	照度が高い（明るい）状態
OPEN_CLOSE	STATE_OPEN	窓などの状態が OPEN の状態
	STATE_CLOSE	窓などの状態が CLOSE の状態
IN_OUT	STATE_IN	扉の状態が IN の状態
	STATE_OUT	扉の状態が OUT の状態
MODE	STATE_COOL	機器の運転状態が冷房の状態
	STATE_HEAT	機器の運転状態が暖房の状態
	STATE_DRY	機器の運転状態が除湿の状態
	STATE_AUTO	機器の運転状態が自動の状態
ANGLE	STATE_RIGHT90	機器の向き状態が右向き 90 度の状態
	STATE_RIGHT45	機器の向き状態が右向き 45 度の状態
	STATE_CENTER	機器の向き状態が真正面の状態
	STATE_LEFT45	機器の向き状態が左向き 45 度の状態
	STATE_LEFT90	機器の向き状態が左向き 90 度の状態
WIND	STATE_LOW	機器の風量が弱の状態
	STATE_MID	機器の風量が中の状態
	STATE_HIGH	機器の風量が強の状態
TIMER	STATE_TIMEx	タイマーの時間が x の状態 (ランダム時間の記述は x に RANDAM を記述)

6.3　システムの処理の流れ

連携サービスの記述から連携サービスの動作までのシステムの処理の流れを図 6.14 に示す。

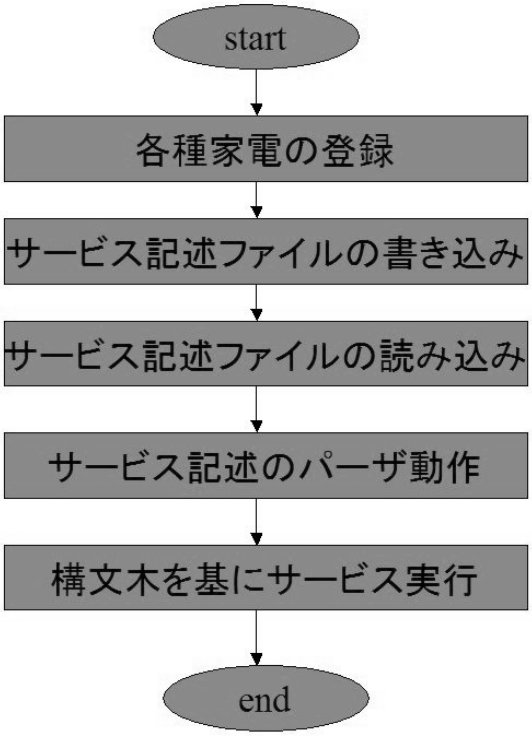

図 6.14: 処理の流れ

記述した連携サービスファイルを読み込み、その記述を受理できるかどうかを判断する。受理できれば、構文木が作成され、その構文木に基づいて、各処理を実行する。

システム内での動作としてシステム内のその処理の番号を加えた流れを以下に示す。

1. システム番号 1　家電の登録 (6.1.1 節参照)
2. システム番号 7　家電の登録状態の確認 (6.1.7 節参照)
3. システム番号 5　連携サービス記述 (6.1.5 節参照)
4. システム番号 6　連携サービスパーザの動作 (6.1.6 節参照)
5. システム番号 9　連携サービススタート (6.1.9 節参照)

6.4　問題点とその解決

- TV の買い換え時に起こる問題
  TV の買い換えによって、従来登録されていた TV がなくなった際にどう動くかは、

従来の登録されていた場所 (ID) に新たな TV を登録していれば、従来の機能は従来のままで使用できる。しかし、その買い換えた TV に新たな機能があった際、その機能を使用したい場合は、新たな登録作業が必要となる。

- TV が同じ部屋に 2 台あった際に起こる問題
  同じ部屋に 2 つの TV があり、TV を使った連携サービス記述がなされていたときにどっちの TV での動作となるかの判断は、記述した連携サービスにもよるが、連携サービス記述の部分で、TV の部屋の指定や接続条件などにより、特定されることがほとんどで、それでも特定されなかった場合には、直接的な指定、すなわち、AP_ID やニックネームなどの指定を記述の際にする必要がある。

- タイマーの扱い
  タイマーを扱う際に同じタイマーで 2 つの機器の待ちが存在することがあるので、タイマーは個別に情報を持ち、それぞれ ID により、同じ待ちが起こらない工夫を行う。

- プロセス管理の導入
  現在のシステムではスレッドを用いて連携サービスの動作を行っている。そのため、複数の連携サービスを動作させた際に、連携サービスの競合が発生し、正しく動作しない可能性が考えられる。そこで、スレッドでの管理からプロセスでの管理に変更することで、そのような競合発生における連携サービスの誤動作を防ぐことができると考えられる。

第7章　提案システムの動作確認とその評価

本章では、連携サービスパーザ、ホームネットワークシステムの動作確認と、その考察及び評価を行う。

動作確認は表 7.1 の環境で行った。

表 7.1: 動作環境

OS	LINUX
CPU	1.13GHz
メモリ	640MB
言語	C

7.1　連携サービスパーザの動作確認

連携サービスパーザの動作については、連携サービス記述から受理できるか否かの判断までの評価サンプルとして、いくつかの連携サービス記述について動作確認を行う。

- 連携サービス記述 1
  - 記述
    def service1
    if room1.tv.power = STATE_OFF &
    room1.vtr.power == STATE_ON &
    room1.vtr.connect ∋ room1.tv.apid
    then com (room1.tv.power , STATE_ON) &
    com (room1.tv.ch , STATE_CHVIDEO)
  - 字句解析の結果

```
result of token analysis
 0: 0:
 1:10:def
 2:11:service1
 3:12:if
 4:15:room1.tv.power
 5:16:=
 6:18:STATE_OFF
 7:17:&
 8:15:room1.vtr.power
 9:16:==
10:18:STATE_ON
11:17:&
12:15:room1.vtr.connect
13:16:->
14:15:room1.tv.apid
15:13:then
16:14:com
17:19:(
18:15:room1.tv.power
19:21:,
20:18:STATE_ON
21:20:)
22:17:&
23:14:com
24:19:(
25:15:room1.tv.ch
26:21:,
27:18:STATE_CHVIDEO
28:20:)
```

