Vol.2010-SE-169 No.10
2010/7/23

プロセス代数によるアーキテクチャ記述の容易化に関する研究

田 口 博 一[†1]　豊 島 真 澄[†1]

筆者はプロセス代数によるアーキテクチャ記述を改善するためのツール EasyADS を設計した．本ツールの目的は記述性を向上させることによって，アーキテクチャ記述者の負担を改善することと，ステークホルダが要求する情報に素早くたどり着くことを支援することである．本論文では，EasyADS の設計と，ツールを用いたアーキテクチャ記述容易化の手法について説明する．

Research of Easy-description of Architecture by Process Algebra

Hirokazu Taguchi[†1] and Masumi Toyoshima[†1]

We designed tool for improvement of description which name is “EasyADS”. The purposes of designing tool are to reduce of architecture descripter’s task and appear information that is required by stakeholders as soon as possible. In this paper, we explain design of EasyADS and method of easy-decription of architecture by the tool.

1. はじめに

アーキテクチャの段階で，品質やコストの厳密な見積りを行うことが近年の開発で求められている[5]．なぜならば，この見積もりの結果は，プロジェクトを左右する判断を行う貴重な客観的材料を表すことができるからである．以前から，アーキテクチャレベルで厳密にこれらを見積もる手法としてプロセス代数が注目されてきた.[2][8]
プロセス代数がアーキテクチャ記述で注目されてきた理由は以下の通りである．

(1) 形式記述であり，コンピュータを用いた計算が可能である[1][7]．
(2) プロセス代数を用いたアーキテクチャ設計に有用な計算処理技法がすでにいくつか提案されている[6]．
(3) 設計の詳細が決まっていなくても記述可能であり，アーキテクチャ記述に都合がいいこと．
(4) 記述者が状況に応じた抽象レベルで記述することが可能である．

上記の理由から，プロセス代数はシステムのアーキテクチャを記述するのに非常に便利な技術であるといえる．しかしながら，プロセス代数には記述が複雑化し，結果として熟練者の育成や，記述に多くの時間が必要となってしまうという課題がある．我々はこの課題を克服するための手法とツールを設計した．本研究では，この手法とツールの設計を紹介する．
本論文の構成であるが，第 2 章では課題について詳しく説明する．第 3 章では，課題に対するアプローチと本研究で開発したツールである EasyADS の概要について述べる．第 4 章では，EasyADS の設計を説明し，第 5 章では，実際に実例を用いた説明を行う．

2. 課　　題

これまでプロセス代数とアーキテクチャについてさまざまな研究が行われてきた．しかしながら，プロセス代数はスピーディな記述が実現されているとは言えない．これまでグラフィカルな記法の導入などによって，記述性の改善が行われてきた．ところがまだ以下の要因が記述性を阻害している．

(1) 粗結合な記述を記述者に強要すること
(2) ある程度の詳細なモデルを記述しなければステークホルダが必要とする情報を導けないこと

この記述性を阻害する要因についてより詳しく説明する．

2.1 粗結合な記述による課題

プロセス代数はプロセス間の通信に注目して記述するため，粗結合な記述であると言える．この特徴はシステムの構成要素を追加したり削除できるためアーキテクチャ記述において大変便利である．しかし，疎結合の記述だけでシステムを設計することは熟練者と時間が必要である．

2.2 記述が詳細化する問題

アーキテクチャはモデルを使った計算とモデルそのものの見える化の双方に期待されている．特にモデルを使った計算の結果はステークホルダの素早い判断を補助する．しかしながら，計算を行うためのモデルを書ためにはどうしても詳細なモデルが必要である．本問題

†1 北九州市立大学院国際環境工学研究科
Graduate School of International Environment Technology, The University of Kiakyushu

