ソフトウェア工学 126－1
（2000．3．9）

# 通信システム仕様の合成支援環境の設計と実現

氏家 有[†] 古屋 幸治[†] 高橋 薫[†] ベッドB. ビスタ[††]

[†]仙台電波工業高等専門学校 [††]岩手県立大学ソフトウェア情報学部

**あらまし** プロトコルを設計するための最も有望な方法の一つに合成法がある．この方法を用いることにより，小さく簡潔で扱いやすいプロトコルを合成して，より大きく複雑なプロトコルを設計し検証できるようになる．合成法のなかでも，筆者らは，仕様記述言語LOTOSを用い，サービス仕様とプロトコル仕様を同時に合成するアプローチを提案してきた．本論文では，その合成法に基づき，LOTOSで表現されるサービス仕様とプロトコル仕様を構成するためのソフトウェア支援環境の設計と実現について述べる．本支援環境では，選択的，順次的，並列的な仕様合成支援機能をユーザ(設計者)に提供する．合成されたサービス仕様とプロトコル仕様は観測等価である．

**キーワード** プロトコル，サービス，仕様，合成，LOTOS．

# Design and Implementation of a Support Environment for Composing Communication System Specifications

Yu Ujiie[†], Koji Furuya[†], KaoruTakahashi[†] and Bhed Bahadur Bista[††]

[†]Sendai National College of Technology

[††]Faculty of Software & Information Science, Iwate Prefectural University

**Abstract** One of the most important techniques for designing protocols is a compositional technique. Using this technique, a large and complex protocol can be designed and verified by composing small and simple protocols. Among compositional techniques, we have proposed an approach in which service specifications and protocol specifications are simultaneously composed using the specification language LOTOS. In this paper, we design and implement a support environment for constructing service and protocol specifications, based on the approach. The system supports alternative, sequential and parallel composition of the specifications. The composed service specification and protocol specification are observationally equivalent.

**Keywords** Protocol, Service, Specification, Composition, LOTOS.

# 1 はじめに

プロトコルは，あるタスクを達成するため，共通な規則の下で互いに通信するいくつかのエンティティから構成される．そのようなプロトコルを正しく的確に設計するためには，プロトコルを適切にモデル化し，形式的に記述することが重要である．

プロトコルの設計過程では，サービスの定義段階とプロトコルの仕様化段階がある．サービス仕様はプロトコルが提供すべきサービスを仕様化したものであり，プロトコルの内部メカニズムは隠され，関係するSAP(サービスアクセス点) で起こりうるアクションの時間順序が仕様中に反映される．一方，プロトコル仕様には，SAP 毎にサービスを実現するエンティティ(プロセス) 群と，それらエンティティ間のメッセージ交換が反映される．

以上のような関係から，サービス仕様をプロトコル仕様に分解することで，プロトコルを設計しようとする方法の研究がある [1][2]．一方，部分機能的なプロトコル仕様群を独立に設計し，新たなプロトコルを設計するのにそれらを合成しようとする方法の研究もある [3][4]．後者では従来，通信システムの重要な論理構成要素であるサービスについては何も考慮しておらず，そのため，筆者らは，サービスとプロトコルを同時に考慮したプロトコル合成アプローチを新しく提案した [5]．このアプローチでは，各プロトコル仕様を，与えられたサービス仕様からLangerak[6] の方法を用いて観測等価分解 (弱双模倣等価分解) して導出し，それら仕様群を合成して (サービス仕様群の合成とプロトコル仕様群の合成)，再び等価なサービス仕様とプロトコル仕様を得ている．仕様は仕様記述言語LOTOS[7] で記述され，合成形態にはLOTOSのオペレータに対応して，選択的合成，順次的合成，割込み的合成，並列的合成の四つがある．

本論文では，上述のプロトコル合成アプローチに基づき，LOTOS で表現されるサービス仕様とプロトコル仕様を構成するためのソフトウェア支援環境の設計と実現について述べる．本支援環境では，ユーザから二つのサービス仕様を入力し，それぞれ，それと弱双模倣等価なプロトコル仕様を自動導出する．そしてユーザの指示により，サービス仕様対およびプロトコル仕様対に対して，選択的，順次的，並列的な合成を行う．これには筆者らの合成アプローチを用いる．合成されたサービス仕様とプロトコル仕様は弱双模倣等価性を保存し，サービスとプロトコルの間に一貫性が保証される．

以下本論文では，まず，プロトコルの合成アプローチについて要約する．そして，そのアプローチに基づいた合成支援環境の機能要件，ソフトウェア設計，実装について述べる．併せて，簡単で具体的な使用例を例示する．最後にまとめと今後の課題を示す．

