卒業論文　　2002年度 (平成14年度)

アプリケーションの動的再構成を実現する基盤ソフトウェアの研究

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

氏名：高橋 元

卒業論文要旨　　2002年度 (平成14年度)

アプリケーションの動的再構成を実現する基盤ソフトウェアの研究

論文要旨

現在，ネットワーク接続性を持つ家電器機 (情報家電) の協調動作が可能となった．また，センサネットワーク等により家屋内の様々なものの状態及び位置情報が取得可能になったことで，分散オブジェクト間のより多様な協調動作が可能となった．本論文では，このような計算環境をインタラクティブスペースと呼ぶ．インタラクティブスペースでは、情報家電やPCで稼動するアプリケーションなど様々な分散オブジェクト間の協調が可能である．分散オブジェクトの協調動作を実現するアプリケーションを記述する場合，分散オブジェクト間の協調動作 (以下振る舞いと呼ぶ) が静的に定義されてしまう．そのため，新規デバイスの参加などユーザが置かれる状況の変更に伴う振る舞いの変更要求に柔軟に対処できない．本論文では，この問題に対処するために2つの側面からアプローチする．1つ目には，分散オブジェクトの機能定義と分散オブジェクト間の協調動作を定義を分離する．協調動作定義を再定義することで，協調動作を実行時に再定義可能なユーザの要求に適応的な環境を実現する．2つ目には，インタラクティブスペースで生活するユーザ自身が協調動作を再定義するエンドユーザプログラミング環境を想定し，協調動作定義をよりユーザの認知対象に近い記述モデルで実現する．

本研究では，この2つの問題に対処するために，分散オブジェクトにより構成される環境をユーザの直接認知可能な離散状態の集合して表現した．また，分散オブジェクト間の協調動作を離散状態の遷移規則として表現する言語 Interactive Space Programming Language(ISPL) を提案し，Java RMI を利用して ISPL のエバリュエータとパーサの実装及び評価を行った．

キーワード：

1 分散オブジェクト　2 振る舞い記述　3 状態遷移
4 アプリケーションの動的再構成

慶應義塾大学　環境情報学部
高橋 元

Abstract of Bachelor's Thesis

A Research of Dynamic Composition of Distributed Application

Academic Year 2002

Summary

At present, collaborative behavior between home networked appliances have become possible. Also, sensor networks within the household have enabled status and location information to be obtained allowing various collaborative behavior possible.

In this paper, we call such computing environment Interactive Space. In Interactive Space, collaboration between objects such as applications which run on home networked appliances and PCs is possible. When descripting applications which realize collaborative behavior of distributed objects, behavior is defined statically. Therefor, request for change of behavior due to change in status of the user's surrounding environment such as participation of new devices can not be dealt with flexibly. We approach the above problems from two aspects. First, we separate the definition of distributed object functions and collaborative behavior of distributed objects. By redefining collaborative behavior, an environment adaptive to redefinition of user requests at runtime is possible. Second, we assume an end user programming environment where the user exisiting in the Interactive Space redefines collaborative behavior. This realizes collaborative behavior definition with a description model more user recognizable.

In this research, to solve these problems, the environment structured from distributed objects is described as a collective of discrete status which the user can directly recognize. Interactive Space Programming Language(ISPL), which describes the transition rule of collaborative behavior between distributed objects is proposed. Implementations and evaluations of an evaluator and parser of ISPL using Java RMI are also presented.

Keyworkds:

1 Distributed Object 2 Behavior Protocol Description 3 State Transition

4 Dynamic Application Reconstruction

Keio University Faculty of Environmental Information

TAKAHASHI Gen

目 次

図 目 次

表 目 次

第 1 章

序論

1.1 本研究の背景ーユビキタス計算環境ー

本節では，本研究で前堤となる背景として，ユビキタス計算環境について述べる．近年のホームネットワーク環境では情報家電やスマートファーニチャ[13] が登場し，これらは高度な計算処理能力とネットワーク接続性を有している．これにより，情報家電等の我々の身の回りに存在するものが遠隔から制御可能となった．これら情報家電や PC 上のソフトウェアなど多様な分散コンポーネントの相互接続を可能とする通信基盤ミドルウェアに関する研究として，CORBA[8]，Jini[10]，JavaSpaces[9]，JEcho[12]，VNA[4, 5]，DRAGON[2, 3] などが挙げられる．これら通信基盤技術の研究開発によりホームネットワーク及びインターネットにおける多様な分散コンポーネントの相互接続が実現され，分散コンポーネント間の様々な機能的な協調動作が実現している．

1.2 本研究の問題意識

前節で述べたように，従来から我々のまわりに存在する情報家電や家具等がネットワーク接続性を持った．それらの情報家電は分散コンポーネントとして表現され，分散コンポーネントの集合が相互に相互機能を利用し，一つのアプリケーションを構成することが可能となった．このような多様な協調が可能であるユビキタス計算環境を空間を本研究では，インタラクティブスペースと呼ぶ．このようなインタラクティブスペースでは，情報家電や PC で稼動するアプリケーション等の様々なオブジェクト間の協調が可能なインタラクティブスペースにおいて，オブジェクトの協調動作を実現するアプリケーションを記述する場合，オブジェクト間の協調動作 (以下振る舞いと呼ぶ) 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振る舞いの変更要求に柔軟に対処することができない．

振る舞いの動的な変更の必要性は，ユーザの置かれる環境の変化に即応して移り変わるユーザの要求とインタラクティブスペースのオブジェクト間の協調動作の整合性を維持する必要から生じる．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情報家電協調動作アプリケーションを考える．新規に CD プレーヤが

ホームネットワークに参加した場合，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振る舞いの動的な変更要求の実現は，振る舞いの定義が静的であると実現できない．

また，インタラクティブスペースにおいて，振る舞いの動的な変更要求は，そのほとんどがそこで生活するユーザから直接生じる．従来のソフトウェア開発モデルでは，ソフトウェアの記述は，ソフトウェアベンダが行い，ユーザはソフトウェアを記述することはなかった．しかし，インタラクティブスペースのような多様な協調動作が可能となる空間では，多様な分散オブジェクトが協調動作を行うアプリケーション開発に従来のソフトウェアモデルを適応すると，ユーザは要求を満たす協調動作アプリケーションを得るまでに，時間的，金銭的な様々なコストがかかる．従って，インタラクティブスペースでは，エンドユーザプログラミング環境を実現するのが望ましい．

1.3 本研究の目的と概要

本研究では，インタラクティブスペースにおけるユーザが自由に要求に応じたオブジェクト間の協調動作を定義可能な計算環境を実現するために，2つの問題を扱う．1つ目には，分散オブジェクトの機能定義と分散オブジェクト間の協調動作を定義を分離し，協調動作定義を再定義することで，協調動作を実行時に再定義可能なユーザの要求に適応的な環境を実現する．2つ目には，インタラクティブスペースで生活するユーザ自身が協調動作を再定義するエンドユーザプログラミング環境を想定し，協調動作定義をよりユーザの認知対象に近い記述モデルで実現することである．本研究では，この2つの問題に対処するために，分散オブジェクトにより構成される環境をユーザの直接認知可能な離散状態の集合して表現し，分散オブジェクト間の協調動作を離散状態の遷移規則として表現する言語 Interactice Space Programming Language(ISPL) を提案する．離散状態の遷移規則として協調動作を表現することは，自然言語による協調動作の表現と近い．従って，従来の計算機に近い表現に比べ，ユーザの認知モデルに近い表現である．さらに，ISPLの実行環境としてJava RMIを利用したASARの実装及び基本性能の評価を行う．

1.4 本研究の構成

本論文では，まず2章で動的アプリケーション再構成を実現するアプリケーション記述系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，ASARの特徴と設計を述べ，5章でASARの実装に付いて述べ，第6章で基本性能の測定を行う．7章でASARと他の関連研究との比較を行いASARの優位性を述べ，最後に8章で本論文をまとめる．

第 2 章

インタラクティブスペースと振る舞い

オブジェクトの機能定義とその振る舞いの動的束縛による状況に適応したアプリケーションを実現するためシステムを考察するために，本章では，用語の定義及び，インタラクティブスペースにおけるオブジェクト間の協調動作 (振る舞い) の分類及び記述者の記述能力の分類を行い，インタラクティブスペースに適した振る舞いの記述系を提案する．また，動的再構成を実現するシステムのアーキテクチャに関する分類を行い，インタラクティブスペースに適したアーキテクチャの提案を行う．

2.1 用語定義

本節では，アプリケーションの動的再構成に関する議論を行う上で明確にすべき用語の定義を行う．

2.1.1 インタラクティブスペース

インタラクティブスペースでは，情報家電，Smart Furniture，PC で稼動するアプリケーション，気温・湿度等の環境情報を取得するセンサ，人の入退室を検知する位置情報センサ等は分散コンポーネントとして表現され，お互いの機能及び情報を利用しあいながら様々な協調動作を実現する．また，分散オブジェクトの集合として表現されたインタラクティブスペースでは，以下のようなアプリケーションが可能となる．

直接操作型インタフェース

RF-Reader による位置情報を利用した直接操作型インタフェースの例を挙げる．PDA の GUI アプリケーションとプリンタの協調動作を例に挙げる．PDA は，操作性が PC に比べて格段に低い．また，携帯を目的に設計され，常に移動先での利用が想定される．ユーザは，PDA のデータを出張先の会社の目の前にあるプリンタに印刷したい．プリンタは，ユーザの目の前にあるが，プリンタの参照名は検索できても，目の前にあるプリンタの指定は困難である．この場合，RF-Reader 及び PDA の GUI アプリケーションとの協調動作として，PDA を目の前のプリンタに置くと印刷が行われる協調動作を定義しておくと，直接操作型インタフェースによる高いユーザビリティを持つインタラクションが可能となる．

情報家電間の協調

電話と AV 機器情報家電間の協調動作の例を挙げる．ユーザは，リビングでオーディオを大音量で聞いてたところ電話がコールされた．普段はそのままであるが，電話の相手が会社の上司であったためオーディオを消したい．このようなシナリオの場合，ユーザは電話とオーディオ機器の協調動作を予め定義し，自動化することができる．

2.1.2 アプリケーション

本論文で扱うアプリケーションとは，あるホスト上のソフトウェアがそれ自体単体で動作するものではなく，他のホストで稼動するソフトウェアと協調して動作しなんらかの問題を解決するものである．

2.2 分散オブジェクトの協調動作記述

本節では，分散オブジェクト間の協調動作記述について分類を行う．

2.2.1 協調動作記述の分類

分散オブジェクトが協調動作を行う場合，協調動作の記述を行う必要がある．協調動作記述は，静的な協調動作記述モデルと動的な協調動作モデルに分けられる．静的な協調動作モデルは，分散オブジェクトに協調動作プロトコルが埋めこまれ，分散オブジェクトを一度配置してしまえば，分散オブジェクト間の協調動作を変更することは困難である．協調動作プロトコルとは，分散オブジェクト間でどのようなデータが流れるか，また，分散オブジェクトが提供するメソッドがどのような制御フローで呼び出されるかということを指す．従って，協調動作プロトコルの記述が分散オブジェクトの機能定義と区別されない協調動作記述モデルは，ユーザの要求に適応的な協調動作記述モデルではない．一方，動的な協調動作記述モデルは，分散オブジェクトの機能定義と協調動作プロトコルが分離され記述される．従って，分散オブジェクトの記述を一切変更することなく，協調動作プロトコル記述の変更のみにより，ユーザの要求に適応的な分散オブジェクト間の協調動作が実現できる．図 2.1 に動的な協調動作記述モデルを示す．分散オブジェクト間は，それぞれ他の分散オブジェクト参照することもなく，従って，図 2.1 のように他の分散オブジェクトと独立している．分散オブジェクトが協調するアプリケーションを定義する場合は，分散オブジェクト同士がメソッドを呼び出し合う必要がある．分散オブジェクト間のメソッド呼び出し関係の定義と分散オブジェクトの機能定義を分離し，協調動作定義の評価・実行環境が実行時に分散オブジェクト間の呼び出し関係を決定する．