© 2010 Information Processing Society of Japan

Vol.2010-SE-169 No.10
2010/7/23

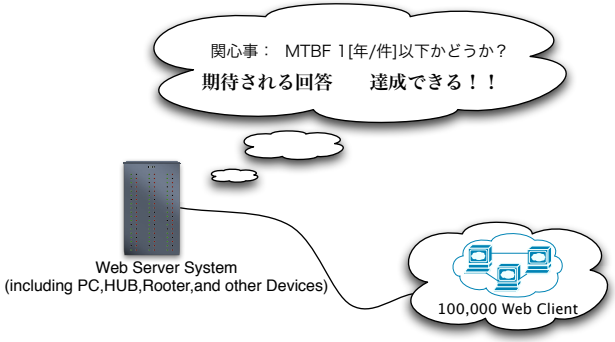

図 1　抽象レベル 1 のアーキテクチャ設計例
Fig. 1　Example of Architecture Design(Abstruction Level 1)

は，例を用いて説明する．
10 万台のクライアントからのアクセスに耐える必要があるウェブサーバをこれから構築したいとする．最初の設計としては，図 1 のようにサーバーとその外部に位置するクライアントとの関係を表したものになるだろう．この最初の設計をレベル 1 のアーキテクチャ設計とする．この図からすぐに「10 万台の負荷に耐えられるような設計になっているか？」という関心事が生まれる．
しかし，この関心事を客観的に計算するためのモデルは図 1 のレベルをプロセス代数化したところで役に立たない．PC, ルータなどのデバイスや配線を含めたもう数レベル詳細なモデルを記述して初めて求められる結果が得られるようになる．
このようにステークホルダが期待する回答を得るには，より詳細なモデルを書かなければならない．

2.3　課題のまとめ

記述が詳細になってしまうことと，粗結合の記述問題が組み合わされて，記述に時間が掛かってしまう．アーキテクチャのステークホルダは，素早くシステムの状況を把握し適切な判断を行う必要があるので，記述に時間がかかることは問題である．
我々は，本課題を克服することを目指したツール EasyADS を設計した．

3.　アプローチ

本研究では，プロセス代数の厳密さを失わずに，記述の複雑な部分を隠蔽し，より使いやすくするためのシステム「EasyADS(Easy Architecture Description System)」を提案する．前に述べたように，本研究ではアーキテクチャ記述を容易化することが目的である．しかしながら，容易化されたという基準は数値化することが難しい．よって，まず事例を従来のアーキテクチャ記述法に則って記述し，それを改善するという手法を用いた．

3.1　EasyADS 概要

EasyADS の概要を以下に示す．

(1)　**特定の目的に特化した記述システム**
筆者は，耐故障性やパフォーマンスの演算などのカテゴリごとに個々の記述パターンが必要だという立場である．本システムは，プロセス代数の記述者がカテゴリ毎にパターンの登録が可能である．

(2)　**アーキテクチャインタフェースの導入**
アーキテクチャインタフェースの詳細は第 3.2 節で説明するが，このインタフェースに沿った入力によって，記述の容易化を実現する．

(3)　**言語に依存しない中間モデルの利用．**
既存のアーキテクチャ言語は，独自に検証可能なツールを持っているものもある．筆者はこれらの能力を生かすため，厳密に型付けを行い中間モデルを定義する．

(4)　**プロセス代数の専門家とアーキテクチャ記述者を結びつけるためのシステム**
筆者は，プロセス代数の専門家とアーキテクチャ記述者を分けて考え，それぞれに異なる機能を提供するよう EasyADS を設計した．

3.2　アーキテクチャインタフェース

EasyADS はプロセス代数を自動生成することによって，記述の容易化を行う．本システムは自動生成の規則そのものを登録するシステムを有している．このとき自動生成のために必要な入力情報が「アーキテクチャインタフェース」である．このアーキテクチャインタフェースは，プロセス代数の専門家によって記述・登録され，アーキテクチャの記述者は定義されたインタフェースに沿ってデータを入力することで生成されたプロセス代数を得ることができる．これが基本的な EasyADS の記述容易化のメカニズムである．