1 行目はトークン番号を表し、2 行目はトークンの種類、3 行目にトークンの中身を表している。また、記述の際の ∋ は-> に置換されている。

トークンの種類の一覧は表 5.1 に示す。

– スタックの中身の動き

```
:10  0  0  0  0  0  0  0  0  0  0  0  effective stack  1
:32 11  0  0  0  0  0  0  0  0  0  0  effective stack  1
:32 12  0  0  0  0  0  0  0  0  0  0  effective stack  2
```

```
:32 12 15  0  0  0  0  0  0  0  0  0  effective stack  3
:32 12 15 16  0  0  0  0  0  0  0  0  effective stack  4
:32 12 30 16 18  0  0  0  0  0  0  0  effective stack  3
:32 12 30 17 18  0  0  0  0  0  0  0  effective stack  4
:32 12 30 17 15  0  0  0  0  0  0  0  effective stack  5
:32 12 30 17 15 16  0  0  0  0  0  0  effective stack  6
:32 12 30 17 30 16 18  0  0  0  0  0  effective stack  3
:32 12 30 17 30 16 18  0  0  0  0  0  effective stack  4
:32 12 30 17 15 16 18  0  0  0  0  0  effective stack  5
:32 12 30 17 15 16 18  0  0  0  0  0  effective stack  6
:32 12 30 17 30 16 15  0  0  0  0  0  effective stack  3
:32 12 30 13 30 16 15  0  0  0  0  0  effective stack  4
:32 12 30 13 14 16 15  0  0  0  0  0  effective stack  5
:32 12 30 13 14 19 15  0  0  0  0  0  effective stack  6
:32 12 30 13 14 19 15  0  0  0  0  0  effective stack  7
:32 12 30 13 14 19 15 21  0  0  0  0  effective stack  8
:32 12 30 13 14 19 15 21 18  0  0  0  effective stack  9
:32 12 30 13 31 19 15 21 18 20  0  0  effective stack  5
:32 12 30 13 31 17 15 21 18 20  0  0  effective stack  6
:32 12 30 13 31 17 14 21 18 20  0  0  effective stack  7
:32 12 30 13 31 17 14 19 18 20  0  0  effective stack  8
:32 12 30 13 31 17 14 19 15 20  0  0  effective stack  9
:32 12 30 13 31 17 14 19 15 21  0  0  effective stack 10
:32 12 30 13 31 17 14 19 15 21 18  0  effective stack 11
:34 33 30 13 31 17 31 19 15 21 18 20  effectivestack  1
```

– 構文木の表示

TreeNo は木のノード番号を表し、parent はそのノードの親となるノード番号、reft、right はそのノードの子供となるノードの番号、type はそのノードの種類を表している。

その種類は先ほどの表 5.1 と同様である。

```
Show Tree
TreeNo parent reft right type
    0      0    0     0    0
    1      3    0     0   10
    2      3    0     0   11
    3      0    1     2   32
    4      0    0     0   12
    5      8    0     0   15
    6      8    0     0   16
    7      8    0     0   18
    8      0    5     6   31
    9      0    0     0   17
```

10	13	0	0	15
11	14	0	0	16
12	14	0	0	18
13	14	10	11	31
14	0	11	12	30
15	0	0	0	17
16	19	0	0	15
17	20	0	0	16
18	20	0	0	15
19	20	16	17	30
20	0	17	18	30
21	0	0	0	13
22	28	0	0	14
23	28	0	0	19
24	28	0	0	15
25	28	0	0	21
26	28	0	0	18
27	28	0	0	20
28	0	22	23	31
29	0	0	0	17
30	36	0	0	14
31	36	0	0	19
32	36	0	0	15
33	36	0	0	21
34	38	0	0	18
35	38	0	0	20
36	38	30	31	31
37	39	34	35	31
38	39	34	35	33
39	0	37	38	34

– 構文木の図

構文木を図 7.1 に示す。

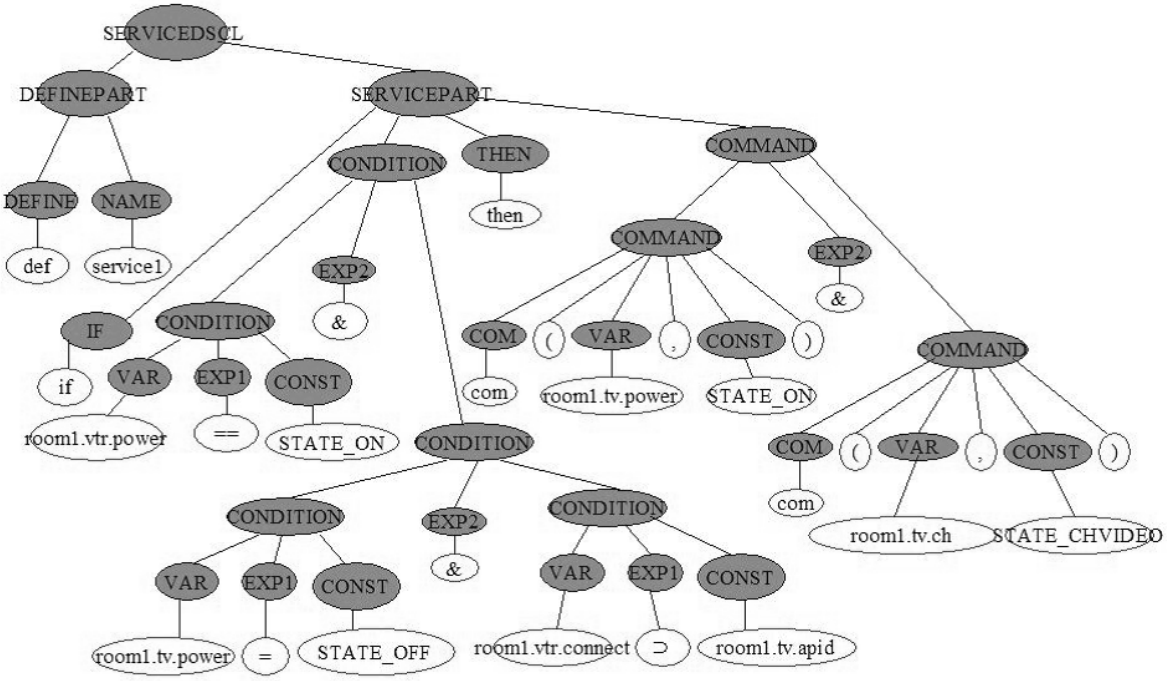
SERVICEDSCL
DEFINEPART
SERVICEPART
DEFINE
NAME
def
service1
CONDITION
THEN
then
COMMAND
IF
CONDITION
EXP2
&
if
VAR
EXP1
CONST
room1.vtr.power
==
STATE_ON
CONDITION
COMMAND
EXP2
&
COM
(
VAR
,
CONST
)
com
room1.tv.power
STATE_ON
COMMAND
COM
(
VAR
,
CONST
)
com
room1.tv.ch
STATE_CHVIDEO
CONDITION
EXP2
CONDITION
&
VAR
EXP1
CONST
room1.tv.power
=
STATE_OFF
VAR
EXP1
CONST
room1.vtr.connect
⊃
room1.tv.apid