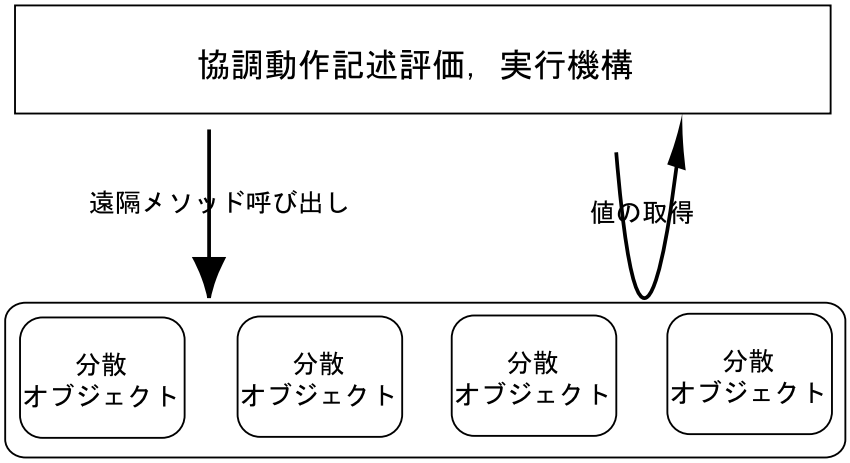

図 2.1: 協調動作記述と分散オブジェクト機能定義の分離

2.2.2 動的な協調動作記述モデル

動的な協調動作記述モデルは，分散オブジェクトの機能定義と分散オブジェクト間の協調プロトコルを分離する記述モデルである．動的な協調動作記述は，協調動作記

述に関して，データパス記述型及び制御フロー記述型に分かれる．以下にそれぞれの特徴を述べる．

データパス記述型

データパス記述型の協調動作記述モデルとは，分散オブジェクト間を流れるデータパスを記述するモデルをいう．データパス記述型では，分散オブジェクトは入力及び出力を持ち，この分散オブジェクト間の入出力をつなぎかえることで協調動作を定義する．

データパス型の特徴は，協調動作記述がデータフローのみであるため，記述が容易である半面，流れるデータ内容による処理の分岐等，制御フローを分散オブジェクトにおいて記述する必要があり，表現力に欠ける．適応分野として，同型のデータの処理の流れを動的に変更するフィルタ機構などに適している．データパス型の振る舞い記述系の例として，MANIFOLD[1] が挙げられる．MANIFOLD は，プロセス，ポート，ストリーム及びイベントを組み合わせることによりデータフローを構築する．

制御フロー記述型

制御フロー記述型の協調動作記述モデルは，分散オブジェクトの制御 (分散オブジェクトのメソッド呼出し) の流れに関する条件分岐及び繰り返しを記述しすることで協調動作を定義する．

制御フロー記述型の特徴は，協調動作記述において，制御フローを記述できるため，多彩な振る舞いの記述が可能となる半面，記述が，既存の構造化プログラム言語に近く記述系と記述対象の隔りが大きい．制御フロー記述型の記述系の例として，CLAM[6] が挙げられる．CLAM は，データフロー及び制御フローを記述でき，上位記述系において，分散オブジェクトのメソッド呼出しと結果の取得や変数の定義，分岐文の記述が可能である．

2.3 振る舞いの動的束縛による協調動作記述

本研究では，分散オブジェクトの機能定義と制御フローによる協調動作プロトコルの束縛を振る舞いの動的束縛と呼ぶ．本節では，振る舞いについての定義とその特徴について述べ，最後に本研究で利用する振る舞いによる協調動作記述モデルをまとめる．

2.3.1 振る舞い

本研究において振る舞いとは，分散オブジェクト間の協調動作を指し，分散オブジェクトが取り得る状態の遷移を切っ掛けに，他の分散オブジェクトが様々な状態をとるような動作を扱う．

2.3.2 振る舞いの分類

オブジェクト間の振る舞いの記述を行うには，記述系において振る舞いの動的な側面 (系の挙動) を捉える必要がある．系の挙動とは，任意の時刻 t における分散オブジェクトの状態を指す．系の挙動は，制御フロー型の記述モデルの場合，離散時間的挙動及び離散時間的挙動の 2 つが記述できる．

1. **離散時間的特性**
   離散時間的特性とは，振る舞いの動的な側面 (系の挙動) を状態変数の依存関係や代数方程式及び微分方程式で表現される特性を指し，状態の変化が連続的な系の表現に適している．

2. **離散事象的特性**
   離散事象システムにおける系の挙動は，系が離散的な状態の集合及びその遷移規則で表現され，離散状態の集合からなる系の表現に適している．

2.3.3 振る舞いの記述能力

本項では，振る舞い記述の技能に関する分類を行う．まず，ノーマンは人間のシステム使用時における一連の認知過程 [7] を以下のように分類している．

1. **目標・意図の設定**
   例えば，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欲しいと思う段階である．

2. **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．

3. **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．

4. **知覚**
   知覚とは，働き書けた結果，システムの動きを知覚する段階である．

5. **状態解釈・評価**
   状態の解釈・及び評価とは，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．

上記の認知過程に関する分類において，本研究における振る舞いの記述行為は上記の「システムに対する行為系列の特定」に相当する．本研究では，記述に関する行為系列の多様度に着目し，インタラクティブスペースにおけるアプリケーション記述者の技能に関して以下のように分類する．

多様度小 記述系の多様度が低い場合は，例えば，予め定義済みの振る舞いをユーザの要求にもとづき選択する場合は，ユーザの要求が予め定義されたものと合致していれば問題ないが，合致していない場合は，振る舞い定義は無駄となる．この場合，記述者の能力は，プログラムを記述することについて全く関知しなくてよい．

多様度中 問題領域に近い表現により，要求された振る舞いを記述すると記述者はある程度記述系の構文等について理解する必要があが，記述力は多様度小の場合に比べ表現に富む．

多様度大 計算機に近い表現によって，要求される振る舞いを記述することは，計算機で可能なことはほぼ実現できるが，記述が対象と解離しているため，記述系の構文及び意味と対象の差異を埋める能力が必要である．

2.4 採用するモデル

本節では，第 2.2 節及び第 2.3 節で述べた分類をもとに，アプリケーションの動的再構成に適した振る舞い記述の考察及び提案を行う．

2.4.1 採用する記述モデル

本研究がターゲットするアプリケーションは，第 1.3 節で述べたインタラクティブスペースにおける分散オブジェクト間の協調動作の実現である．これらアプリケーションにおけるオブジェクト間の振る舞いは，離散的状態と連続的状態が混在する系であり，離散時間的側面及び離散事象的な側面から記述される．しかし，第 2.3.3 項において要求の実現性 (記述者の記述技能) と行為系列の多様度の関係を見たように，計算機に対するプログラミングに習熟していないエンドユーザ又はシステム管理者がアプリケーションの動的再構築を容易に実現するには，記述モデル及び記述形態をエンドユーザから見た対象に近い形で提示する必要がある．

離散事象的表現は，オブジェクトのユーザによる観察可能な状態及びその遷移を記述することで表現され，記述と対象が近く，記述自体に振る舞いの意味が表現可能なため記述が容易である．一方，離散時間的振る舞いの表現は，代数方程式及び微分方程式により表現され，ユーザは，記述対象と記述の表現に隔りがあるため，記述が困難である．

本研究では，離散時間的振る舞いは，分散オブジェクトの機能定義において記述され，振る舞いの記述系では，分散オブジェクトの離散事象に基づいた振る舞いの記述を行う．離散事象に基づいた記述とは，分散オブジェクト間の振る舞いに関して，**ユーザが直接認知可能な離散状態の集合及びその遷移として記述する対象指向的記述系**を提案する．この記述系の特徴は，**分散オブジェクトの状態に基づいた制御フロー記述型による振る舞いの記述**であり，また**対象指向記述**である．

2.5 本章のまとめ

本章では，分散オブジェクトの協調動作記述について述べ，アプリケーションの動的再構成に適した記述系を考察し，採用モデルについて述べた．次章では，本節で見た記述モデルを表現する記述言語 Interactive Space Programming Language (ISPL) の定義を述べる．

第 3 章

Interactive Space Programming Language

本章では，アプリケーション動的再構成を実現する協調動作記述系 ISPL について述べる．ISPL の特徴は，オブジェクトの機能定義と振る舞いの分離によるアプリケーションの動的再構成を実現する機構において，情報家電機器，ソフトウェア等の分散オブジェクトの状態に関連した振る舞いの記述を離散事象系として把握し，データフロー及び制御フローについて記述することである．また，ユーザが認知可能な状態の集合及びその遷移をもとに記述することで，記述対象関する意味が記述系に表現でき，記述モデルが記述対象に近似的である．

3.1 Interactive Space Programming Languageの概要

本節では2.4.1項で述べた振る舞いの記述モデルに従い振る舞い記述言語Interactive Space Programming Language(ISPL)の概要について述べる．ISPLの特徴は，分散オブジェクトが取る状態の遷移規則の定義による振る舞いの記述であり，分散オブジェクトが取る状態とは，**ユーザが直接認知可能な分散オブジェクトの離散状態**である．従って，記述モデルと記述対象が近似的であり，**対象指向記述**である．ISPLでは，分散オブジェクト間の協調動作である遠隔メソッド呼び出しは分散オブジェクトの状態遷移として表現される．また，ISPLでは，オブジェクトの状態に入出力を持たせることで，状態の遷移系列を定義することでデータフローを表現する．

```
HANDLE light;//部屋の電燈を表す分散オブジェクト
HANDLE radio;//ラジオを表す分散オブジェクト
HANDLE television;//テレビを表す分散オブジェクト
//一定時刻置きに
while(true){
  if(light.getState().equals("OFF")){
    ((TV)television).turnOff();//テレビを消すオペレーション
    ((Radio)radio).turnOn();//ラジオを付ける
  }
  Sleep(10);//一定時刻待つ
}
```

図 3.1: 手続指向言語による擬似コード

図3.1で見るような従来の言語による振る舞い記述では，機能(メソッド呼出し)及び状態の管理が区別されて表現されていたが，ISPLでは振る舞いを状態及び状態の遷移系列により表現する．また，制御フローの記述は，状態及び過去の状態遷移系列及び状態間を流れるデータ内容による記述が行える．

3.2 Interactive Space Programming Languageの設計

本節では，Interactive Space Programming Language(ISPL)の設計について述べる．ISPLでは，分散オブジェクトの状態の遷移系列を記述することで振る舞いを定義する．状態の遷移系列の記述を行うには，分散オブジェクトの状態の定義及びその遷移規則の定義が必要である．

3.2.1 オブジェクトの定義

ISPL では，オブジェクトをプロセスの集合として定義する．プロセスは，オブジェクトの取り得る状態でかつ排他的関係にある状態の集合からなる．オブジェクトの状態とは，例えば，DVD が再生中状態である，停止中状態であるなどの図 3.2 に示すユーザによる観測可能な状態を指す.

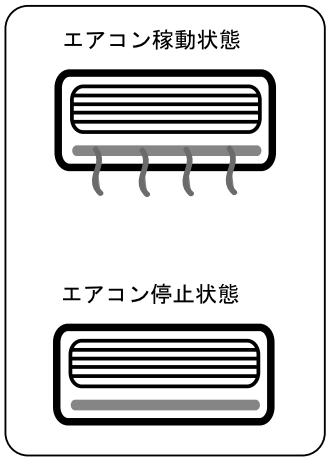

図 3.2: オブジェクトの状態

プロセスの定義

プロセスは，オブジェクトの取り得る状態の集合である．また，プロセスを構成する状態は互いに排他的であり，従って，プロセスを構成する状態は，時刻 t において同時に成立しない.

状態の定義

状態とは，分散オブジェクトが不変条件を保つ間の状況を表し，他のどの分散オブジェクトとの関係に依存せず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，動的な状況を表現することもできる．静的な状況とは,「DVD プレーヤの停止状態」などのように，ユーザが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，なんらかの活動を実行しているプロセス，例えば,「DVD プレーヤの再生中状態」ような動的な状況である．オブジェクトは，このような状態の集合として定義される．状態の構成要素を表 3.1 に示す.

入場動作として記述可能な分散オブジェクトが提供するメソッドのシグネチャは，表 3.2 で見るように入力, メソッド識別名及び出力である．入力及び出力は図 3.3 で見るように状態が持つ入出力に対応する.

表 3.1: 状態の構成要素

名前	状態はプロセス内で一意な名前を持つ.
入場動作	入場動作とは，静的な状態や動的な状態に関わらず，状態に遷移する為に呼び出されるメソッドの型 (シグネチャ) 記述する．例えば，DVD プレーヤを再生させるるリモードメソッド A が定義されている場合，入場動作として A のシグネチャを記述する.

表 3.2: リモートメソッドのシグネチャ

入力	状態は入力を持たないかもしくは n 個の入力を持つ.
出力	状態は出力を持たないかもしくは 1 個の出力を持つ.
識別名	リモートメソッドの識別名

3.2.2 遷移規則

遷移規則とは，上記のオブジェクトの状態間の遷移を決定する．遷移規則は，トリガ，遷移先及びガードにより構成される．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，オブジェクトの状態及び状態遷移系列である．また，状態の遷移系列をトリガに指定する場合，遷移系列評価の有効時間を指定する.