© 2010 Information Processing Society of Japan

Vol.2010-SE-169 No.10
2010/7/23

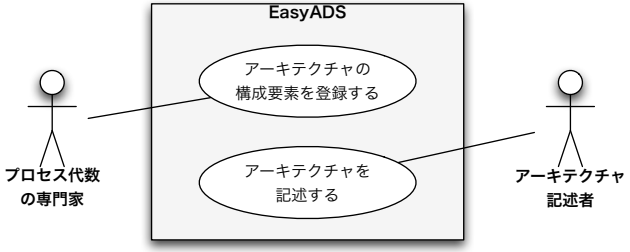

図 2 EasyADS ユースケース図
Fig. 2 Usecase Diagram of EasyADS

4. EasyADS の設計

4.1 EasyADS ユースケース

まず，ツールのユースケースを紹介し，ツールがどのような利用を想定されているか明らかにする．

4.1.1 ア ク タ

EasyADS で登場するアクタを以下のように定義する

(1) **アーキテクチャ記述者**
ステークホルダの関心事に合わせた情報を提供するためのモデルを記述する

(2) **プロセス代数の専門家**
プロセス代数の知識が豊富な専門家

4.1.2 ユースケース図・記述

図 2 にユースケース図を示し，またシナリオを説明する．ユースケース図が示すようにアーキテクチャ記述者と，プロセス代数の専門家の役割が明確になるように EasyADS を設計している．

(1) **アーキテクチャの構成要素を登録する**
- (a) プロセス代数の専門家は，パターン化に関する要件を獲得する．
- (b) プロセス代数の専門家は，要件を元にアーキテクチャインタフェースと生成パターンを記述する
- (c) プロセス代数の専門家は，EasyADS にパターンとアーキテクチャインタフェースをを登録する．

(2) **アーキテクチャを記述する**
- (a) アーキテクチャの記述者は，検索エンジンからアーキテクチャを検索する．
- (b) アーキテクチャインタフェースを取得し，実体化に必要な情報を得る．
- (c) アーキテクチャインタフェースに沿って情報を入力し，プロセス代数記述を取得する．[*1]．

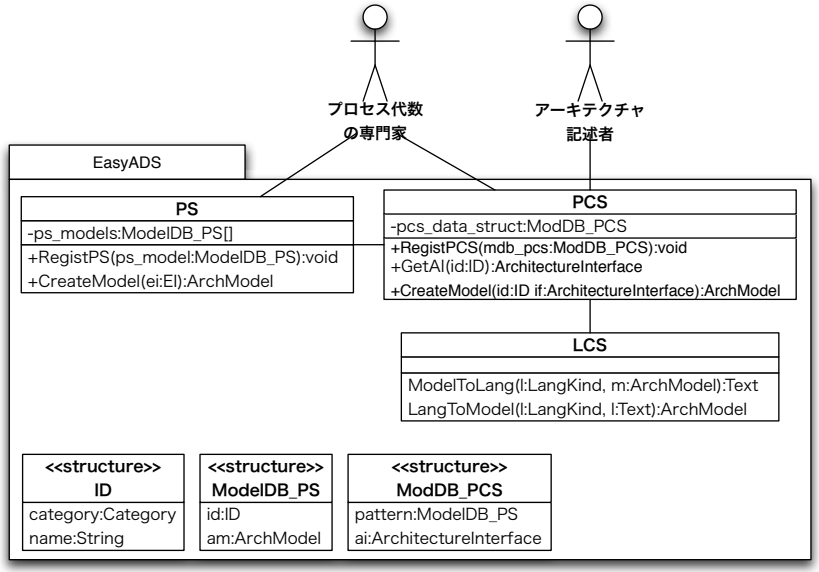

図 3 EasyADS 簡易クラス図
Fig. 3 Class Diagram of EasyADS(part)

4.2 設 計 概 要

EasyADS は以下の 3 つのサブシステムから構成される．クラス図を図 3 に示す．

- Primitive Subsystem(PS)
- Primitive Combinator Subsystem(PCS)

*1 実際のツールでは，グラフィカルな記述として表示される．

© 2010 Information Processing Society of Japan

- Language Creator Subsystem(LCS)