図 7.1: 構文木 1

- 連携サービス記述 2
  - 記述

```
def service2
if ghb.state == STATE_ON
then com ( room1.tv.power , STATE_ON ) &
com ( room1.light.power , STATE_ON ) &
com ( room1.aircon.power , STATE_ON )
```

  - 字句解析の結果
    字句解析結果を表 7.2 に示す。
  - 構文木の図
    構文木を図 7.2 に示す。

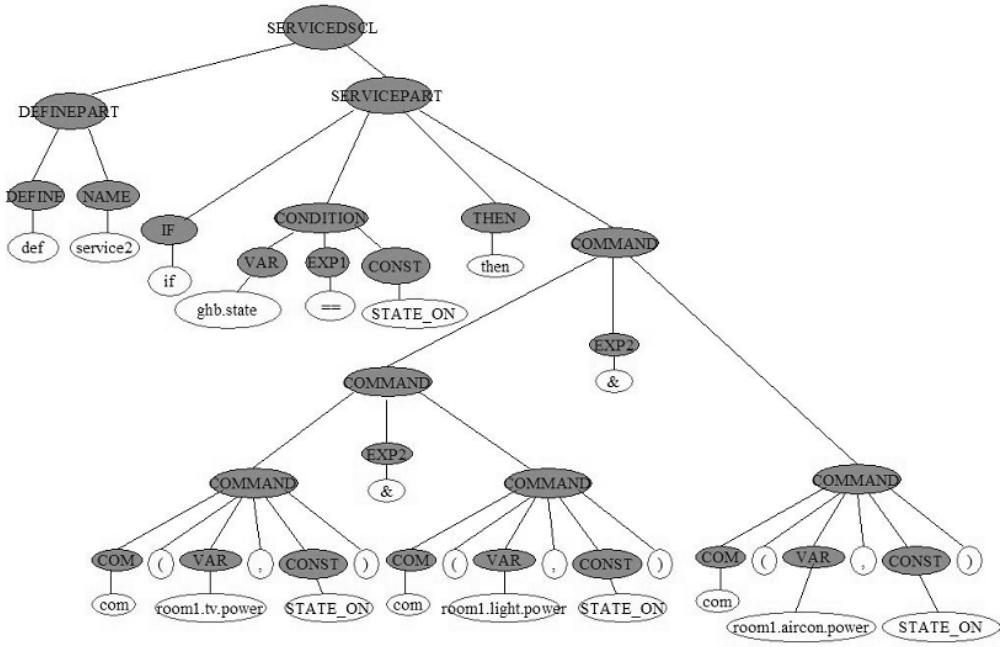

図 7.2: 構文木 2

表 7.2: 字句解析の結果 2

字句タイプ	記述
10	def
11	service2
12	if
15	ghb.state
16	==
18	STATE_ON
13	then
14	com
19	(
15	room1.tv.power
21	,
18	STATE_ON
20	)
17	&
14	com
19	(
15	room1.light.power
21	,
18	STATE_ON
20	)
17	&
14	com
19	(
15	room1.aircon.power
21	,
18	STATE_ON
20	)

- 連携サービス記述 3
  - 記述

```
def service3
if room1.light.power == STATE_OFF & timer1.value = STATE_UNUSED
then com ( timer1.value , STATE_TIME30 )
if timer1.vale == STATE_TIME0
then com ( room1.tv.power , STATE_OFF )
```