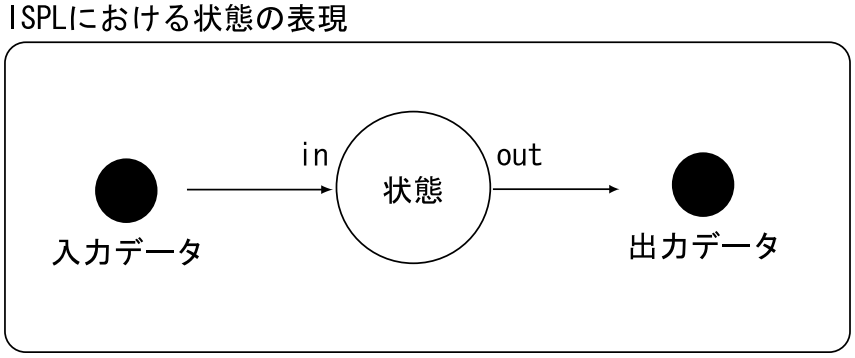

図 3.3: 状態の入出力

ガード

ガードは，遷移の際してトリガの他にさらに細かい制御として遷移に関連付けられた論理式である．トリガである遷移の条件が満たされた場合に，ガードとして指定された論理式が評価される．論理式の評価結果が真の場合，状態の遷移が実行され，偽の場合は遷移が抑制される．論理式として，排他状態の指定及び状態間を流れるデータを参照する論理式を記述することでより条件分岐などの細かい制御フローの記述を行う．ガードとして以下の論理式を記述できる．

- **状態の排他的関係**
  状態の排他的関係を定義することで，並行動作における処理の流れの同期を記述できる．

- **データ内容による論理式**
  データの内容による論理式を定義した，遷移規則を複数定義することで，データ内容による条件分岐の記述ができる．

3.2.3 データフロー

状態は，有限個の入力と 1 つの出力を持つ．データフローは，遷移先の入力チャネルと遷移元からの出力を接続することで記述する．状態間のデータ例として，ユーザインタフェースなどのオブジェクトと DVD などの情報家電オブジェクトが協調動作を行いたい場合に，ユーザからの入力などが考えられる．図 3.4 に状態の入力チャネル及び出力チャネルの関係を示す.

3.3 ISPL の構文

ISPL は，eXtensible Markup Language(XML)[11] による構造化文書として記述される．本節では，XML 言語による優位性及び実際の XML による ISPL 記述について述べる．

3.3.1 構造化文書 eXtensible Markup Language による ISPL

ISPL は XML による構造化されたドキュメントとして記述される．XML の利点は，言語処理系の実装の容易性である．言語処理系 (インタプリタ) の実現は，図 3.5 のような構成になる．パーサの役割は，字句解析，構文解析，意味解析を行うことである．XML を使う利点は，パーサの字句解析，構文解析を汎用 XML パーサが行うことである．従って，処理系の実装者は，パーサの意味解析部及びエバリュエータの設計・実装を行うだけでよい．このことは，アプリケーション記述要求の変化による言語の段階的発展に伴う処理系の柔軟な変更要求の対処に適した環境である．また，現在汎用 XML パーサは，Java，C/C++，Perl をはじめ様々な言語に実装されているため，処

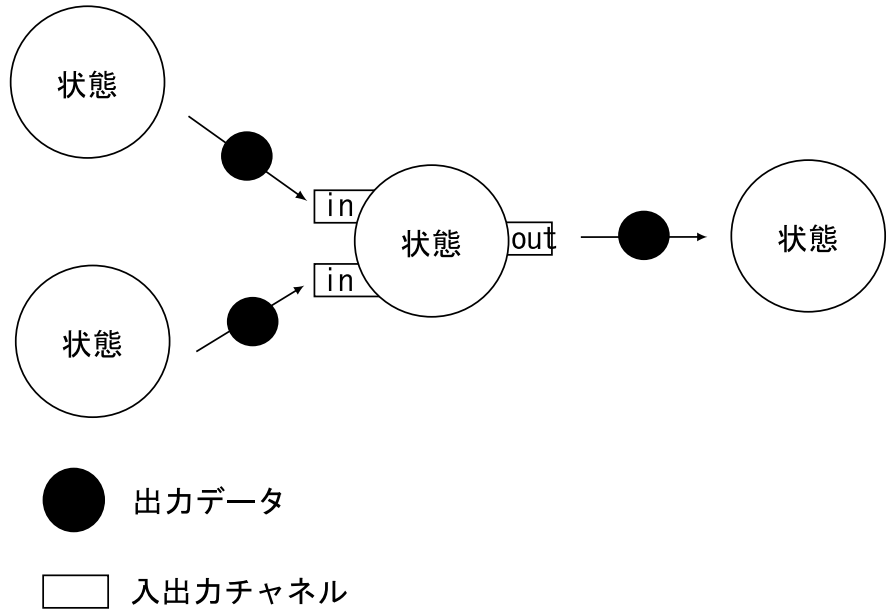

図 3.4: 状態の入出力チャネル

理系の実装も多様な言語による実行環境上で可能となる．これらのことから，ISPL は，XML による実現が適している．

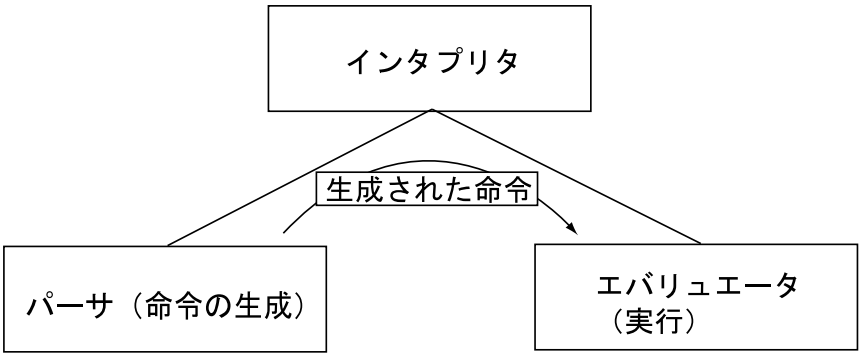

図 3.5: 言語処理系の構成

3.3.2 ISPL の宣言

ISPL は，XML による構造化された文書であるため，XML 文書の宣言をファイルの先頭で行う．また，ISPL であることを宣言するために，<ISPL>タグを記述する．

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

3.3.3 オブジェクトの定義

本項では，ISPL におけるオブジェクトの定義について述べる．オブジェクトの定義は，<entity>タグにより行い，属性として，オブジェクトの識別名を記述する．

```
<ISPL>
  <entity name="ObjectName">
  </entity>
<ISPL>
```

また，オブジェクトは，第 3.2 節で述べたように，分散オブジェクトの状態の集合される．以下に XML による状態の定義を示す．

分散オブジェクトの状態

オブジェクトの状態の定義には，<internal_state_definition>タグを用いる．

```
<!--内在的状態-->
<entity name="ObjectName">
  <internal_state_definition>
  </internal_state_definition>
</entity>
```

さらに，排他的状態の集合であるプロセスの集合を定義する．プロセスの定義は，<process>タグを用いる．また，オブジェクトは 1 つ以上のプロセスからなる．

```
<!--プロセス-->
<internal_state_definition>
  <process name="ProcessName">
  </process>
</internal_state_definition>
```

プロセスは，有限個の状態の集合である．状態の定義は，<state>タグを用いる．属性はとして，所属するプロセスで固有な識別名を持つ．また，状態に入る入場動作を定義する．入場動作に定義は，<entrance_action>タグを用い，入場動作のシグネチャを<signature>により定義する．<signature>の属性として，入出力の型を定義する．入出力の型を表 3.3 に示す．また，入力チャネルの定義は，<input_channel>で行う．

```
<!--状態-->
<state name="StateName">
  <entrance_action>
   <signature type="in/out">
     <invoked_method>hoge</invoked_method>
     <input_channel>arg1</input_channel>
     <input_channel>arg2<input_channel>
   </signature>
  </entrance_ction>
</state>
```

表 3.3: 状態が持つ入出力の型

void/void	状態は入出力を持たない.
in/void	状態は入力を持つが出力を持たない.
void/out	状態は出力を持つが入力を持たない.
in/out	状態は入出力を持つ.

3.3.4 振る舞いの定義

本項では, ISPL における振る舞いの定義について述べる. 振る舞いの定義は, <behavior> タグを用いる. 属性として, 振る舞いを識別する一意な名前を記述する.

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

振る舞いは, 前節で述べたように, 状態の遷移規則の集合として定義される. 以下に XML による遷移規則の定義について述べる. 遷移規則の定義は, <transition_rule> タグを用いる.

```
<behavior name="BehaviorName">
  <transition_rule>
  </transition_rule>
</behavior>
```

遷移規則は, 遷移条件, 遷移先及びガードにより構成される. 以下にそれぞれの記法について述べる.

遷移条件の記法

遷移条件は，<trigger>タグを用いる．

```
<transition_rule>
  <trigger>
  </trigger>
</transition_rule>
```

トリガには，状態もしくは状態の遷移系列を指定する．状態の指定は，<state>タグを用いる．属性として，状態を持つオブジェクトの識別名を指定する．

```
<trigger>
  <state entity="ObjectName">StateName</state>
</trigger>
```

また，同時に成立する複数の状態をトリガとする場合，<simultaneous_states>タグを用いる．<simultaneous_states>の属性として，同時性の有効時間を指定する．有効時間は正規表現

$$digit^*(ms|s|h|) \tag{3.1}$$

で表される．

```
<trigger>
  <simultaneous_states valid_time="12s">
    <state entity="ObjectName">StateName</state>
    <state entity="ObjectName">StateName</state>
  </simultaneous_states>
</trigger>
```

遷移系列をトリガとして指定する場合，<transition_sequence>タグ及び<state>タグを用いる．また，<transition_sequence>タグの属性として，遷移系列評価の有効時間を指定する．有効時間は正規表現

$$digit^*(ms|s|h|) \tag{3.2}$$

で表される．

```
<trigger>
  <transition_sequence valid_time="1200ms">
    <state entity="ObjectName">StateName</state>
    <state entity="ObjectName">StateName</state>
  </transition_sequence>
</trigger>
```

遷移先の記法

遷移先の定義には，<transition_target>タグ及び<state>タグを用い，遷移先オブジェクトの状態を定義する．<transition_target>タグの属性として，オブジェクトの識別名を記述する．<state>タグにより遷移先状態を定義する．<io_channel_mapping>タグを用いる．

```
<transition_rule>
  <transition target_entity="ObjectName">
    <state>StateName</state>
  </transition>
</transition_rule>
```

また，データフローをI/O チャネルのマッピングにより定義する．状態が持つインプットチャネルと他の状態のアウトプットチャネルを接続する．I/O チャネルのマッピングは，<tuple>を用い，<input_channel>タグ及び<source_state>タグによるI/O のマッピングを記述する．

```
<transition target_entity="ObjectName">
  <io_channel_mapping>
    <tuple>
      <input_channel>InputChannelName</input_channel>
      <source_state>SourceStateName</source_state>
    </tuple>
  </io_channel_mapping>
</transition>
```

ガードの記法

ガードの定義には，<gard>タグを用いる．

```
<transition_rule>
  <gard>
  </gard>
</transition_rule>
```

ガードには，排他的な状態の集合及び状態間を流れるデータを評価する論理式を記述する．排他的状態の集合は，<exclusive_state> により指定する．

```
<gard>
  <exclusive_state entity="ObjectName">StateName</exclusive_state>
  <exclusive_state entity="ObjectName">StateName</exclusive_state>
</gard>
```

状態間を流れるデータを評価する論理式は，<logic_formula>タグを用いる．<logic_formula>に命題を定義する．

```
<gard>
  <logic_formula>
    [ParameterName1 is bigger than ParameterName2]
  </logic_formula>
</gard>
```

3.4 ISPL の利用

本節では，ISPL の利用モデルを述べる．アクタはエンドユーザ，システム管理者及びオブジェクトの提供者である．本研究が期待するアクタの役割を以下に述べる．

エンドユーザ

エンドユーザとは，一般に目的を実現するための労力を惜しむ．よって，予め定義された振る舞いテンプレートもとに振る舞いを記述することにより労力を最小限にすることが望ましい．

システム管理者

現在，システム管理者は，現在取り得る方法を駆使して目的を達成しようとする．よって，システム管理者は，ISPL をスクラッチから記述することが期待できる．

オブジェクトの提供者

情報家電等を提供するベンダがオブジェクトの提供者である．オブジェクトの提供者は，オブジェクト自体の配布とともに，オブジェクトが取り得る状態の定義する ISPL による記述を提供する．また，エンドユーザ向に振る舞いのテンプレートを定義し提供することが期待される．