4.3 PS

アーキテクチャを構成する要素であるアーキテクチャプリミティブを登録・管理するためのシステムである．このアーキテクチャプリミティブは，アーキテクチャとしての意味を持たないが，システムを構成する上で必要不可欠なサブシステムのことである．PCS が PS に対してアーキテクチャプリミティブの生成と取得を指令する．この指令システムはコンピュータ間の通信を想定しているため，Web サービスの仕組みを参考として設計したているが，システムの内部関数であることから，図 3 では省略している．PS へのプリミティブ登録者は，プロセス代数の専門家である．

PS に所属する関数と属性を説明する．

- 属性
  - **ps_models:ModelDB_PS[]**
    PS が管理するアーキテクチャプリミティブを格納するための属性である．
- 関数
  - **RegistPS(mdpds:ModelDB_PS):void**
    PS にアーキテクチャプリミティブを登録するための関数である．ModelDB_PS 型は，図 3 にあるようにモデルをカテゴライズして特定するための ID 型と，モデル本体から成る．登録者はプロセス代数の専門家に相当　する人物で，別途グラフィカルなインタフェースなどを介して利用される．
  - **CreateModel(el:ElementInterface):ArchModel**
    GetEI 関数で取得した ElementInterface を入力として ArchModel 型のモデルを作成する．このモデルは PCS に提供され，PCS は PS から得たモデルを組み合わせて記述者が望むアーキテクチャのモデルを生成する．

4.4 PCS

PS で述べたアーキテクチャプリミティブの組み合わせをパターン化して定義し，このパターンと，アーキテクチャインタフェースの入力によってアーキテクチャ記述を生成するサブシステムである．

図 3 に示すように，機能そのものは PS と似ているが，プロセス代数の専門家だけでなくアーキテクチャ記述者が関われるように設計されていることが大きく異なる．組み合わせの定義については，プロセス代数の専門家が行う．また，アーキテクチャの記述者はアーキテクチャインタフェースを介して生成されたモデルを取得することができる．アーキテクチャインタフェースの型は，アーキテクチャ記述者が扱えるよう Int,String のような基本的なデータ型で行う．

- 属性
  - **pcs_data_struct:ModelDB_PCS**
    PS が管理するアーキテクチャプリミティブを格納するための属性である．
- 関数
  - **RegistPCS(mdb_pcs:ModelDB_PCS):void**
    PCS にアーキテクチャ生成のパターンを登録するための関数．登録に必要なものはパターンとアーキテクチャインタフェースである．
  - **GetAI(i:ID):ArchitectureInterface**
    生成したいアーキテクチャモデルを指定する ID 型の入力に対して，生成に必要な ArchitectureInterface 型を返すための関数．PCS は，この関数で取得した ArchitectureInterface 型を入力として，アーキテクチャモデルの生成を行う．
  - **CreateModel(id:ID if:ArchitectureInterface):ArchModel**
    GetAI 関数で取得した ArchitectureInterface を入力として ArchModel 型のモデルを作成する．このモデルは LCS によってプロセス代数の文法記述に変換することが可能であり，また，他のシステムによって直接処理を行うことも可能である．

4.5 LCS

PCS,PS は生成したモデルの種別に応じて厳密に型付けされた ArchModel 型が生成される．これは，将来的にモデルを機械な互換性を考えた措置であるが，ArchModel はプロセス代数記述を反映したモデルであり，プロセス代数記述に変換可能である．この変換を行うサブシステムが LCS である．

- 関数
  - **ModelToLang(l:LanguageKind, m:ArchModel):Text**
    言語の種類と，PCS によって生成された ArchModel 型のモデルを入力として，指定された言語で書かれたアーキテクチャ記述を生成する．
  - **LangToModel(l:LanguageKind, l:Text):ArchModel**
    言語を入力して解析されて出力された ArchModel 型のモデルを出力する関数である．

© 2010 Information Processing Society of Japan

Vol.2010-SE-169 No.10
2010/7/23

基本データ型		
$BaseType$	$=$	$INT \mid REAL \mid STRING \mid BOOL$
$ConnectionType$	$=$	$IN \mid OUT$
$ConnectionName$	$=$	$STRING$
$RepeatOrEnd$	$=$	$REPEAT \mid END$
組み合わせ型		
$ArchModel$	$=$	$ArchModel\ list \mid Parallel\ list$
$Parallel$	$=$	$Exclusive\ list$
$Exclusive$	$=$	$Sequential\ list$
$Sequential$	$=$	$\{Connection\ list,\ RepeatOrEnd\}$
$Connection$	$=$	$\{ConnectionName,\ ConnectionType,\ TransData\ list\}$
$TransData$	$=$	$Connection\ list$
		$BaseType$