  - 字句解析の結果
    字句解析結果を表 7.3 に示す。
  - 構文木の図
    構文木を図 7.3 に示す。

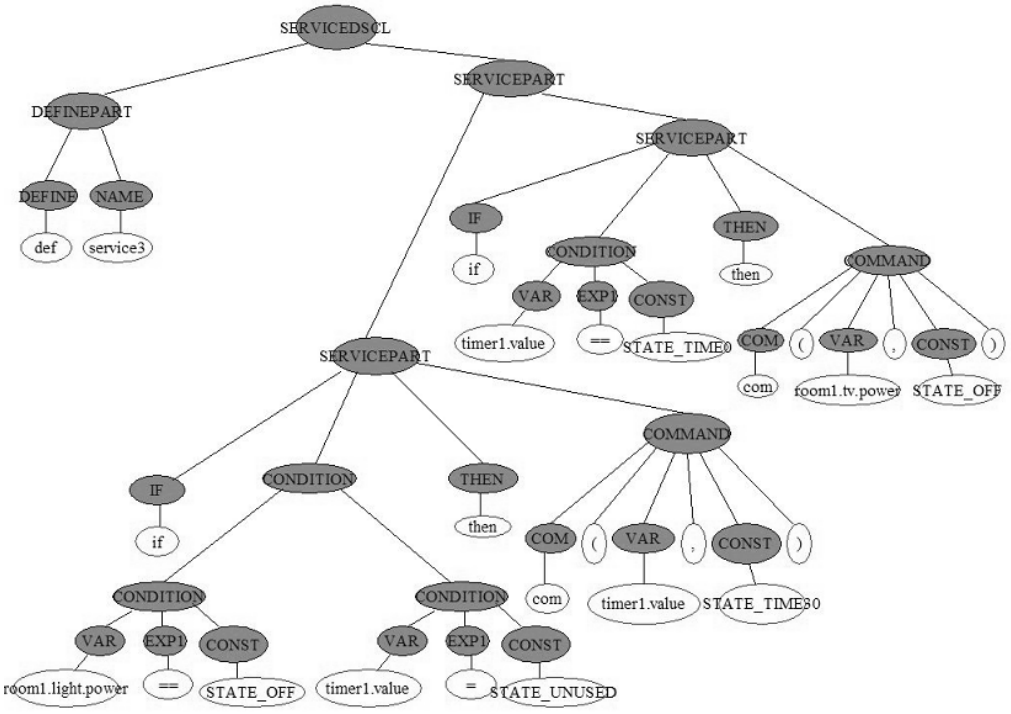

図 7.3: 構文木 3

表 7.3: 字句解析の結果 3

字句タイプ	記述
10	def
11	service3
12	if
15	room1.light.power
16	==
18	STATE_OFF
17	&
15	timer1.value
16	=
18	STATE_UNUSED
13	then
14	com
19	(
15	timer1.value
21	,
18	STATE_TIME30
20	)
12	if
15	timer1.vale
16	==
18	STATE_TIME0
13	then
14	com
19	(
15	room1.tv.power
21	,
18	STATE_OFF
20	)

- 連携サービス記述 4
  - 記述

```
def service4
if room1.bright.value == STATE_LOW
then com ( room1.light.power , STATE_ON )
```

  - 字句解析の結果
    字句解析結果を表 7.4 に示す。

表 7.4: 字句解析の結果 4

字句タイプ	記述
10	def
11	service4
12	if
15	room1.bright.value
16	==
18	STATE_LOW
13	then
14	com
19	(
15	room1.light.power
21	,
18	STATE_ON
20	)

  - 構文木の図
    構文木を図 7.4 に示す。

- 連携サービス記述 5
  - 記述

```
def service5
if time.value = STATE_TIME0730
then com ( room1.tv.power , STATE_ON ) &
com ( room1.right.power , STATE_ON )
```

  - 字句解析の結果
    字句解析結果を表 7.5 に示す。

表 7.5: 字句解析の結果 5

字句タイプ	記述
10	def
11	service5
12	if
15	time.value
16	=
18	STATE_TIME0730
13	then
14	com
19	(
15	room1.tv.power
21	,
18	STATE_ON
20	)
17	&
14	com
19	(
15	room1.right.power
21	,
18	STATE_ON
20	)