3.5 本章のまとめ

本章では，アプリケーションの振る舞いを記述する Interactive Space Programming Language の要件，意味論，統辞論について述べた．また，オブジェクトの提供者，エンドユーザ及びシステム管理者について，本研究で期待する記述内容について述べた．次章では，アプリケーションの動的生成機構 ASAR における ISPL の処理系について述べる．

第 4 章

アプリケーションの動的再構成を実現する基盤システム ASAR の設計

本章では，アプリケーション動的再構成を実現する記述系 ISPL の実行環境である ASAR の設計を述べる．ISPL によるアプリケーション記述系及び ASAR によるアプリケーション実行環境は，分散オブジェクトの機能定義と振る舞いの分離し，アプリケーションを記述することを特徴とする．また，振る舞いの記述系は，分散オブジェクト間の協調動作をユーザの認知可能な離散状態の遷移系列として記述することを特徴としている．

4.1 ASARの設計方針

本節では，ASARの設計方針であるユーザの要求適応性及び分散オブジェクトの動的配置性ついて述べる．ユーザの要求適応性とは，ユーザの要求に応じた分散オブジェクト間の協調動作を動的に再定義可能な機構により実現される．アプリケーションの構築にあたり，オブジェクトの機能定義と協調動作定義を分離し，ユーザによる状況に適したオブジェクトの協調動作の再定義を実現する．オブジェクトの機能定義とオブジェクト間の振る舞い定義を分離し，オブジェクト間の振る舞いを記述可能な言語ISPLを第3章で定義した．これを機能定義と振る舞いの動的束縛によるアプリケーションの動的再構成と呼ぶ．振る舞いの動的束縛機構は，以下の機能により実現される．

4.1.1 振る舞い定義の評価・実行機能

振る舞いの定義ファイルを解析し，定義ファイルに基づいてオブジェクトの振る舞いを実行する機能が必要となる．

4.1.2 分散オブジェクトの動的配置機能

また，ASARの設計分散オブジェクトの動的配置機能とは，分散オブジェクトがASARシステムにが容易に参加でき，新たに参加したオブジェクトとの振る舞いをシステムの再起動なしにユーザが記述可能でなければならない．このために，オブジェクトの発見及び登録機構などの動的オブジェクト参加機構が必要となる．

4.2 ASARの設計

本節では，第4.1節で述べた，アプリケーションの動的再構成を実現する機能要件に従いASARの設計を示す．ASARの主な主張は，アプリケーションを分散オブジェクトの機能定義と協調動作定義に分離し，ISPLによる協調動作定義の記述によりアプリケーションの動的再構成を実現することである．ASARのハードウェア構成及びソフトウェア構成を以下に示す．

4.2.1 ハードウェア構成

本システムにおいて，ハードウェア構成は図4.1に示されるように3層に区分される．第1の層は，システムがユーザに提示する環境世界である．この層では，情報家電やソフトウェアサービスが存在し，分散オブジェクトとして相互に協調が可能となる層である．第2の層は，第1層に存在するオブジェクトにネットワーク相互接続性を持たせる層である．分散オブジェクトは，この層のネットワーク相互接続性により協調動作を実現する．第3の層では，アプリケーションの動的再構成を実現する基盤システムASARが配置されるコンピュータが存在する．この基盤システムASARにより，第1層におけるオブジェクトの振る舞いが動的に規定される．

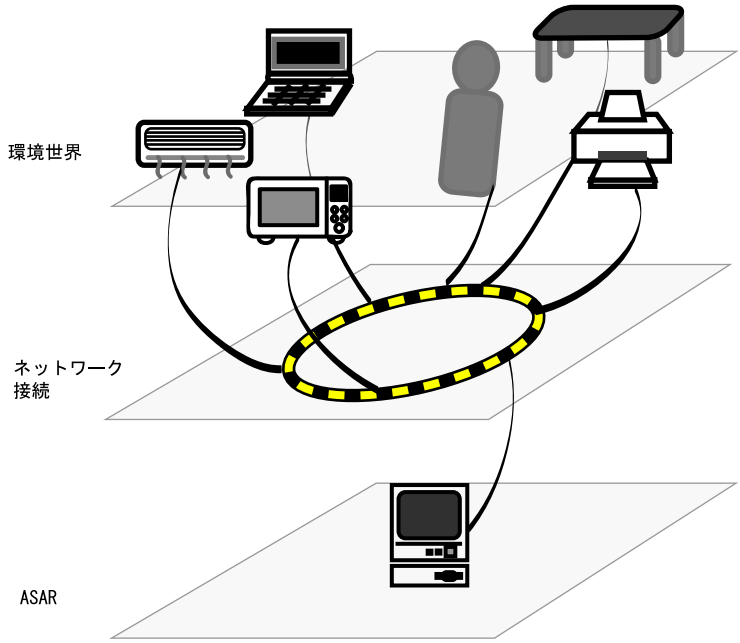

図 4.1: ハードウェア構成

4.2.2 ソフトウェア構成

本システムのソフトウェア構成は図 4.2 に示す．本節では，ARAR システムの中，振る舞いの動的束縛機構である Behavior Interpreter モジュール，Behavior Activator モジュール，ASAR Abstract Object モジュールについて述べ，分散オブジェクトの動的配置機能である Object Deployment モジュールについて説明する．以下に，それぞれの役割及び関係を述べる．

Behavior Interpreter モジュール

Behavior Interpreter では，ISPL で記述された振る舞い定義ファイルの字句解析，構文解析及び意味解析を行い，記述内容の評価・実行を行う．Behavior Interpreter モジュールは，Behavior Parser，Behavior Evaluator 及び State Change Notifier の各サブモジュールにより構成される．

Behavior Activator モジュール

Behavior Activator モジュールは，振る舞い定義ファイルの読み込みを行い，Behavior Interpreter モジュールにおいて振る舞いの評価・実行を可能にする．また，現在評価・実行対象となっている振る舞いの定義ファイルを取得し振る舞いの再定義を可能にする．

Object Deployment モジュール

Object Deployment モジュールは，オブジェクト動的参加機能を実現する．インタラクティブスペースへのオブジェクトを参加を検知し，オブジェクトの定義ファイルを読み込み及び Behavior Interpreter モジュールにおいて新規参加オブジェクトとの振る舞いの評価・実行を可能にする．

ASAR Abstract Object モジュール

ASAR Abstract Object モジュールとは，ASAR システムにおけるオブジェクトが共通して持つ機能をモジュール化したものである．

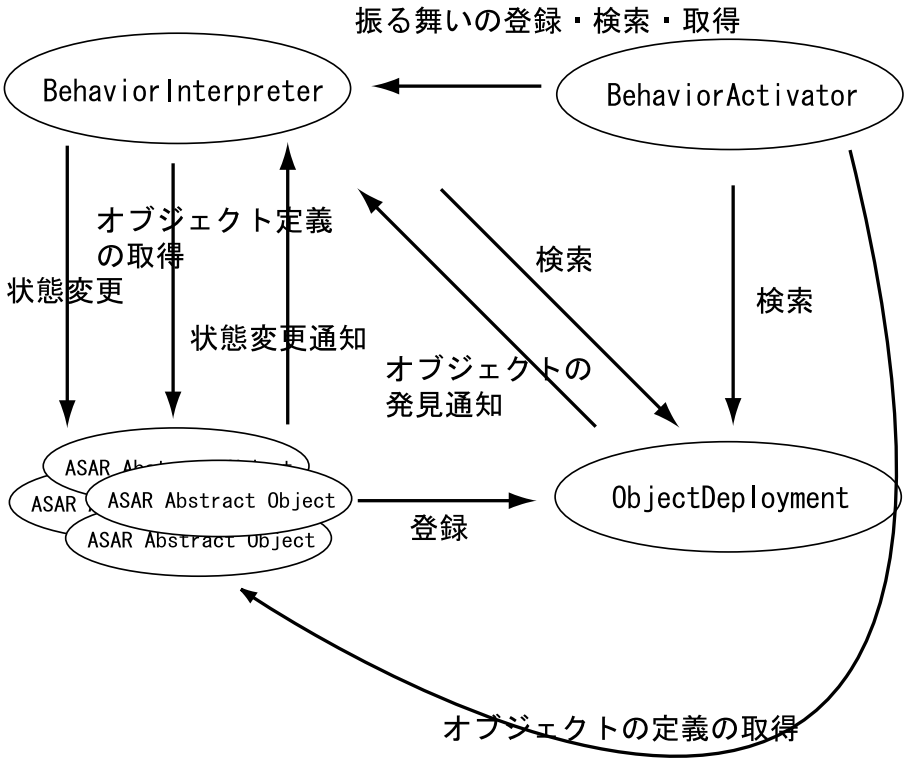

図 4.2: ASAR を構成するモジュール間の関係

4.3 Behavior Interpreter

Behavior Interpreter モジュールは，Behavior Parser，Behavior Evaluator，State Change Notifier 及び State Recorder のサブモジュール群により構成される．本節では，Behavior Interpreter を各サブモジュール群の構造，データフロー及び挙動により定義する．

Behavior Parser

Behavior Parser は ISPL 言語で記述した振る舞い定義ファイルの字句解析，構文解析及び意味解析を行う．Behavior Parser は，意味解析終了時に状態遷移規則テーブルを出力する．状態遷移規則テーブルとは，ISPL で定義した遷移規則の集合である．

Behavior Evaluator

Behavior Evaluator は，Behavior Parser が生成した状態遷移規則テーブルと状態テーブルをもとに ASAR Abstract Object に対する状態変更要求出す．状態変更要求は，ASAR Abstract Object が提供する分散オブジェクトの遠隔メソッドを呼び出すことで実現する．

State Recorder

また，遷移規則のトリガとして遷移系列が指定された場合に，Behavior Evaluator がトリガを適切に評価する必要がある．したがって，Behavior Evaluator は ASAR Abstract Object の状態遷移履歴の管理は，State Record Manager が行う．State Record Manager は，状態遷移履歴を利用した効率的な振る舞いの評価・実行のために，State Record Manager が状態遷移履歴を管理するデータ構造について述べる．状態遷移履歴は，FIFO 構造体として表現される．図 4.3 に FIFO と遷移系列の関係を示す．

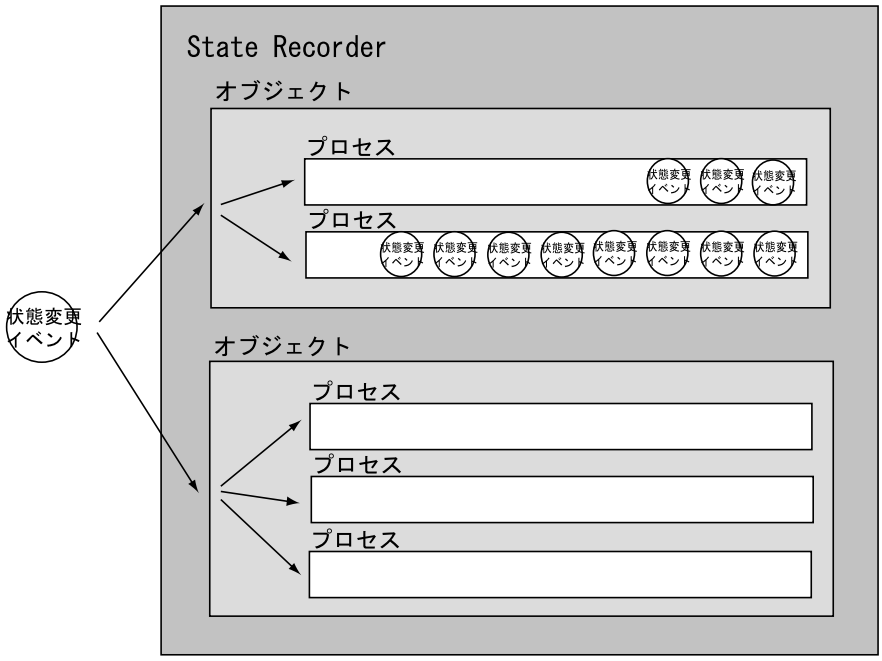

図 4.3: 状態遷移履歴管理

State Change Notifier

State Change Notifier は，ASAR Abstract Object の状態の変更を検知し Behavior Evaluator に通知する．

4.4 Behavior Activator

Behavior Activator モジュールは，Behavior Interpreter に対して振る舞い定義の登録・検索及び定義ファイルの取得を行う．また，オブジェクトの検索を行い，オブジェクトの定義ファイルを取得する．振る舞い定義の登録・検索・定義ファイル取得はネットワークを介して行えること，振る舞い定義の検索に関しては，検索方法が多様であることが要件として挙げられる．名前検索以外にも，例えば振る舞いに関連するオブジェクトで検索することや他の様々なメタ情報を利用した全文検索などが考えられる．