図 4 ArchModel 型の定義
Fig. 4 Definition of ArchModel Type

4.6 型 定 義

本設計では筆者が定義した型が多く使われている．ここでは主要な型を説明する．

4.6.1 ArchModel 型

ArchModel 型は以下のように定義された型である[*1]． 図 4 に示すように ArchModel 型は並行プロセスを示す Parallel, 排他処理を示す Exclusive, 逐次処理を示す Sequential, プロセス間通信を示す Connection 型の組み合わせから構成された型である．Connection 型を通して，TransData 型を通信できる．TransData 型は ArchModel 型を扱えない．これは，まだ高階化ができていないことを意味しており，今後の課題としている．
また，本定義の構造は結合度を表している．実際のプロセス代数処理において，排他処理 Exclusive は複数の逐次処理 Sequential list から選択して実行するものであるし，また，並行処理型 Parallel は複数の排他処理型 Exclusive list を並行動作させるものである．直観的には Parallel 型は複数の逐次処理型 Sequential を並行動作させるような感覚があるが，これは，Parallel 型が 1 つの逐次処理型を持つと考えればよい．

*1 Connection 型と，TransData 型は循環型定義になっており，実装するには工夫が必要だった．しかし，ここでは説明のしやすさのために循環型定義を許可している．

$$ArchitectureInterface \quad = \quad BaseType$$

図 5 ArchitectureInterface 型の定義
Fig. 5 Definition of ArchitectureInterface

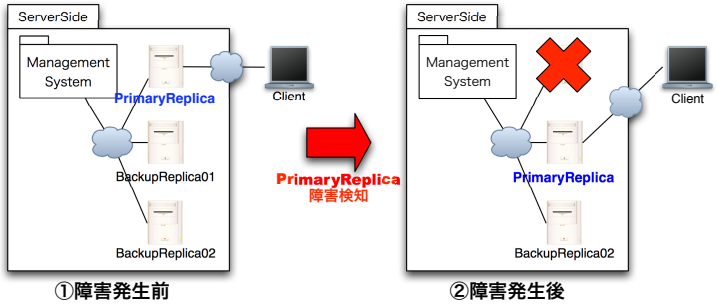

図 6 受動的多重化概要図
Fig. 6 Outline of Passive Replication

4.6.2 ArchitectureInterface 型

ArchitectureInterface は，アーキテクチャ記述者の視点でアーキテクチャを記述することを支援するためのインタフェースである．よって，ArchModel 型が持つような特別な型を扱わない．よって，型定義は図 5 の通りである．

5. 事例による実証

本章では，本システム理解しやすくするために実例を用いてツールの説明を行う．事例として受動的多重化を用いる．受動的多重化は，クラッシュ故障に耐性を持つ汎用的な耐故障技術の一つである．

5.1 仕 様

5.1.1 概 要

概要図を図 6 に示し，仕様の説明を行う．サーバ側 (Server Side) はレプリカされた複数のマシンと，そのレプリカ群を管理するためのマシンから構成されている．レプリカのうち一つがプライマリレプリカに指定されており，このプライマリレプリカだけがクライアント

© 2010 Information Processing Society of Japan

と通信を行うことができる．その他のレプリカはバックアップレプリカと呼ばれスリープ状態にある．
もし，プライマリレプリカが故障して検知できたとすれば，バックアップレプリカの一つが新たなプライマリレプリカとなって故障したレプリカに変わりクライアントと通信を行う．こうして，クライアントはサーバー側に起きた故障の影響が受けにくくなる．
また，レプリカの管理を行うシステムが本設計には必要であるから，これを ManagementSystem と名付けた．