  - 構文木の図
    構文木を図 7.5 に示す。

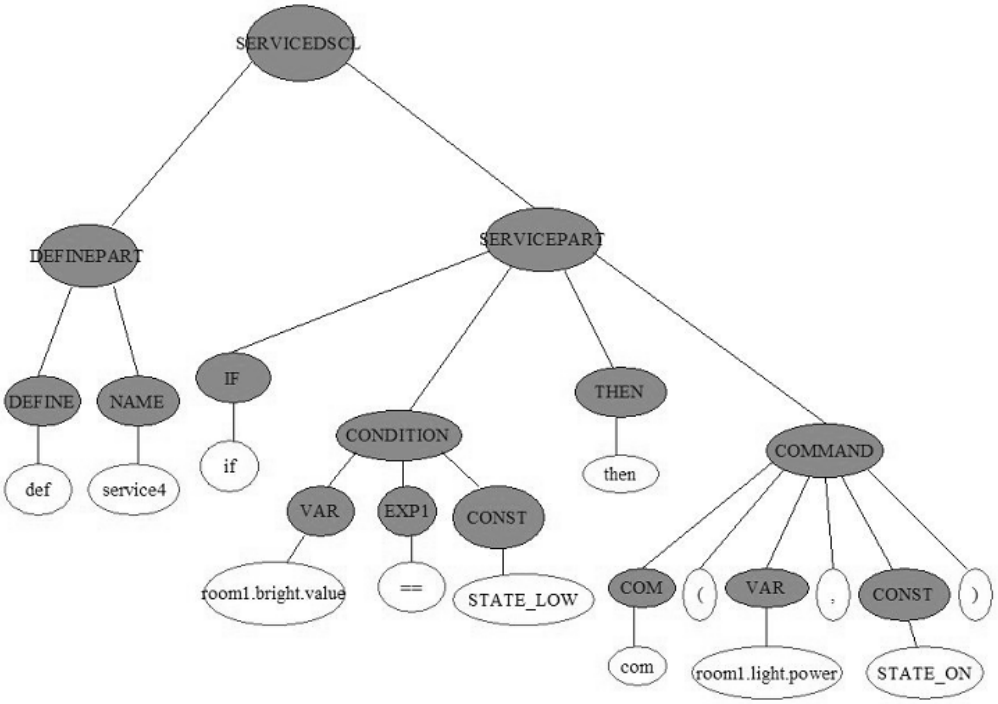

図 7.4: 構文木 4

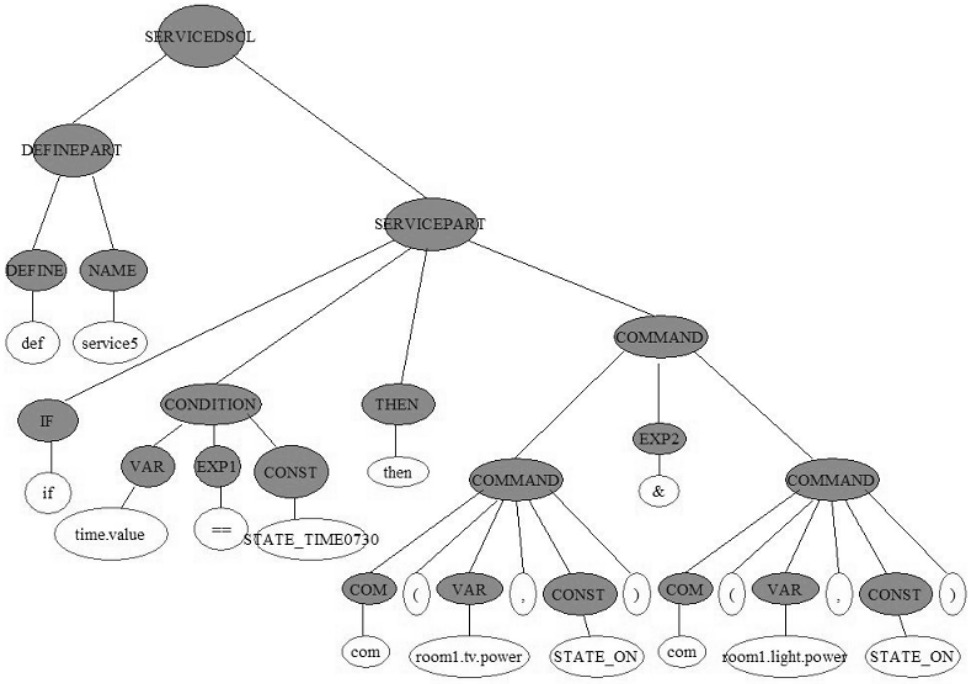

図 7.5: 構文木 5

実行結果の考察

記述言語の設計に基づいた記述が受理され、木が作られていることを確認できる。また、記述の際にどこかにエラーがあった場合、連携サービスパーザは字句解析を行い、その後のスタックの動作を確認でき、記述は受理されず、木の作成も行われなかった。

この連携サービスパーザは記述が記述言語仕様に合っているかを調べるものであり、そのサービス記述の意味までは理解していない。そのため、記述がサービス内容的におかしな記述であっても言語仕様を満たしている限り受理され、構文木が作成される。例えば、条件部に tv.state $\neq$ tv.state といった記述があった際に、その部分は全く意味をなさない。しかし、言語仕様は合っているため木が作成され、そのサービス動作時に条件部分の一致が起こらないため、サービス実行は永久に行われない。

7.2 ホームネットワークシステムの動作確認

ホームネットワークシステムの動作については、システム動作のシミュレーションを行い確認する。実際の機器を使用した動作実験ではなく、ホームサーバ内で管理している状態情報の確認を行う作業である。その方法は、連携サービスの記述や家電の登録を行い、連携サービスパーザを動作させ、その結果、システム内で状態遷移が行われていることを確認する。

今回、システム内に登録した家電は TV と VTR で、連携サービスとして VTR の電源 ON により TV の電源を ON かつ CH を VIDEO というサービスを動作させ、その動作を確認した。確認の際は連携サービス開始により、スレッドを用いた連携サービスの動作が行われ、擬似的な VTR や TV の操作により、そのそれぞれの操作を行う前後に登録家電の現在の状態の確認を行うことで、サービスの提供が行われているかを確認し、VTR を OFF から ON にした際に TV が OFF から ON、CH が VIDEO になることを確認した。

以下にその各種データと流れを示す。

システムを起動し TV と VTR を登録する。

その登録情報を図 7.6、図 7.7 に示す。

図 7.6: TV 登録情報

id	1
maker	Sony
roomid	1
power	STATE_OFF
channel	STATE_CH1
vol	STATE_VOL0
aux	STATE_LINE0
sensorid	1
sensortype	Ir

図 7.7: VTR 登録情報

id	2
maker	Sony
roomid	1
power	STATE_OFF
channel	STATE_CH1
vol	STATE_VOL0
aux	STATE_LINE0
player_recorder	STATE_STOP
sensorid	2
sensortype	Ir

次に連携サービスを記述し、その記述ファイルを連携サービスパーサに通す。その後、連携サービスを動作させ、家電機器に擬似的なコマンド送信を行い状態遷移を確認する。

疑似コマンド送信は以下のように行った。

```
Input Test Appliance(1:tv,2:vtr):2
Please Input Command (0:compower,1:hardlink,2:CHUP,3:CHDOWN,4:CHVIDEO): 0
```

VTR に対して電源を入れる。

動作後の状態を図 7.8、図 7.9 に示す。

図 7.8: TV 登録情報

id	1
maker	Sony
roomid	1
power	STATE_ON
channel	STATE_CHVIDEO
vol	STATE_VOL0
aux	STATE_LINE0
sensorid	1
sensortype	Ir

図 7.9: VTR 登録情報

id	2
maker	Sony
roomid	1
power	STATE_ON
channel	STATE_CH1
vol	STATE_VOL0
aux	STATE_LINE0
player_recorder	STATE_STOP
sensorid	2
sensortype	Ir

このとき VTR の電源 OFF 作業により、TV の電源 ON かつ CH を VIDEO にする連携サービスが動作し、状態が遷移していることがわかる。

今回、擬似的に確認した動作は、コマンド制御の実装により、実際の機器を動作させることができると考えられる。

7.3 考察と評価

本研究における目的である連携サービスの提供が、記述言語の作成、連携サービスパーザの作成、ホームネットワークシステムの作成により可能となった。

また、従来の連携サービスは情報家電のボタンによる連携サービスのみであったが、それが、提案したシステムを用いることにより、レガシーデバイスでのボタンによる連携サービスの提供が可能となり、さらに、機器の状態変化からの連携サービスの提供、タイマーを用いた連携サービスの提供が可能となった。

連携サービスの提供は記述されたとおりの動作をするので、意味のない動作であるサービス記述も書くことができる。その動作の改善方法はサービス実行しているものを実際に確認し、異常に気付き、記述を変更するという方法で改善される。

第8章　今後の課題

今後の課題として以下のものが挙げられる。

8.1　登録時の機器の状態の対応

現在の機器登録は、初期状態に人手によって実際の機器を操作する必要がある。この部分は、センサなどにより、現在の登録しようとする機器の状態を検知し、それをシステム側で利用することで、人手による作業がなくなると考えられる。

8.2　記述者に対するサポート

現状では、記述者がシステム内の情報を把握しておき、それをもとに記述を行っている。記述には、システム内の情報把握が必須となっているため、今後、それらの情報を記述の際に何らかの形で明示しておく必要がある。