4.5 Object Deployment

Object Deployment モジュールは，ASAR Abstract Object の統一的な管理及び検索を実現する．ASAR Abstract Object が ObjectDeployment モジュールを発見するためにマルチキャストを利用する．しかし，マルチキャストによる Object Deployment モジュールの発見機構を利用した場合，インターネットを越えて，Object Deployment モジュールの発見を行うことができない．この問題を解決するために，Object Deployment モジュールは，インターネットを介して同期をとる機構が必要である．同期機構として，以下の 2 つの方法が考えられ，Object Deployment モジュールの機構を単純化するために，方法 B による同期機構を採用する．

方法 A：Object Deployment モジュール間で登録同期

インターネットを越えて，Object Deployment モジュール間で登録されれいる ASAR Abstract Object の情報を Object Deployment モジュール自身が，インターネット上の他の Object Deployment モジュールと常に同期をとる機構を持つ方法である．この方法は，Object Deployment モジュールの本来の ASAR Abstract Object の統一的な管理及び検索以外の機構を備える必要があるため，ソフトウェア構成がシンプルではない．

方法 B：マルチキャストパケットフォワーディング

マルチキャストパケットフォワーディングによる方法における Object Deployment モジュールの構成は，インターネット越しの他の Object Deployment モジュール間の同期機構を必要としない．TCP/IP のパケットにパケットマルチキャストパケットを内包させ，インターネットを介して他のネットワークへ転送する．この方法は，Object

Deployment モジュール自体に，同期機構を必要とせずソフトウェア構成がシンプルなものとなる．

4.6 ASAR Abstract Object

ASAR Abstract Object は，ASAR におけるオブジェクトを表現し，オブジェクトが共通して持つ機能をモジュール化したものである．共通する機能を ASAR Abstract Object として抽象化し，フレームワーク化することにより，オブジェクト提供者は分散オブジェクトのアプリケーションロジックに集中して開発を行える．ASAR においてオブジェクトを提供するベンダは，ASAR Abstract フレームワークに基づいてオブジェクトを実装する．共通して持つ機能とは，分散オブジェクトの状態の変更通知機構，ISLP の状態定義ファイルの保持及び ObjectDeployment モジュールへの登録機構を備える．ASAR Abstract フレームワークは，**状態遷移通知機能**， **告知機能**，**オブジェクト定義リポジトリ**の機能を提供する．

状態遷移通知機能

状態遷移通知機能とは，オブジェクトの状態変更を Behavior Interpreter に通知する機構である．

告知機能

告知機能とは，ASAR Abstract Object が起動されると Object Deployment モジュールに対して自己の存在を告知する．告知は，マルチキャストを利用し，ASAR のネットワークにおける識別名のみをマルチキャストする．

オブジェクト定義ファイルリポジトリ

オブジェクト定義ファイルリポジトリとは，ISPL により定義されたオブジェクト定義ファイルを保持し，Behavior Interpreter 及び Behavior Activator からオブジェクトの定義ファイルの取得要求を受理しオブジェクト定義ファイルを送信する．

4.7 ASAR Abstract Object モジュール と Behavior Interpreter モジュール間の関係

ASAR Abstract Object と Behavior Interpreter の構成として以下の 2 つ関係が考えらる．

集中管理型

集中管理型アーキテクチャとは，図 4.4 のような制御フロー及びデータフローを集中的に管理するアーキテクチャである．このアーキテクチャの特徴は，データフローと制御フローを集中的に管理するため，集中管理機構がボトルネックとなりがちである．しかし，分散オブジェクト自体に，通信機構及び動的にデータフロー及び制御フローを変更可能な機構が必要としないため，集中管理機構以外は軽量コンポーネントでよい．

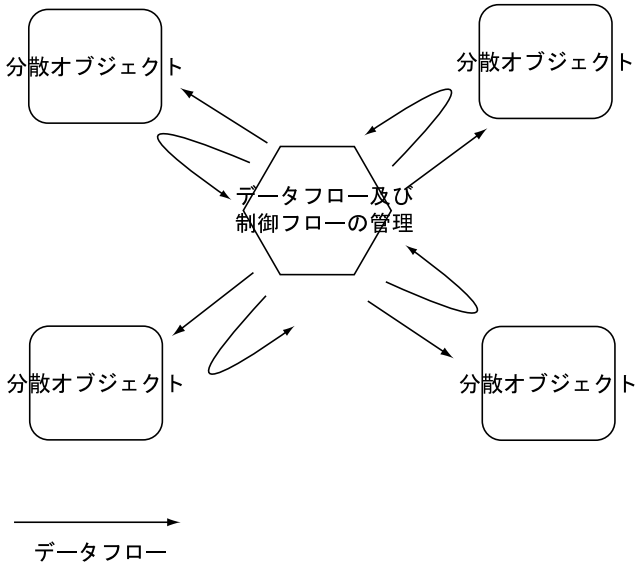

図 4.4: 集中管理型アーキテクチャ

分散管理型

分散管理型アーキテクチャとは，図 4.5 制御フロー及びデータフローをそれぞれの分散オブジェクトが管理するアーキテクチャである．このアーキテクチャの特徴は，通信時に，集中管理部ボトルネックになることがなく通信効率にすぐれているが，分散オブジェクト自体に，通信機構及び動的にデータフロー及び制御フローを変更可能な機構が必要となる，

採用するアーキテクチャ

インターネットなどの広域分散システムにおいて，分散オブジェクトが配置される計算機の計算能力が潤沢に利用できる場合は，分散管理型が優れると言える．しかし，ホームネットワークにおけるインタラクティブスペースでの運用を考慮する場合，情報家電機器等に計算能力の乏しい組込み機器が多く存在し，このような機器に通信機構及び動的にデータフロー及び制御フローを変更可能な機構を求めるのは期待できない．

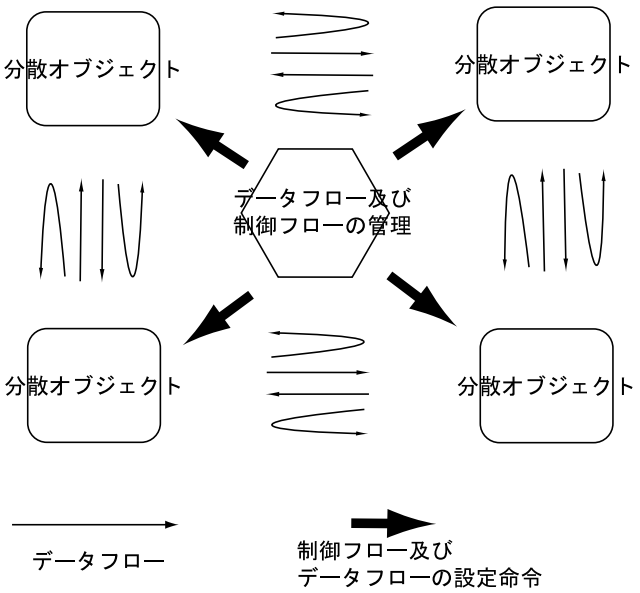

図 4.5: 分散管理型アーキテクチャ

また，組込み機器によっては，IEEE1394 や RS232C などを持つような分散オブジェクトとして実現できない機器も存在する．よって，本研究では，集中管理型アーキテクチャを採用する．

4.8 本章のまとめ

本章では，ASAR の根幹を成す振る舞いの動的再構成を実現する Behavior Interpreter モジュール及びその関連モジュールの設計について述べた．次章では，この設計に基づいた ASAR における Behavior Interpreter モジュール，ASAR Abstract Object モジュール．Object Deployment モジュール，Behavior Activator モジュールの実装について述べる．

第 5 章

ASARの実装

本章では，前章の設計に基づいた振る舞いの動的再構成機構 ASAR のプロトタイプ実装について述べる．

5.1 ASARのプロトタイプ実装

本節では，ASARのプロトタイプ実装について述べる．ASARの実装環境を表5.1に示す．ASARの実装にはプラットフォーム非依存性を実現するJavaVM及びJava言語を用いた．また，Behavior Interpreterモジュール，ASAR Behavior Activatorモジュール，Behavior Interpreterモジュール及びObject Deploymentモジュール間の通信は主にRMIにより行った．本実装では，XMLによるISPL振る舞い定義ファイルによる振る舞いの実行環境の根幹部であるBehavior Interpreterモジュール及びObject Deploymentモジュール，ASAR Abstract Objectモジュールを実装した．Behavior Interpreterモジュールの実装は，状態に関する振る舞い記述の評価・実行を実装した．表5.1に実装環境を示す.

表 5.1: 実装環境

項目	環境
CPU	Athron Thunderbird 1GHz
主記憶	DDR SDRAM 256MB
OS/Distribution	kernel 2.4.18/Vine Linux 2.4.18
開発言語/環境	Java言語/J2sdk1.4.0
分散オブジェクトシステム	Java RMI
XML処理系	JAXP

5.2 Behavior Interpreterモジュールの実装

本節では，Java言語を用いたBehavior Interpreterの実装について述べる．Behavior Interpreterは，BehaviorIntepreterFacadeクラス，Behavior Parserクラス，Object Parserクラス，Behavior Evaluatorクラス，State Change Notifierクラス及びState Recorderクラスから構成される．以下にそれぞれの実装について述べる．

5.2.1 BehaviorInterpreterFacadeクラス

BehaviorInterpreterFacadeクラスは，BehaviorInterpreterモジュールと他のモジュールとの間のRMIリモートインタフェースを定義する．表5.2にBehaviorInterpreterFacadeのリモートインタフェースを示す.

5.2.2 BehaviorInterpreterMediatorクラス

BehaviorIntepreterMediatorクラスは，Behavior Interpreterを構成するクラス群の結合度を低めることを促進する．これにより，BehaviorParser等のクラスの再利用を

表 5.2: BehaviorInterpreterFacade クラスのリモートインタフェース

public void stateChanged(StateEvent evt) throws RemoteException
public void objectDiscoveryEventFired(ObjectDiscoveryEvent evt) throws RemoteException

促進する．図 5.1 で示すように，クラス群のインタラクションは，つねに BehaviorIntepreterMediator を介して行われ，クラスの機能定義とインタラクションが切り放されるからである．表 5.3 に BehaviorInterpreterMediator のメソッドを示す.

表 5.3: BehaviorInterpreterMediator クラスのメソッド
public void update(BehaviorInterpreterColleague colleague，Object obj)

5.2.3 BehaviorInterpreterColleague 抽象クラス

BehaviorIntepreterColleague とは，BehaviorInterpreter モジュール構成するサブモジュール群に共通する機能を持つ．共通する機能とは，イベント通知機構である．サブモジュール間の通信は，BehaviorInterpreterMediator を介して行い，サブモジュール群が生成するイベントを BehaviorInterpreterMediator が消費し，イベントの種類によって，サブモジュールのメソッドを呼ぶ．BehaviorInterpreterColleague のメソッドを表 5.4 に示す.

表 5.4: BehaviorInterpreterColleague 抽象クラスのメソッド
public void addBehaviorMediator(BehaviorInterpreterMediator mediator)
protected void notify(Object obj)

BehaiviorIntepreterColleague を継承する ConcreteColleague クラスと BehaviorInterpreterMediator との協調関係を図 5.2 に示す.