5.1.2 ManagementSystem

前節で，レプリカの管理を行う ManagementSystem について紹介した．本システムは設計上必要であるから内部構成を簡単に説明する．本システムは，以下のようなサブシステムから構成される．

- 故障検出器 (FD)
- 安定記憶装置 (SS)
- レプリカ生成器 (RC)

FD は，ハートビートの受信を行う．送信者はプライマリレプリカである[*1]．もしハートビートの受信が途絶えれば故障の検出と見なし，検出信号を出力する．
SS は，プライマリレプリカの状態を保持する．プライマリレプリカに異常があれば保持している状態を新たに指定されたプライマリレプリカへ送信して処理を続行させる．
RC は，故障検出器の故障検出信号を受けてスリープ状態にあるバックアップレプリカを起こしてプライマリレプリカに変更させる．ステーブルストレージの保持するプライマリレプリカの状態を起きたばかりのプライマリレプリカへ送信するのも RC の役割である．

5.2 アーキテクチャインタフェース

アーキテクチャ記述者に視点へ移して考える．彼らのアーキテクチャ設計に関する関心事を以下のように仮定する．

- どれだけのレプリカがあれば，耐故障要件を満たせるのか調べたい．

これは，本システムに対する要件としては，純粋にレプリカの数によってモデルを生成し，この生成されたモデルから耐故障要件を満たせるかどうか計算するという流れになる．システムに対する入力は「レプリカの数」であり，INT 型である．

*1 故障検出器はハートビート受信サブシステムを持ち，プライマリレプリカはハートビート送信サブシステムを持つ必要がある

PR のサブシステム		
$BKSender(backup)$	$=$	$\overline{backup}.BKSender(backup)$
$HBSender(hb)$	$=$	$\overline{hb}.HBSender(hb)$
$NB(in, out)$	$=$	$in.\overline{out}.NB(in, out)$
PR		
$PR(backup, hb, in, out)$	$=$	$NB(in, out) \mid HBSender(hb) \mid BKSender(backup)$

図 7 プライマリレプリカのπ計算の記述
Fig. 7 pi-calculus description of Primary Replica

5.3 PS への登録

アーキテクチャ記述者がアーキテクチャを記述する前に，プロセス代数の専門家は，アーキテクチャインタフェースからモデルを生成するための情報を登録しておく必要がある．
PS はアーキテクチャを構成する要素を登録するための EasyADS サブシステムである．本仕様を満たすために必要で，かつ，PS へ登録する必要があるサブシステムとしては，FD，SS，RC，によって構成される ManagementSystem(MS) 及び，プライマリレプリカ (PR)，バックアップレプリカ (BR) である．全てを説明することは紙面の都合上不可能なので，PR のみ説明する．
PR は，プライマリレプリカとしてクライアントからの要求に応えるための振る舞い (NB) だけでなく，FD へのハートビートの送信機能 (HBSender)，と定期的に SS へ状態を送信する機能 (BKSender) の 3 つの構成要素から成るシステムである．これらの構成要素は互いに並行動作する．
π計算で記述すると，図 7 の記述になる．

LCS は図 7 を構文解析し，ArchModel 型として登録を行う．LCS が構文解析を行い，ArchModel 型として登録する様子を図 8，9 に示す．この変換は図 3 の関数 LangToModel が行い，変換された ArchModel 型を関数 RegistPS を用いて登録する．太文字は図 4 で定義した型を表している．

まず図 8 で示されているようにπ計算の記述を構文解析しやすい形式に変換して型定義を行っていく．例えばπ計算における in は，”in”という名前のついたプロセスへの入力信号を示しており，かつ，通信されるデータは存在しないことを示している．よって，LCS は in をコネクション名を示す Connecti onName 型, コネクションの形式を示す ConnectionType 型, 通信されるデータを示す TransData 型に分割する．分割された型は組み合わせ型へ変