8.3　センサからの情報収集

現状では、センサからの情報収集の部分は実装されておらず、擬似的なセンサからの情報という形でセンサ情報を扱っている。今後、実際にセンサからの情報収集を行い、システムに追加する必要がある。

8.4　実際の機器の制御

現状では、本システムにレガシーデバイスホームネットワークシステムの家電の動作制御部分の実装は行っておらず、状態遷移の確認は実際の機器ではなく、システム内の情報情報で行っている。

8.5　インアクティブタイマーの実装

インアクティブタイマーとは、機器の状態変化が数分から数時間行われなかった際に行われる処理で、その部分は、機器の制御時に状態変化が起こったことを検知することで実現できると考えられる。そのため、実際の機器の制御の課題と共に、インアクティブタイマーの実装も課題となっている。

8.6　サービスの意味論

現状では、意味のないサービスを検知する術がない。今後、サービスの意味論として、そういったサービスを排除するといったことが必要になると考えられる。

8.7　サービスの競合検知

現状では、サービスの競合発生時に検知し、制御することはできない。今後、サービスの意味論と共に、この部分についても課題となる。

第9章　まとめ

本研究では、連携サービス提供のため、記述言語を考案し、その記述を解釈する連携サービスパーザの作成を行い、連携サービス提供システムとしてレガシーデバイスホームネットワークシステムをもとにシステム構築を行った。

第一に、連携サービス提供のため、機器の状態変化やボタン、イベントタイマー、スケジュールタイマーをトリガーとした連携サービスを考案し、2連携の状態変化をトリガーとしたものについては表の作成を行った。

第二に、それらの連携サービス記述のため、BNF 記法をもとにした構文解析ルールを作成し、連携サービス記述言語仕様を作成した。また、そのサービス記述言語を解釈する連携サービスパーザを作成した。

第三に、レガシーデバイスホームネットワークシステムをもとに連携サービス記述部分、連携サービス提供部分を追加したホームネットワークシステムを構築した。

最後に評価として、これらの連携サービスパーザ、連携サービス提供システムの動作確認を行い、連携サービスを提供できることを確認した。

本研究で、ホームネットワークにおいて既存のサービスであったボタンのサービスがレガシーデバイスでも使用でき、さらに、ボタン以外のサービスとして、家電機器の状態変化からのサービスや、タイマーを使用したサービスが提供可能になった。また、記述言語を作成したことにより、ユーザの嗜好に合わせたサービス提供が可能となった。

謝辞

本研究を進めるに当たり, 研究の方向性について指針を与えてくださり, また熱心なご指導, ご助言を頂きました丹康雄助教授に深く感謝致します. また, 本研究に関して非常に多くのご意見を頂きました助手の増井健司博士, 博士課程の中田潤也氏, 牧野義樹氏に心から感謝致します. さらにホームネットワーク班として互いに切磋琢磨して研究に励んできた増田耕一氏に感謝致します. そして, 励まし合いながら研究生活学生生活を過ごしてきた丹研究室の皆様に深く感謝致します. 最後に私を支えてくれた多くの友人, そして家族に感謝し, 謝辞と致します.

参考文献

[1] 石井 智康, Legacy Device を中心としたホームネットワークに関する研究, 北陸先端科学技術大学院大学情報科学研究科, 修士論文, 2003.3

[2] 松崎 寛之, レガシーデバイスによるホームネットワーク構築における機器の状態取得及び管理に関する研究, 北陸先端科学技術大学院大学情報科学研究科, 修士論文, 2005.3

[3] 浜本 賢一, ホームネットワーク規格との接続性を考慮した Legacy Device ホームネットワークの構築に関する研究, 北陸先端科学技術大学院大学情報科学研究科, 修士論文, 2005.3

[4] 出村 哲也, 丹 康雄, センサを用いたレガシーデバイスホームネットワークの連携サービスシステムの提案, 平成 17 年度電気関係学会北陸支部連合大会, 2005.9

[5] 丹康雄監修, 宅内情報通信・放送高度化フォーラム編, ユビキタス技術ホームネットワークと情報家電, オーム社, 2004

[6] 中田 育男, コンパイラ, オーム社, 2004

[7] 石田 綾, スモールコンパイラの制作で学ぶプログラムのしくみ, 技術評論社, 2004

[8] ECHONET CONSORTIUM, The ECHONET Specification Version 2.11, ECHONET CONSORTIUM, 2002

[9] DLNA, on-line available at http://www.dlna.org

[10] HAVi, on-line available at http://havi.org

[11] X-10 Technology and Resource Forum, on-line available at http://www.x10.com

[12] TOSHIBA CONSUMER MARKETING CORPORATION, on-line available at http://www3.toshiba.co.jp/feminity

[13] 石田 亨, 福本 理人, インタラクション設計言語Qの提案, 人工知能学会論文誌 17 巻 2 号, 2002

[14] 中西 英之, 石田 亨, 伊藤 英明, 福本 理人, 仮想都市空間シミュレータ FreeWalk/Q, SICE システムインテグレーション部門講演会 (SI2002), 2002

[15] 村上 陽平, 石田 亨, 川添 智幸, 菱山 玲子, インタラクション設計に基づくマルチエージェントシミュレーション, 人工知能学会論文誌 18 巻 5 号, 2003