# 2 プロトコル合成アプローチ

本節では，合成支援環境の基礎として用いるプロトコル合成アプローチ [5] を必要な範囲で概観する．より詳細については文献 [5] を参照されたい．

サービスおよびプロトコルの仕様は仕様記述言語 LOTOS を用いて記述するため，まずLOTOS について説明する．次に，合成のためのサービスとプロトコルのモデルを述べ，最後に，合成アプローチを示す．

## 2.1 LOTOS

LOTOS は，分散システムを形式的に記述するために開発された仕様記述言語である．LOTOS で仕様化されるシステムは，互いに相互作用するいくつかのプロセスから成る．プロセスとは，$\mathbf{i}$ として表現される内部アクションを行ったり，ゲートと呼ばれる同期点での通信アクションを通して他のプロセスと通信したりする抽象的な実体である．プロセスの動作式は，そのアクションの時間順序の観点から表現される．プロセスあるいはシステムの動作式を構成するのに様々なLOTOS オペレータが存在する．特に本論文で使用するオペレータの一覧を表 1 に示す．

LOTOS の要素には動作式の他にデータの表現があるが，本論文では主に動作式を取り扱う．従って，以下に定義する遷移システムを取り扱う．まず，$Act$ をアクションの集合，$Act^*$ を $Act$ 中のアクションの系列の集合，$\mathbf{i}^*$ を 0 個以上の$\mathbf{i}$ アクションの系列とする．プロセス $P$ に対し，$Act(P)$ を，$P$ が実行可能なアクションの集合とする．

**定義 1** 遷移システム $L$ は次の 4 項組である．

表 1: LOTOS オペレーター覧

| 名　前 | オペレータ |
|---|---|
| 無動作 | **stop** |
| 正常終了 | **exit** |
| アクションプレフィクス | a; P　,　i; P |
| インスタンシエーション | P |
| 選択 | P1 [] P2 |
| 順次合成 | P1 >> P2 |
| 並列合成 (同期並列) | P1 \|[G]\| P2 |
| 並列合成 (非同期並列) | P1 \|\|\| P2 |
| 割込み | P1 [> P2 |

$$L = \langle S, A, T, s_0 \rangle$$

ここで，$S$ は状態の集合，$A$ は $Act$ の部分集合，$T \subseteq S \times A \times S$ は遷移関係，$s_0 \in S$ は $L$ の初期状態である．□

プロセスの動作式が与えられると，定められた遷移規則の適用により，対応する遷移システムが導出される．遷移システムによって与えられる操作的意味に基づき，弱双模倣関係を以下に定義する．定義において，$t = a_1 \cdots a_n \in (Act - \{\mathbf{i}\})^*$ のとき，$s \stackrel{t}{\Rightarrow} s'$ は $s \stackrel{\mathbf{i}^* a_1 \mathbf{i}^* a_2 \mathbf{i}^*}{\longrightarrow} \cdots \stackrel{\mathbf{i}^* a_n \mathbf{i}^*}{\longrightarrow} s'$ を意味する．

**定義 2** $L_1 = \langle S_1, A_1, T_1, s_{10} \rangle$ と $L_2 = \langle S_2, A_2, T_2, s_{20} \rangle$ を遷移システムとする．$(s_1, s_2) \in R$ と任意の $t \in (Act - \{\mathbf{i}\})^*$ に対して，次の条件 1 と 2 が成り立つとき，関係 $R \subseteq S_1 \times S_2$ を弱双模倣関係という．

1. $s_1 \stackrel{t}{\Rightarrow} s_1'$ である $s_1'$ が存在するならば，$s_2 \stackrel{t}{\Rightarrow} s_2'$ で $(s_1', s_2') \in R$ である $s_2'$ が存在する．
2. $s_2 \stackrel{t}{\Rightarrow} s_2'$ である $s_2'$ が存在するならば，$s_1 \stackrel{t}{\Rightarrow} s_1'$ で $(s_1', s_2') \in R$ である $s_1'$ が存在する．

ある弱双模倣関係 $R$ に対して $(s_{10}, s_{20}) \in R$ ならば，$L_1$ と $L_2$ は弱双模倣等価 (観測等価) であるといい，$L_1 \approx L_2$ と書く．また，$L_1$ がプロセス $P$ の遷移システムで，$L_2$ がプロセス $Q$ の遷移システムであるならば，$P$ と $Q$ は弱双模倣等価 (観測等価) であるといい，$P \approx Q$ と書く．□

弱双模倣等価は，二つのシステムの動作が外部観測上区別できないことを表す概念である．

簡単化のため，以下では，動作式を等式の形で書く．

## 2.2 サービスとプロトコルのモデル

図 1(a) に示すように，サービスは SAP での動作だけが観測されるブラックボックスである．従って，SAP で実行されるイベントの順序によってサービスを仕様化する．各 SAP を数値で指標付け，SAP で実行されるイベントを $event\_name^{node}$ の形式で表現する．ここで，$event\_name$ はサービスプリミティブであり，$node$ はそれが生起する SAP である．