© 2010 Information Processing Society of Japan

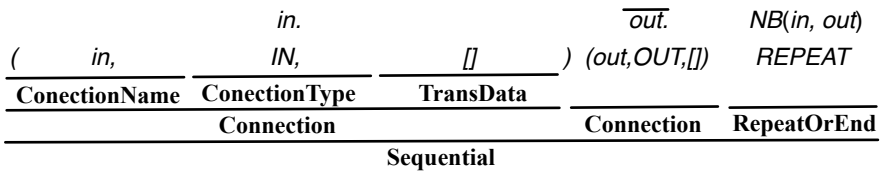

図 8 LCS による NB の構文解析
Fig. 8 Parsing of NB using LCS

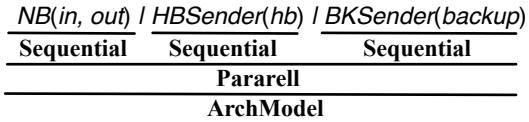

図 9 LCS による PR の構文解析
Fig. 9 Parsing of PR using LCS

換されていく．こうして最終的に図 9 に示されるように ArchModel 型となり，本システムに意味付けされた形で登録され処理できるようになる．
同様にして BR,MS, 及び MS の構成サブシステムである FD,SS,RC もまた ArchModel 型として FC へ登録される．

5.4 PCS への登録

仕様より，本事例におけるアーキテクチャインタフェースは「レプリカの数」である．これを変数 rn で INT 型とする．レプリカの数が変われば当然ながら構造と振る舞いも変化する．rn=3 のときは 2 つのレプリカの故障に耐えられるが，rn=4 のときは 3 つのレプリカの故障に耐えられる．これはアーキテクチャの振る舞いが変化しているのであるから，モデルもまた，この変化を実現できるように作らなければならない．幸運にもプロセス代数の中でπ計算はこの生成規則を容易に実現できる仕組みを持っている．筆者はこの仕組みを利用して，PCS を設計した．本研究で用いたπ計算の仕組みは，次節で説明する．

5.4.1 π計算の有用な特徴

PCS を説明するために，π計算の説明を行っておく．π計算を全て説明することは紙面の都合上できないので，本研究で必要な部分のみ簡単に説明する．

- 名前によるコネクションの確立
  π計算はコネクションに付けられた名前が同名であればコネクション関係にあることを示す．よって，リネームするだけで容易にコネクション関係を変更することができる．
- 名前の置換による具象化
  π計算は名前の置換によって具象化する．
- モビリティ
  π計算はコネクションを送ることができる．名前付けによるコネクションの確立と合わせると実行時にコネクションを確立したり切断して構造を変化させることができる．

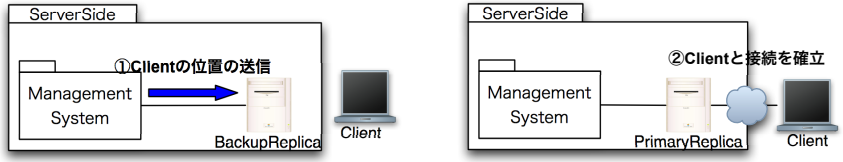

図 10 PR におけるモビリティのメカニズム
Fig. 10 Mobility Mechanism in PR

5.4.2 BK の設計

BK に注目してアーキテクチャインタフェースから構造が構築されるための仕組みを記述する．
BK は通常スリープ状態にあり，MS からの wakeup 命令によってクライアントと通信可能になる．これを実現するのが，π計算の「モビリティ」である．BK は MS から図 10 のようにクライアントの位置を実行中に知る．
また，MS は複数存在するあるバックアップレプリカを識別し，それぞれのレプリカに適切な通信を行わなければならないが，これも図 10 と同様の技術と，自動的に名前を付ける機能を加えることで解決していることを付け加えておく．
このようにして作成した記述を PS に RegistPS 関数を用いて登録する．バックアップレプリカはこのように動的に構造を組み替えるための仕組みを記述することで，アーキテクチャインタフェースとパターンによって具象化されるための準備を行う．

5.4.3 アーキテクチャインタフェースとパターンの登録

BK が PCS を強く意識した記述を行うことで，PCS を用いてモデル生成を行うための準

© 2010 Information Processing Society of Japan

備が整った．π計算はモデルの構造がコネクションの名前の付け方によって簡単に変化できるという便利な特徴があるため，これを用いてパターンの実装を行った．作成したパターンとレプリカ数を表す INT 型をアーキテクチャインタフェースとして PCS の RegistPCS 関数にを用いて登録する．