5.2.4 BehaviorDefinitionParser クラス及び ObjectDefinitionParser

BehaviorDefinitionParser クラスは，XML による振る舞いの定義ファイルを読み込み，遷移規則のデータ構造である Behavior オブジェクトを生成する．ObjectDefinitionParser

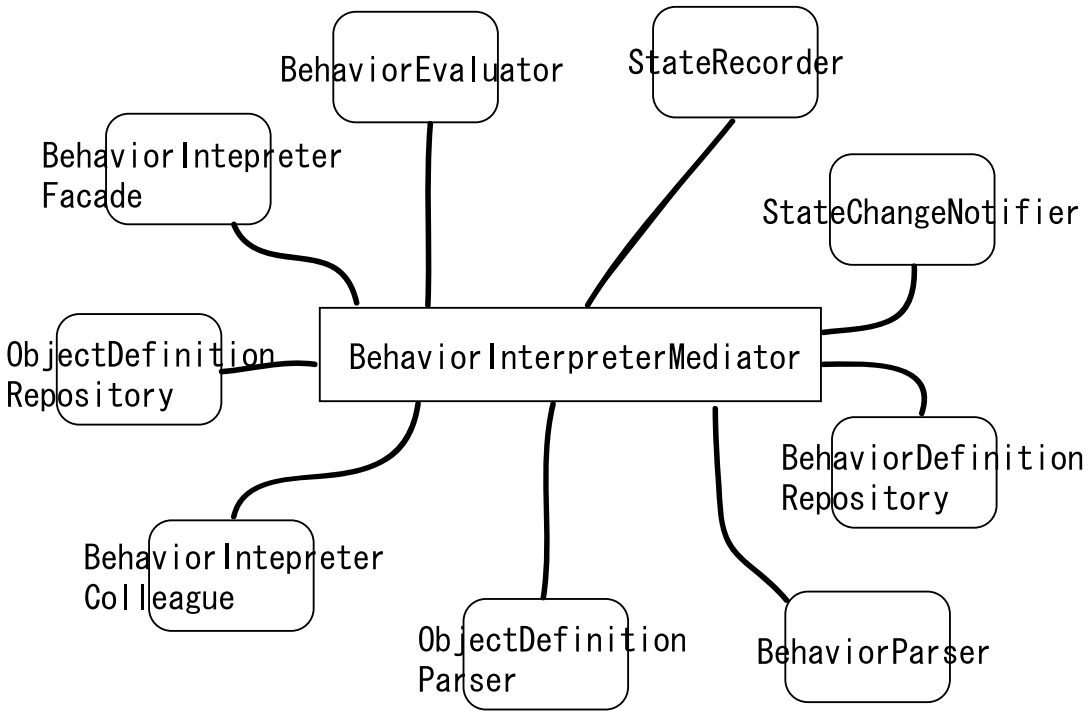

図 5.1: BehaviorIntepreter モジュールを構成するクラス群の関係

は，XML によるオブジェクトの定義ファイルを読み込み，オブジェクトの Java 言語による表現である ASARObject を生成する．また，XML ファイルのパースには DOM を利用した．以下に BehaviorDefinitionParser クラス及び ObjectDefintionParser について述べる．

BehaviorDefinitionParser

BehaviorDefinitionParser が提供するメソッドを表 5.5 に示し，以下で説明する．

表 5.5: BehaviorDefinitionParser のメソッド

public Behavior parse(String xml)
protected void encounteredTagA(Node node)
protected void encounteredTag(Node node)

parse メソッド parse メソッドは，振る舞い定義 XML ファイルの String オブジェクト表現から振る舞いの Java 言語による表現 Behavior オブジェクトに変換するインタフェースである．

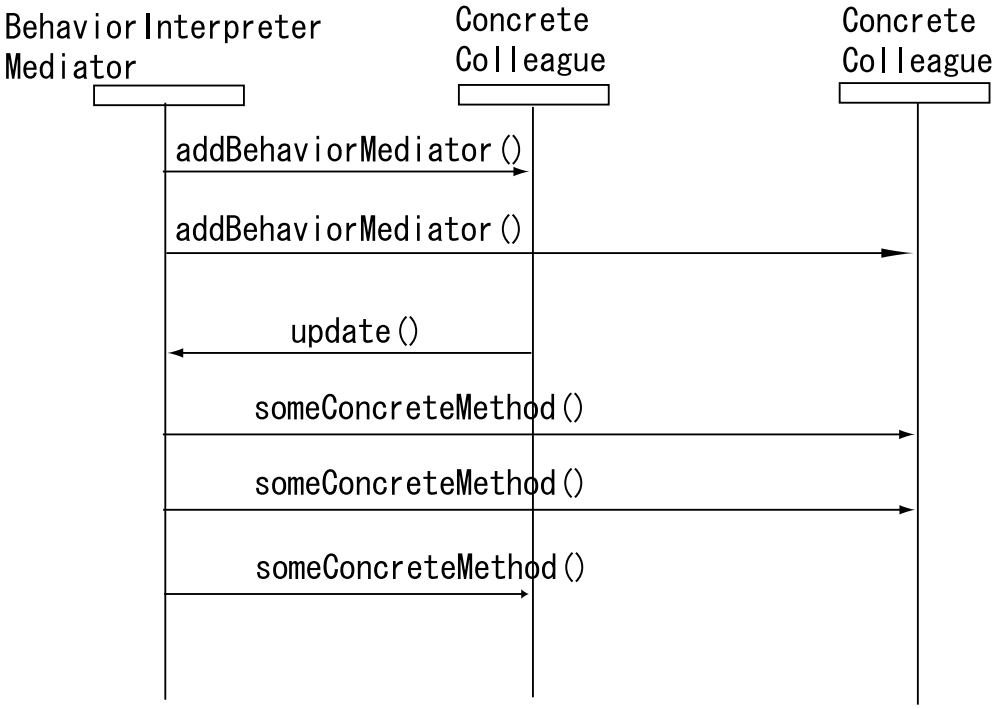

図 5.2: BehaviorIntepreter モジュールを構成するクラス群の協調関係

encounteredTagA メソッド encounteredTagA メソッドは，BehaviorTreeVisitor オブジェクトファイルを走査中に XML タグ<A>を発見するとこのメソッドが呼ばれる．このメソッドに TagA の属性及び要素を参照し Behavior オブジェクトの属性を設定するようなコードを記述する．

BehaviorDefinitionParser はインナークラスとして，BehaviorTreeVisitor クラスを持つ．BehaviorTreeVisitor オブジェクトは，DOM ツリーを再帰的に走査し，Behavior オブジェクトを生成する走査戦略を表現するクラスである．本実装では，深さ優先探索による走査を採用した．表 5.6 に BehaviorTreeVisitor クラスが持つメソッドについて述べる．

表 5.6: BehaviorTreeVisitor のメソッド

public void visit(Node node) throws ParseException
protected void dispatch(Node node)

visit メソッド DOM ノードを辿るメソッドである．再帰呼び出しされる．

dispatch XML タグに対応する処理を実行するための分岐コードが記述される.

図 5.2.4 に XML による振る舞い定義ファイルをパースするコードを掲載する.

```
//深さ優先走査
public void visit (Node node){
  dispatch(node);
  for(int i = 0; i< nodeList.getLength();i++){
    parentNode = node;
    tree.visit(nodeList.item(i));
    }
}
//タグの出現時の処理の分離
protected void dispatch(Node node){
   String tagName = node.getNodeName();
      if(tagName.equals(BehaviorParser.TAG_ISPL)){
        parser.encounteredTagISPL(node);
      }else if(tagName.equals(BehaviorParser.TAG_ENTITY)){
      .
      .
      .
      }else{
      }
}
```

図 5.3: XML ファイルのパースコード

Behavior クラス

Behavior クラスは，BehaviorInterpreter 内クラスの振る舞いを表現するクラスである.

ObjectDefinitionParser クラス

BehaviorDefinitionParser が提供するメソッドを表5.7 に示す．ObjectDefinitionParser の構成は，BehaviorDifinitionParser の構成と同様に，TreeVisitor が再帰的に DOM ツリーを走査し，ASARObject を生成する.

表 5.7: ObjectDefinitionParser のメソッド

public ASARObject parse(String xml)
protected void encounteredTagA(Node node)
protected void encounteredTagB(Node node)

ASARObject クラス

ASARObject クラスは，BehaviorInterpreter モジュール内でオブジェクトを表現するクラスである．表 5.8 に ASARObject クラスのメソッドを示す.

表 5.8: ASARObject のメソッド

public Token setState(String stateName)
public Token setState(String stateName, Token[] tokens)
protected Method getMappedEntranceAction(String stateName)

setState メソッド ASARAbstractObject を継承した分散オブジェクトの状態を変更する．状態の変更は，ISPL により定義されたオブジェクトの状態情報から Java 言語のリフレクションを用い，状態の入場動作として定義された RMI リモートメソッドを呼び出す.

リフレクションを利用した setState(String stateName) の実装を 5.2.4 に示す.

5.2.5 BehaviorEvaluator クラス

BehaviorEvaluator クラスは，StateChangeNotifier クラスから StateChangeEvent を受け取ると，状態遷移規則を評価・実行を行う．状態遷移規則のトリガの評価は，以下の 3 つの項目の評価し，遷移先を決定する.

状態成立の評価 オブジェクトの状態が成立しているかどうか

状態の同時成立評価 複数の状態が同時に成立しているかどうか

状態の遷移系列の評価 ある遷移系列で状態が遷移したかどうか

ガードの評価 排他的状態及び状態間を流れるデータを参照する論理式の評価

```
Class clazz = RMIClassLoader.loadClass(ipAddr, objName);

public Token setState(String stateName){
  Method m = this.getMappedEntranceAction(stateName);
  Object o = m.invoke(asarRemoteObj, null);
  return o;
}
protected Method getMappedEnctanceAction(String stateName)
    throws NoSuchMethodException{
  Methods[] m = clazz.getMethods();
  for(int i=0;i<methods.length;i++){
    String mName = m[i].getName();
    if(nName.equals(stateName)){
      return n[i];
    }
  }
  throw new NoSuchMethodException("入場動作として定義されたメソッドが存
在しません. ");
}
```

図 5.4: Java リフレクション機構によるメソッドの動的呼び出し

これらトリガの評価は，StateRecorder に委譲する．評価が真であれば，状態を遷移させる．ASAR Abstract Object の状態の遷移は，状態の入場動作である RMI リモートメソッドを呼び出すことで実現する．実際の ASARAbstractObject モジュールに対するメソッド呼び出しは，ASARObject の setState メソッドに委譲する．

5.2.6 BehaviorDefinitionRepository クラス

BehaviorDefinitionRepository クラスは，XML による振る舞い定義ファイルの Java 言語による表現である Behavior オブジェクトを格納するクラスである．

5.2.7 ObjectDefinitionRepository クラス

ObjectDefinitionRepository クラスは，XML によるオブジェクト定義ファイルの Java 言語による表現である ASARObject オブジェクトを格納するクラスである．

5.2.8 StateChangeNotifier クラス

StateChangeNotifer クラスは，ASARAbstractObject の状態遷移通知を受け取り，BehaviorEvaluator に ASARStateChangeEvent を送信することで状態変更を通知する．また，StateRecorder へ StateChangeEvent を状態遷移履歴として保存する．

5.2.9 StateChangedEvent クラス

StateChangedEvent クラスは，ASARAbstractObject モジュールの状態遷移を表現する BehaviorInterpreter モジュールにおける内部表現である．このクラスは，ASARStateChangeEvent クラスのラッパークラスであり，状態の履歴管理のためのイベントタイムスタンプフィールドを持つ．

5.2.10 StateRecorder クラス

StateRecorder クラスは，状態遷移履歴を保持するクラスである．状態履歴は StateChangeEvent オブジェクトの集合として表現される．また，StateRecorder は，振る舞いの定義におけるトリガの評価を行う．表 5.9 に StateRecorder クラスのメソッドを示す．

表 5.9: StateRecorder のメソッド

public void insertStateChangedEvent(StateChanedEvent evt)
public State setState(String stateName, Token[] tokens)
protected Method getMappedEntranceAction(String stateName)

5.2.11 ASARStateChangeEvent クラス

ASARStateChangeEvent クラスは，ASARAbstractObject モジュールから BehaviorInterpreter モジュールへ流れる状態遷移イベントである．ASARStateChangeEvent のメソッドを表 5.10 に示す．また，このオブジェクトは，java.io.Serializable インタフェースを実装し，RMI リモートオブジェクト間で転送される．

getSourceObject メソッド ASAR オブジェクトの参照を返す．

getSourceState メソッド 遷移元の状態名を返す．

getCurrentState メソッド 遷移先の状態を返す．

getToken メソッド 状態からの出力を取得する．

表 5.10: ASARStateChangeEvent のメソッド

public ASARAbstractObject getSourceObject()
public String getSourceState(String stateName)
public String getCurrentState(String stateName)
public Token getToken()

5.3 ASAR Abstract Object モジュールの実装

ASAR Abstract Object モジュールとは，ASAR システムにおける分散オブジェクトが共通して持つ機能を実装している．共通して持つ機能とは，状態の変更の通知機構，オブジェクト定義ファイルの管理，RMIRegistry 及び HTTP プロトコルにおける GET メソッドである．これらは，第 5.3.1 項で述べる ASARAbstractObject クラスで表現する．

5.3.1 ASARAbstractObject 抽象クラス

ASARAbstactObject クラスは，ASAR における分散オブジェクトの提供者によって，実装のフレームワークとなる．フレームワークとして，ASAR における分散オブジェクト状態の変更通知機能を持つ．また，本実装において分散オブジェクトシステムとして RMI を採用してる．従って，ASARAbstractObject クラスは，RMI 関連の機能として，rmiregistry 及び HTTP プロトコルの GET を持つ必要がある．これらは，j2sdk1.4.0 の機能として提供されている．表 5.11 に本抽象クラスが持つメソッドを示す．