付 録A　Syntax Diagram

A.1　字句解析

NUMBER

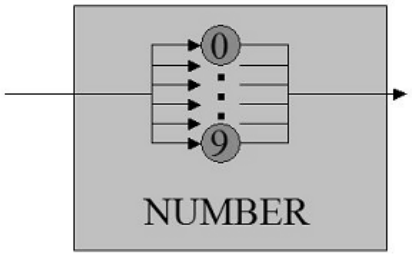

図 A.1: 数字

ALPHA

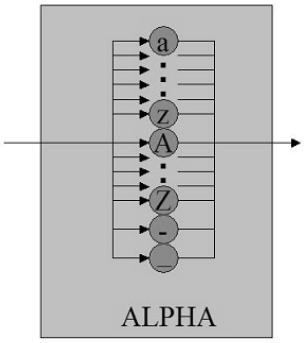

図 A.2: 英字

NAME

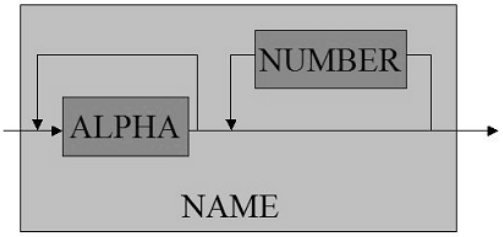

図 A.3: 名前

VAR

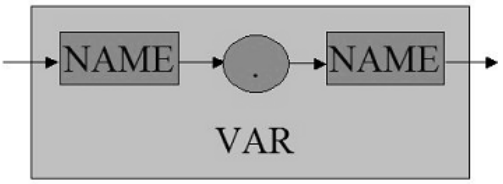

図 A.4: 変数

EXP

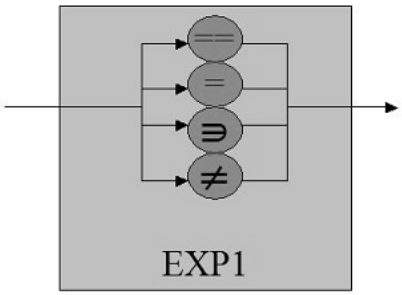

図 A.5: 条件式

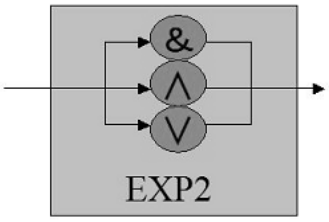

図 A.6: 接続式

CONST

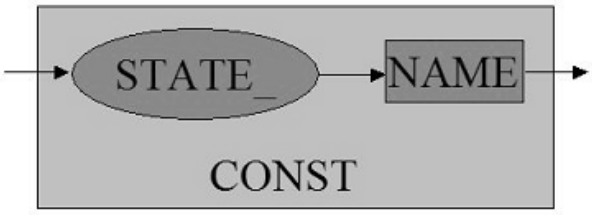

図 A.7: 定数

A.2 構文解析

サービス記述

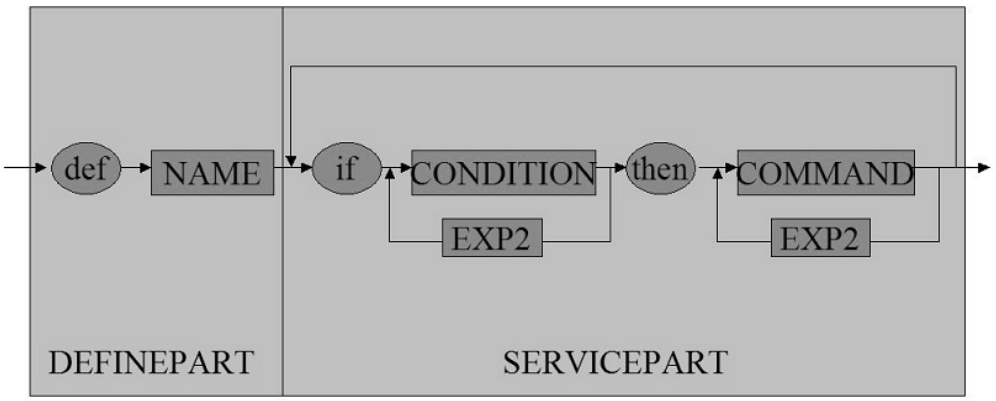

図 A.8: サービス記述

付 録B　状態遷移表

B.1　POWER

表 B.1: POWER の状態遷移表

状態 \ 入力	POWER	POWERLINK	time
STATE_ON	STATE_OFF	STATE_CUT	STATE_OFF
STATE_OFF	STATE_ON	STATE_CUT	
STATE_CUT		STATE_ON	

B.2　CHANEL

表 B.2: CHANEL の状態遷移表

状態 \ 入力	CHx	CHUP	CHDOWN
STATE_CH1	STATE_CHx	STATE_CH2	STATE_CHVIDEO
STATE_CH2	STATE_CHx	STATE_CH3	STATE_CH1
STATE_CH3	STATE_CHx	STATE_CH4	STATE_CH2
...	...	...	...
STATE_CH63	STATE_CHx	STATE_CHVIDEO	STATE_CH62
STATE_CHVIDEO	STATE_CHx	STATE_CH1	STATE_CH63

B.3 AUX

表 B.3: AUX の状態遷移表

状態 \ 入力	LINEx	LINE
STATE_LINE0	STATE_LINEx	STATE_LINE1
STATE_LINE1	STATE_LINEx	STATE_LINE2
STATE_LINE2	STATE_LINEx	STATE_LINE3
STATE_LINE3	STATE_LINEx	STATE_LINE4
STATE_LINE4	STATE_LINEx	STATE_LINE5
STATE_LINE5	STATE_LINEx	STATE_LINE0

B.4 VOL

表 B.4: VOL の状態遷移表

状態 \ 入力	VOLUP	VOLDOWN
STATE_VOL1	STATE_VOL2	
STATE_VOL2	STATE_VOL3	STATE_VOL1
STATE_VOL3	STATE_VOL4	STATE_VOL2
...	...	...
STATE_VOL139	STATE_VOL140	STATE_VOL138
STATE_VOL140		STATE_VOL139

B.5 PLAYER_RECORDER

表 B.5: PLAYER_RECORDER の状態遷移表

状態 \ 入力	POWER	PLAY	REW	FF	PAUSE	REC	STOP
STATE_STOP	STATE_OFFx	STATE_PLAY	STATE_REW	STATE_FF	STATE_PAUSE	STATE_REC	
STATE_PLAY	STATE_OFF		STATE_PLREW	STATE_PLFF	STATE_PAUSE	STATE_REC	STATE_STOP
STATE_REC	STATE_OFF				STATE_PAUSE		STATE_STOP
STATE_FF	STATE_OFF	STATE_PLAY	STATE_REW		STATE_PAUSE	STATE_REC	STATE_STOP
STATE_REW	STATE_OFF	STATE_PLAY		STATE_FF	STATE_PAUSE	STATE_REC	STATE_STOP
STATE_PLFF	STATE_OFF	STATE_PLAY		STATE_PLFF	STATE_PAUSE	STATE_REC	STATE_STOP
STATE_PLREW	STATE_OFF	STATE_PLAY	STATE_PLREW		STATE_PAUSE	STATE_REC	STATE_STOP
STATE_PAUSE	STATE_OFF	STATE_PLAY	STATE_PLREW	STATE_PLFF	STATE_PLAY	STATE_REC	STATE_STOP