5.4.4 アーキテクチャ記述の生成

登録されたパターンは ID 型によって名前付けされている．記述者はこれを検索して GetAI 関数によってアーキテクチャインタフェースを取得する．取得したインタフェースにはレプリカ数を INT 型で入力するように指定してあり，記述者は，この指定にしたがって入力すると，PCS の CreateModel 関数と LCS の ModelToLang 関数によって指定された言語が出力される．

こうして，アーキテクチャ記述者は INT 型の入力のみで受動的多重化のアーキテクチャモデルを取得することができた．筆者は，実際に実装を行い，これまで説明した操作を行うことで，π計算の言語を出力させ，π計算のツール MobilityWorkBench[3] を用いて筆者の意図した通りの動作することを確認した．

6. 今後の課題とまとめ

本研究はプロセス代数の専門家とアーキテクチャ記述者を明確に分け，専らアーキテクチャ記述者にとって記述を容易化するシステムを構築した．グラフィカルな記号を用いて作成するツールまだ実装中であり，グラフィカルインターフェースの紹介しゃ今後ツールの正しさを含めた裏付けを示さなければならないが，アーキテクチャインタフェースによって記述を支援するメカニズムは提案することができた．

しかし，アーキテクチャ記述者にとってよいシステムは，多くの課題を浮き彫りにさせた．まず，現状ではプロセス代数の専門家の負担は確実に増すことが明らかになった．記述だけでなくモデル生成などを気にしながら記述することをシステムが強要するからである．しかしながら，筆者はプロセス代数をアーキテクチャ記述として実践的に使うに当たって，ステークホルダの判断の支援を第一に考えるべきという立場から本システムを提案したわけであり，そのたたき台として大きな役割を本設計は果たした．

今後はより多くの記述を行い，本システムの設計に沿って，アーキテクチャ記述者だけでなくプロセス代数の専門家もまたより記述を楽にできるような，デバッガ，モニタ，入力支援などの仕組みを提案していく予定である．

また，より複雑なアーキテクチャを記述するために，高階記述を可能にしなければならないことが分かっている．残念ながら，現状の EasyADS は高階化の実現はされていないため，今後は高階化の実現を目指し，より複雑なシステムの設計をより容易に記述することを可能とする予定である．

7. 関 連 研 究

本研究の基礎は，π計算によるプロセス代数記述である[7]．シンプルだが，高い記述性と先に述べた特徴を備えている．本ツールは高階化の記述を可能にすることが目標の一つだが，これについては，π計算を高階関数化して作られたπ-ADL があり[1] 実現可能性はある．アーキテクチャの記述については OMG によって提唱されている MARTE[4] がある．MARTE は UML プロファイルが公開されており，グラフィカルな記述による入力支援やパラメータを用いた解析も可能となっている．

参 考 文 献

1) Flavio O.: π-ADL:An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures, ACM Software Engineering Notes, Vol 29, No. 4 (2004).
2) Flavio, O.: π-method: A Model-Driven Formal Method for Architecture-Centric Software Engineering, ACM SIGSOFT Software Engineering Notes, Vol 31, No. 3, (2006).
3) Mobility Workbench. http://www.it.uu.se/research/group/mobility/mwb
4) OMG. UML MARTE. http://www.omgmarte.org/
5) Paul C., Rick K. and Mark K.: Evaluating Software Architectures, Addison Wesley, (2006).
6) Radu M. and Flavio O.: π-AAL:An Architecture Analysis Language for Formally Specifying and Verifying Structual and Behavioual Properties of Software Architectures, ACM Software Engineering Notes, Vol. 31, Num. 2, (2006).
7) Robin M.: Communicating and Mobile Systems:The Pi-Calculus, Cambridge University Press, (1999).
8) Swapna S.G.: Architecture-Based Software Reliability Analysis: Overview and Limitations, IEEE Transactions on Dependable and Secure Computing, Vol. 4, Num. 1, (2007).

© 2010 Information Processing Society of Japan