表 5.11: ASARAbstractObject のメソッド

public void setBehaviorInterpreter(BehaviorInterpreterFacade facade)
throws RemoteException
public String getObjectDefinition() throws RemoteException
public String getCurrentState() throws RemoteException
protected void setCurrentState(String stateName)
protected void setToken(Token t)
proteccted void notify()

また，ASARAbstractObject を継承して分散オブジェクトを定義するアプリケーション開発者は，分散オブジェクトの状態変更を BehaviorInterpreter に通知するために，setCurrentState メソッド及び setToken メソッドを使う．

5.3.2 Token 抽象クラス

Token クラスは，状態間を流れるデータを表現するための抽象クラスである．BehaviorInterpreter モジュールで Token 抽象クラスで表 5.12 の抽象メソッドを実装することで，遷移規則の評価時に状態間を流れるデータの評価する遷移規則の記述が可能となる．

表 5.12: Token の抽象メソッド

public abstract boolean isBiggerThan(String msg)
public abstract boolean isSmallerThan(String msg)
public abstract boolean euqals(String msg)
public abstract boolean isBiggerThan(int val)
public astract boolean isSmallerThan(int val)
public abstract boolean equals(int val)

5.3.3 MulticastAdvertiser クラス

MutlicastAdvertiser クラスは，モジュール通信において唯一の RMI リモートメソッドを持たない通信である．MutilicastAdvertiser クラスは，ASAR Abstract Object が起動したときに，一定間隔おきにマルチキャストパケットを利用し自己の存在をアピールする．パケットフォーマットを表 5.13 に示す．パケットサイズは，固定長で 1024 バイトである．また，値はネットワークバイトオーダである．オペレーションコードを

表 5.13: パケットフォーマット

フィールド名	サイズ	説明
PACKET CLASS	1bytes	パケットの種類
OPCODE	1byte	オペレーションコード
IP ADDRESS	4bytes	IP アドレス
OBJECT NAME LENGTH	4bytes	オブジェクト名のサイズ
OBJECT NAME	最大 1014bytes	オブジェクト名

以下に示す．MulticastAdvertiser クラスの主なメソッドを表 5.15 に示す．

join メソッド ASARAbstractObject の起動時にこのメソッドが ASARAbstractObject から呼ばれ，一定間隔おきに告知パケットを送信し始める．

表 5.14: オペレーションコード

オペレーションコード名	値	意味
JOIN	0x00	オブジェクトの起動
LEAVE	0x01	オブジェクトの終了

表 5.15: MulticastAdvertiser クラスのインタフェース

public void join()
public void leave()

leave メソッド ASARAbstractObject の終了時にこのメソッドが ASARAbstractObject から呼ばれ，終了告知パケットを送信する．

また，MulticastAdvertiser クラスは，Thread が割り当てられ，join 告知パケットを一定間隔おきに送信する．

5.3.4 Repository クラス

Repository クラスは，オブジェクト定義ファイルを保持し，BehaviorInterpreter モジュールから定義ファイルの取得要求が ASARAbstractObject 対して発行されると，Repository クラスは保持しているファイルを返す．

5.4 ASAR Object Deployment モジュールの実装

ASAR Object Deployment は，ASAR Abstract Object モジュールの統一的管理を行うモジュールである．ASAR Object Deployment モジュールは，ASARAbstractObject モジュールの告知メッセージを監視し，ASAR Abstract Object モジュールが起動を検知する．検知時にそのリモート参照取得，保持するリポジトリを持つ．さらに，Behavior Interpreter モジュールにリポジトリの変更通知を行う．本モジュールは以下のクラスにより構成される．

5.4.1 ObjectDeployment クラス

ObjectDeployment は，InterpreterBehavior モジュールと通信を行うリモートオブジェクトである．表 5.16 に ObjectDeployment クラスが持つメソッドを示す．

表 5.16: ObjectDeployment クラスのメソッド

public ASARAbstractObject getObjectReference(String name) throws RemoteException
public ASARAbstractObject getAllObjectReference() throws RemoteException
public void setBehaviorIntepreter(BehaviorIntepreterFacade facade)
throws RemoteException
protected void notify(ASARAbstractObject obj)

表 5.17: RemoteReferenceRepository の構造

キー (rmi://xxxxxx/Name)	リモートオブジェクトの参照

5.4.2 MulticastAdvertisementReceiver クラス

MulticastAdvertisementReceiver クラスは，第 5.3.3 項で述べた告知パケットを監視し，RemoteReferenceRepository クラスにパケットの受信を通知する．

5.4.3 RemoteReferenceRepository クラス

RemoteReferenceRepository クラスは，ASARAbstractObject クラスのリモート参照を保持する．本クラスの構造は，リモート参照の URL と参照オブジェクトの組の集合である．テーブルの構造を表 5.17 に示す．

5.5 本章のまとめ

本章では，ASAR の根幹を成す振る舞いの評価・実行環境の実装を Java 言語及び RMI による分散オブジェクトシステムを用いて実装した．

第 6 章

ASARの基本性能の評価

本章では，ASARのプロトタイプ実装の基本性能の測定を行い，本研究するインタラクティブスペースでのASARの利用について考察を行う．

6.1 ASARの基本性能測定

本節では，ISPLの振る舞い定義に基づいてBehaviorInterpreterモジュール及びASARAbstractオブジェクトモジュールの基本性能について評価を行う．基本性能の評価では，ASARAbstractObject間の振る舞いの評価・実行にかかるコストを測定する．測定環境は，n個数の同型のASARAbstractObjectがそれぞれ自律的に動作している環境を想定する．自律的に動作するとは，ASARAbstractObjectの状態がランダムに他のASARAbstractObjectの状態に依存せず遷移し続けている動作をいう．測定に用いたASARAbstractObjectモジュールの動作を以下に述べる．

- トリガとなるASARAbstractObjectモジュール (TriggerObject)
  A〜Zの26個の状態を持ち50msおきにランダムに状態が変化している．
- 遷移対象となるASARABstractObjectモジュール (TargetObject)
  このオブジェクトは，一つのRMIリモートメソッドαを持ち，ASARBehaviorInterpreterモジュールからの呼び出しを待機している．

測定にに用いた振る舞いの定義は，TriggerObjectの状態の変化がTargetObjectの状態変化となる振る舞いである．また，TriggerObject数を固定し，遷移対象となるTargetObject数を1〜10まで増加させた場合 (Case1) とTargetObject数を固定し，TriggerObject数を1〜10まで増加させた場合の (以下Case2)2種類の測定を行う．両ケースにおける振る舞い定義は，TriggerObjectの任意の状態から同オブジェクトの任意の状態への遷移がTargetObjectのメソッドαの呼び出しに関連付けられている．従って，各場合の振る舞い定義数は1×26〜10×26まで増加する．振る舞いの評価・実行にかかるコスト (T) とは，トリガとなるASARAbstractObjectの遷移完了時から，遷移対象となるASARAbstractObjectの遷移完了までの時間である．この測定により振る舞い定義の増加がアプリケーションの動的適応性を実現する本システムにとって，どの程度の妨げとなっているかを調べる．以下に測定環境，測定結果について述べる．

測定環境

測定環境は，ASARAbstractObjectが配置されるホストA，及びBと振る舞いの評価実行を行うホストCで評価を行った．各ホストは以下の役割を果たす．

- **ホストA**
  1〜10個の状態遷移のトリガとなるオブジェクトが配置される
- **ホストB**
  1〜10個の状態遷移対象となるオブジェクトが配置される
- **ホストC**
  ASARBehaviorInterpreterモジュールが配置される

また，表6.1に測定環境を示し，ネットワーク構成を図6.1に示す．

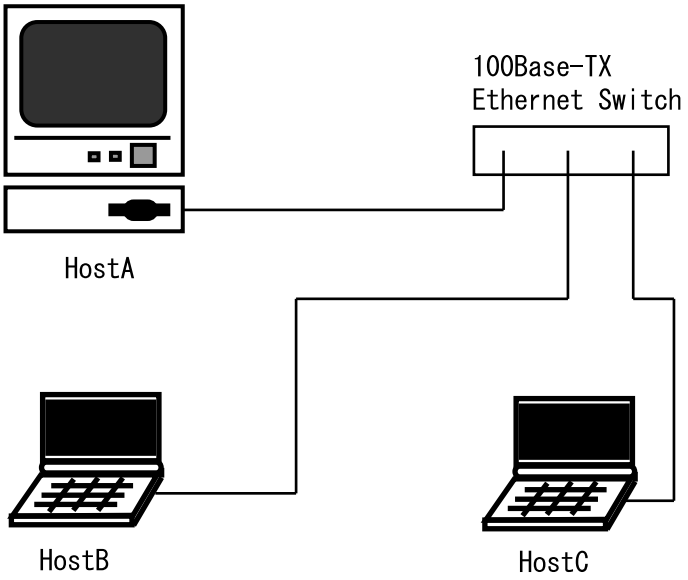

図 6.1: ネットワーク構成

表 6.1: 測定環境

項目	ホスト A(gilles)	ホスト B(roadstar)	ホスト C(Presso)
CPU	Athron Thunderbird 1GHz	Mobile Celeron 3 500MHz	Mobile Pentium 3 1.06 GHz
主記憶	256MB	196MB	1024MB
OS	Linux 2.4.18	Windows2000	WindowXP Professional
JDK	JDK1.4.0	JDK1.4.0	JDK1.4.1

測定結果

Case1 の測定結果

Case1 では，TriggerObject 数を固定し，TargetObject 数 (x) を 1〜10 まで増加させる．TriggerObject 数を固定し TargetObject 数を増加させることで，1 つのオブジェクトの状態遷移が，同時に複数のオブジェクトの状態を変更する振る舞いの性能を見る．比較対象は，TriggerObject から TargetObject の RMI リモートメソッドの直接呼び出しを行った場合である．また，Case1 が想定するアプリケーションとして，例えば，ユーザの入退室により室内のすべての情報家電を低電力モードに切り替える振る舞いが考えられる．図 6.2 に Case1 の測定結果を示す．測定結果は，それぞれ 1000 回測定した平均である．3 つのデータ系列の意味を以下に示す．

遠隔呼び出し 1 TriggerObject から TargetObject のリモートメソッドの直接呼び出し

遠隔呼び出し 2 ASAR を利用する場合

遠隔呼び出し 2-遠隔呼び出し 1 遠隔呼び出し 1 と比較した場合の遠隔呼び出し 2 のオーバヘッド

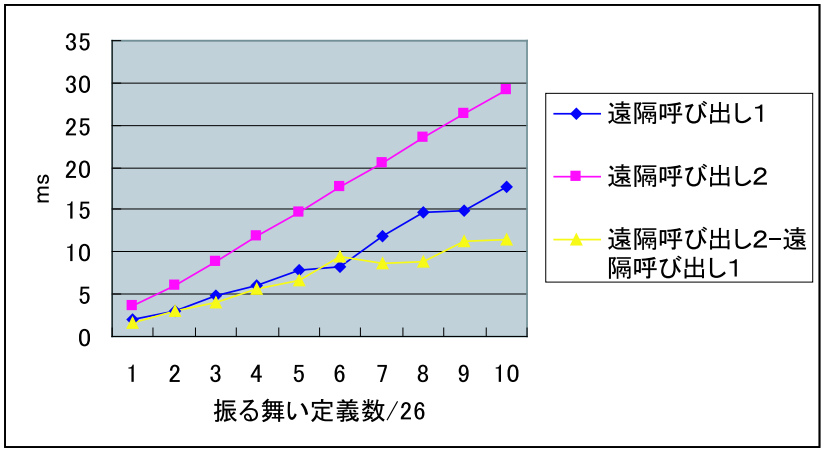

図 6.2: Case1 の測定結果

図 6.2 の遠隔呼び出し 2 の結果を T を一次曲線により近似を行うと式 6.1 で表される．

$$T = 1.56 \times x + 1.99 \tag{6.1}$$

Case2 の測定結果

Case2 では，TargetObject 数を固定し，TriggerObject 数 x を 1〜10 まで増加させる．これにより，複数の TriggerObject の状態遷移が，1 つの TargetObject の状態を変更する振る舞いの性能を見る．例えば，部屋の電灯を消灯する場合，消灯の条件として，DV デッキが再生される，ベッドに横になる，部屋から出るなどの複数の条件が考えられる．このような一つの TargetObject(電灯) と複数の TriggerObject を考慮する振る舞いが考えられる．図 6.3 に Case2 の測定結果を示す．測定結果は，それぞれ 1000 回測定した平均である．3 つのデータ系列の意味を以下に示す．