B.6 OPEN_CLOSE

表 B.6: OPEN_CLOSE の状態遷移表

状態 \ 入力	OPEN	CLOSE
STATE_OPEN		STATE_CLOSE
STATE_CLOSE	STATE_OPEN	

B.7 MODE

表 B.7: MODE の状態遷移表

状態 \ 入力	CHANGEMODE	COOL	HEAT	DRY	AUTO
STATE_COOL	STATE_HEAT		STATE_HEAT	STATE_DRY	STATE_AUTO
STATE_HEAT	STATE_DRY	STATE_COOL		STATE_DRY	STATE_AUTO
STATE_DRY	STATE_AUTO	STATE_COOL	STATE_HEAT		STATE_AUTO
STATE_AUTO	STATE_COOL	STATE_COOL	STATE_HEAT	STATE_DRY	

B.8 ANGLE

表 B.8: ANGLE の状態遷移表

状態 \ 入力	CHANGEANGLE	RIGHT	LEFT
STATE_RIGHT90	STATE_RIGHT45		STATE_RIGHT45
STATE_RIGHT45	STATE_CENTER	STATE_RIGHT90	STATE_CENTER
STATE_CENTER	STATE_LEFT45	STATE_RIGHT45	STATE_LEFT45
STATE_LEFT45	STATE_LEFT90	STATE_CENTER	STATE_LEFT90
STATE_LEFT90	STATE_RIGHT90	STATE_LEFT45	

B.9 WIND

表 B.9: WIND の状態遷移表

状態 \ 入力	CHANGEWIND	HIGH	MID	LOW
STATE_HIGH	STATE_MID		STATE_MID	STATE_LOW
STATE_MID	STATE_LOW	STATE_HIGH		STATE_LOW
STATE_LOW	STATE_HIGH	STATE_HIGH	STATE_MID	

付 録C　仕様関数一覧

C.1　main

main

書式
main()
詳細
ホームネットワークシステムのメインの関数

Menu

書式
Menu()
詳細
システムのメニューの表示とその選択を行う関数

C.2　InputLDEntry

CheckAppid

書式
CheckAppid(apid,delflg,id)
詳細
家電 ID が使用されているかどうかの確認を行う関数

TransAppliance

書式
TransAppliance(aptype)
詳細
家電タイプの変換を行う関数

TransMaker

書式
TransAppliance(apmaker)
詳細
家電メーカーの変換を行う関数

CheckSensorid

書式
CheckSensorid(senid,delflg,id)
詳細
センサ ID が使用されているかどうかの確認を行う関数

TransSensor

書式
TransSensor(sensortype)
詳細
センサの変換を行う関数

EntryLD

書式
EntryLD(apid,aptype,apmaker,model,roomid)
詳細
家電の登録を行う関数

EntrySensor

書式
EntryLD(apid,sensorid,sensortype)
詳細
センサの登録を行う関数

InputLDEntry

書式
InputLDEntry()

詳細
家電、センサの登録処理を行う関数
ファイルからの登録情報読み取りなどはこの関数内で行われる。

C.3 DeleteLDEntry

DeleteLDEntry

書式
DeleteLDEntry()
詳細
登録されている家電を削除する関数

C.4 MakeStateTable

MakeStatetable

書式
MakeStatetable()
詳細
状態遷移表の初期化作業を行う関数

ChStateTable

書式
ChStateTable(state,command)
詳細
チャネルの状態遷移を管理する関数

VolStateTable

書式
VolStateTable(state,command)
詳細
音量の状態遷移を管理する関数

IllumStateTable

書式
IllumStateTable(state,command)
詳細
照明の状態遷移を管理する関数

PlayRecStateTable

書式
PlayRecStateTable(state,command)
詳細
プレイヤーレコーダーの状態遷移を管理する関数

ChStateTable

書式
PowStateTable(state,command)
詳細
電源の状態遷移を管理する関数

InitStateTable

書式
InitStateTable(state,command)
詳細
各種家電の部分状態情報の状態遷移を行う関数

TestStateTable

書式
TestStatetable
詳細
疑似コマンドによる状態遷移の確認を行う関数

C.5 WriteHarmony

WriteHarmony

書式
WriteHarmony()
詳細
テキストファイルとしてサービス記述を書き出す関数

HarmonyCheck

書式
HarmonyCheck()
詳細
サービスファイルを読み、記述を調べる関数

C.6 ShowLDEntry

PrintMaker

書式
PrintMaker(maker)
詳細
メーカー識別子からメーカー名を返す関数

PrintSensor

書式
PrintSensor(sensortype)
詳細
センサタイプ識別子からセンサタイプ名を返す関数

PrintName

書式
PrintName(state)
詳細
機器の状態情報識別子から状態情報を返す関数

ShowLDEntry

書式
ShowLDEntry()
詳細
登録家電の各種情報を表示する関数

C.7 DefineHarmony

DefineHarmony

書式
DefineHarmony()
詳細
初期登録連携サービス（TV と VTR）の動作関数

C.8 Harmony

Start

書式
Start()
詳細
連携サービスを動作させる関数

contorol

書式
void control()
詳細
疑似コマンドを送信し、機器の状態の変化を確認する関数

coodinate0

書式
coodinate0()
詳細
連携サービスの一つを動作させる関数

C.9 Parser

Lexer

書式
Lexer()
詳細
字句解析を行う関数

Parser

書式
parser()
詳細
構文解析を行う関数
解析結果を返す

MakeTree

書式
MakeTree()
詳細
構文木を作成する関数

ShowTree

書式
ShowTree()
詳細
構文木を表示する関数