遠隔呼び出し 1 TriggerObject から TargetObject のリモートメソッドの直接呼び出し

遠隔呼び出し 2 ASAR を利用する場合

遠隔呼び出し 2-遠隔呼び出し 1 遠隔呼び出し 1 と比較した場合の遠隔呼び出し 2 のオーバヘッド

図 6.3 の遠隔呼び出し 2 の結果を一次曲線により近似を行うと式 6.2 で表される．

$$T = 0.25 \times x + 1.99 \tag{6.2}$$

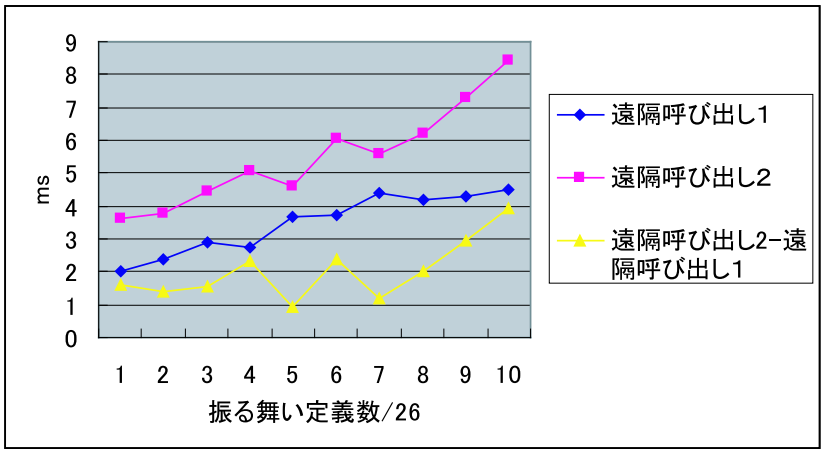

図 6.3: Case2 の測定結果

考察

式 6.1 から本プロトタイプ実装が 1 秒以内に処理可能な TargetObject 数は約 639(期待値) である．また，式 6.2 から本プロトタイプ実装が 1 秒以内に処理可能な TriggerObject 数は約 3992(期待値) である．このことから，本プロトタイプ実装は，オブジェクト数が十数個から数十個までのホームネットワークなどのかぎられた小規模ネットワークでは，応答性について充分許容できると言える．

また，Case1 は，Case2 に比べよりコストがかかっていた．式 6.1 及び式 6.2 から，TriggerObject からの状態遷移通知の処理に比べ，TargetObject のリモートメソッド α の動的呼び出し処理により多くのコストがかかっている．現在のプロトタイプ実装では，n 個の α の呼び出しは逐次的に行っている．しかし，呼び出しを非同期で並行的に行うことで Case1 の性能改善が期待の予測できる．

6.2 本章のまとめ

本章では，振る舞い定義数の増加と振る舞いの評価・実行コストの関係を測定した．測定結果により，本プロトタイプ実装は，ホームネットワーク等の小規模な環境では応答性について許容的であることを述べた．

第 7 章

関連研究との比較

本章では，他の振る舞いの動的再構成を実現するシステムムにおける振る舞い記述系に関する議論を行う．振る舞いの動的再構成を実現するシステムとして，CLAM を挙げISPL と比較する．

```
light_handle = SETUP("LIGHT");//電灯分散オブジェクトのハンドルを取得
dvd_handle = SETUP("DVDPlayer");//DVD 分散オブジェクトのハンドルを取得
WHILE(TRUE)
  dvd_state_handle = dvd_handle.INVOKE("GET_STATE");//分散オブジェクト
メ
ソッド呼び出し
  state = dvd_handle.EXTRACT();//リターン値の取得
  IF(state = "PLAYING"){
    light_handle.INVOKE("OFF");
  }ELSE{

  }
}
```

図 7.1: CLAM によるアプリケーション記述例

CLAM

CLAM は，下位記述としての分散オブジェクトの機能定義と上位記述としての分散オブジェクト間のインタラクション定義を分離することで，振る舞いの再構成を実現している．上位記述は以下のように手続言語に近い表現となっている．DVD の再生とともに部屋の電灯を消すというアプリケーションの記述を例に 7.1 に示す.

CLAM の欠点

CLAM は，メソッド呼び出しの表現をそのまま上位記述に表現している．従って，オブジェクトの状態に考慮した上記のアプリケーションを記述する場合，変数の定義によるオブジェクトの状態の管理，状態を取得するためのメソッド呼び出し及び状態を遷移させるためのメソッド呼び出しなどの様々な概念を考慮する必要がある．また，リアクティブな挙動を記述する場合，上記の記述では，WHILE 文で状態の変更を常に監視しており，コールバックを利用した事象駆動による記述ができない．事象駆動による振る舞いの記述は，事象の生起と結果の関連を記述すため，より対象に近い表現であるといえる．また，位置情報を考慮する振る舞いの記述を行う場合，例えば，TV の前に人が来た場合に，TV の電源をいれるという振る舞いを記述する場合，TV の前に人が来たという離散状態の定義を記述で行う場合，代数方程式を CLAM 言語によって記述する必要がある．代数方程式による空間的状態の定義は，対象に近い記述ではない.

CLAM の利点

しかし，CLAM の特徴は，分散オブジェクトメソッド呼び出しを上位記述においてより細かく制御できることである．例えば，ESTIMATE 呼び出しを用いることで，分散オブジェクトのメソッド呼び出しのコストに関してデータ量及び呼び出し時間を予め把握できる．また，呼び出し方法も INVOKE 呼び出しを使う場合と INVEX 呼び出しを使う場合があり，INVOKE 呼び出しは，メソッド呼び出しとリターンの間に処理を行える．INVEX 命令の場合は，図 7.3 に示すように通常のメソッド呼び出しと同様に呼び出しよリターンの間に処理を行えない．以下に INVOKE 呼び出しを使った分散オブジェクトのメソッド呼び出しの開始とリターンの間に挟まれた処理の例を 7.2 示す．

```
//INVOKE と EXTRACT の間で処理を行える．
object_handle.INVOKE("MethodName")

 ...

var = object_handle.EXTRACT()
```

図 7.2: 非同期呼び出し

```
var = object_handle.INVEX("MethodName")
```

図 7.3: ランデブー

INVOKE 呼び出しと EXAMINE 呼び出しによって，分散オブジェクトのメソッド呼び出し中の状態を細かく取得できる．以下にメソッドの呼び出し中の状態を考慮する記述例を図 7.4 に示す．

考察

CLAM の特徴は，分散オブジェクトメソッド呼び出しの意味論をそのまま上位記述系に反映し，分散オブジェクトのメソッド呼び出しの状態や呼び出しコストの事前見積を実現することで，より表現力が高い振る舞い記述系となっている．ISPL の特徴は，ユーザが観測可能な離散状態の遷移系列を定義することで振る舞いを記述するため，特に位置情報，オブジェクトの空間的状態を利用した振る舞いを記述する場合は，より対象に近く直感的である．

これらのことから，記述者をエンドユーザ，もしくはシステム管理者を想定し，振る舞いの動的再構成を目的とした場合 ISPL がより記述系として適していると言える．しかし，コンポーネント指向大規模ソフトウェア開発を行う上で，分散オブジェクト

```
object_handle = SETUP("ObjectName");
method_handle = object_handle.INVOKE("SomeMethod");
WHILE(method_handle.EXAMINE()=!DONE){
  IF(method_handle.EXAMINE()==PARTIAL){
    //呼び出し中の処理
  }ELSE IF(method_handle.EXAMINE()==ERROR){
   //呼び出し失敗時の処理
  }
}
var = object_handle.EXTRACT();//リターン値の取得
```

図 7.4: メソッド呼び出しの途中経過による処理の分岐

の機能定義と振る舞いの分離によるソフトウェアの開発効率・再利用性の向上を目的とする場合，ISPL の表現性は低く，CLAM がより適していると言える．

第 8 章

結論

8.1 今後の課題

本節では，今後の課題について述べる．

分散オブジェクトシステム非依存性

ISPL は，分散オブジェクトシステム非依存に設計した．本論文で述べた ASAR の実装は RMI を用いたが，ISPL は．設計上 CORBA 等の他の分散オブジェクトシステムにより実装することも可能である．分散オブジェクトシステム非依存性を実証するために，他の分散オブジェクトシステム上に ISPL の処理系を実装する．

記述に関する課題

振る舞いの記述は，状態の遷移系列の定義することにより記述する．しかし，現在の ISPL の仕様及び ASAR の実装では，振る舞いの定義 a 及び b においてトリガとして同一のものが指定され，かつ遷移先状態が a 及び b で異なる記述を行った場合，両者が評価・実行されるのか，片方のみが評価されるのかは，現在，BehaviorInterpreter の実装依存である．この問題を解決するために，複数の振る舞い評価の優先関係などを記述し，評価する必要がある．

より高度なプログラミング環境

エンドユーザがアプリケーションの振る舞いの動的再構成を実現するには，Visual Programming Language(VPL) や例示による実世界プログラミングによる記述が適切である．本論文で述べた ISPL による振る舞い定義ファイルを生成する VPL 環境や実

世界プログラミング環境を実現することでより，エンドユーザやシステム管理者が状況に適応的なインタラクティブスペースを実現できる．

広域分散ネットワークへの対応

広域分散ネットワークを介したオブジェクト間の振る舞いを実現するためには，セキュリティ機構が必要となる．現在 Object Deployment モジュールに登録される ASAR Abstract Object モジュールは，インターネットを越しに同期をとっていれば，すべて参照可能となっている．

8.2 本論文のまとめ

従来の振る舞いの記述系は，アプリケーション開発の効率化に焦点が充てられ，記述者も高度な技術を有すソフトウェア開発者を想定して設計されていた．従って，言語仕様も高度な記述が可能である．しかしながら，エンドユーザやシステム管理者が環境に遍在するオブジェクトの位置情報を考慮した振る舞いを記述するには適していない．本論文では，エンドユーザやシステム管理者のような既存のプログラミング言語に疎いユーザを想定した振る舞いの記述系を提案し，オブジェクトの協調動作 (振る舞い) を動的に変更可能な基盤システム ASAR における振る舞いのその実行環境の設計，実装及び評価を行った．

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．

また，慶應義塾大学政策・メディア研究科博士課程 1 年の由良淳一氏及び岩井将行氏には本論文執筆にあたって多くの御助言を賜りましたことを感謝いたします．また，本論文を書き終えることができたのは，共に執筆した友人でかつ同僚の青木俊氏，門田昌哉氏のお陰でもあります．さらに，石井かおり氏には，私の拙い語学力で書かれた英語及び日本語の文章を根気よくご指導して頂きました．志和木愛子氏には，多くの精神的な励ましを頂きました．Eunos Presso は，昼夜，雨天にかかわらず自宅と研究室の間を極めて迅速にかつ安全に私を移送してくれました．みなさまに深い感謝の意を表します．

平成 15 年 2 月 22 日
高橋 元

参考文献

[1] P. Spilling F. Arbab, I. Herman. An overview of manifold amd its implementation. *Concurency: Practice and Experience*, Vol. 5, No. 1, 1993.

[2] Masayuki Iwai, Jin Nakazawa, and Hideyuki Tokuda. Dragon: Soft Real-Time Event Delivering Architecture for Networked Sensors and Appliances. In *The 7th International Conference on Real-Time Computing System and Applications(RTCSA2000)*, pp. 425–432, December 2000.

[3] Masayuki Iwai, Jin Nakazawa, and Hideyuki Tokuda. Flexible Distributed Event-Driven Programming Framework for Networked Appliances and Sensors. In *3rd International Symposium on Distributed Objects and Applications (DOA'01) Short Papers Proceedings*, pp. 61–68, September 2001.

[4] J. Nakazawa, T. Okoshi, M. Mochizuki, Y. Tobe, and H. Tokuda. VNA: An Object Model for Virtual Network Appliances. In *IEEE 2000 International Conference on Consumer Electronics*, pp. 364–365. IEEE Consumer Electronics Society, June 2000.

[5] 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.

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

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

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

[9] Sun Microsystems Inc. JavaSpaces Service Specification, version 1.1, October 2000. http://www.sun.com/jini/specs/js1_1.pdf.

[10] Sun Microsystems Inc. *Jini Architecture Specification*, October 2000. http://www.sun.com/jini/specs/jini1_1.pdf.

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

[12] Dong Zhou, Karsten Schwan, Greg Eisenhauer, and Yuan Chen. *JECho - Supporting Distributed Collaborative Applications with Java Event Channels*. College of Computing Georgia Institute of Technology, 2000.

[13] 青木崇行 村瀬正名 松宮健太 中澤 仁 西尾 信彦 高汐 一紀 徳田英幸. Smart Furniture:Improvising Ubiquitous Hot-spot Environment . 情報家電コンピューティング研究会. 情報処理学会, November 2002.