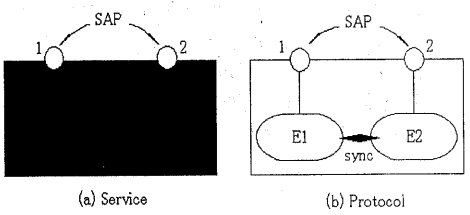


図 1: サービスとプロトコルのモデル

各サービス仕様 $S$ および対応するプロトコル仕様はアクションプレフィクス形式で与えられると仮定する．すなわち，$I$ をインデックス集合，$A_i$ をプロセス名あるいはアクションプレフィクス形式の動作式として，

$$S = \Sigma a_i^1; A_i \mid i \in I$$

の形式で表されるとする．例えば，$I = 1, 2$ に対して，$S = a_1^1; A_1 [] a_2^1; A_2$ で，$A_1 = b_1^2; \mathbf{stop}$，$A_2 = b_2^2; \mathbf{stop}$ のとき

$$S = a_1^1; b_1^2; \mathbf{stop} [] a_2^1; b_2^2; \mathbf{stop}$$

のように表現される．

プロトコルは図 1(b) に示すようにホワイトボックスである．いかにサービスが提供されるかが示され，各ノードに存在し互いに通信しあう通信エンティティの動作がプロトコル仕様として規定される．従って，プロトコル仕様は $E_1 |[sync]| E_2$ のように表現される．ここで，$E_i$ はノード $i$ $(1 \leq i \leq 2)$ のエンティティであり，ゲート $sync$ で同期することにより $E_j$ $(1 \leq j \leq 2, i \neq j)$ と互いに通信する．

本合成アプローチでは，$E_i$ $(1 \leq i \leq 2)$ の仕様は分解法 [6] を用いてサービス仕様から導出されると仮定する．この分解法では，サービス仕様のノード $i$ で順次的に生起するイベントは $E_i$ で順次的に編成し，ノード $i$ のイベントが

ノード $j$ ($j \neq i$) のイベントで後続されるときはノード $i$ と $j$ の間のイベント実行の変化を $E_i$ と $E_j$ の同期アクションにより表現する (逆も同様). 例えば, 上のサービス仕様については以下のエンティティ仕様が導出される.

$$
\begin{aligned}
E_1 = \ & a_1^1; sync!ma_1^1; sync!mb_1^2; \mathbf{stop} \\
[] \ & a_2^1; sync!ma_2^1; sync!mb_2^2; \mathbf{stop} \\
E_2 = \ & sync!ma_1^1; b_1^2; sync!mb_1^2; \mathbf{stop} \\
[] \ & sync!ma_2^1; b_2^2; sync!mb_2^2; \mathbf{stop}
\end{aligned}
$$

以下では, サービス仕様 $S_1$ に対するエンティティ仕様をノード毎に $E_1$ と $E_2$, サービス仕様 $S_2$ についてもノード毎にエンティティ仕様 $F_1$ と $F_2$ と仮定する. また簡単化のため, イベント $a$ についてのメッセージを $ma$ などと表記する.

## 2.3 合成アプローチ

ここでは, サービス仕様とプロトコル仕様の合成アプローチを具体的に示す. 合成形態には, LOTOS オペレータに対応して四つ提案されているが, 合成支援環境で用いる選択的, 順次的, 並列的な合成について示す. 合成アプローチの概要は以下の通りである.

$S_1 \approx \mathbf{hide}\ G_1\ \mathbf{in}\ E_1|[G_1]|E_2$ であるような $S_1$ と $E_1|[G_1]|E_2$, 同様に, $S_2 \approx \mathbf{hide}\ G_2\ \mathbf{in}\ F_1|[G_2]|F_2$ であるような $S_2$ と $F_1|[G_2]|F_2$ が与えられたとき, $S_1 * S_2 \approx \mathbf{hide}\ G_1 \cup G_2\ \mathbf{in}\ (E_1 * F_1)|[G_1 \cup G_2]|(E_2 * F_2)$ であるように, $S_1$ と $S_2$ を, また, $E_1|[G_1]|E_2$ と $F_1|[G_2]|F_2$ を合成する. ここで, $*$ は合成形態に対応するオペレータである.

### 2.3.1 選択的合成

選択的な機能を持つ仕様の合成には, 以下の合成アルゴリズムを適用する.

**アルゴリズム 1**

(* ①初期アクションが同じノード *)
$S_1 = \Sigma a_i^1; A_i | i \in I,\ S_2 = \Sigma b_j^1; B_j | j \in J$
のとき
$S_1 [] S_2 \Longrightarrow (E_1 [] F_1)|[sync]|(E_2 [] F_2)$
(* ②初期アクションが異なるノード *)
$S_1 = \Sigma a_i^1; A_i | i \in I,\ S_2 = \Sigma b_j^2; B_j | j \in J$
のとき
$S_1 [] S_2 \Longrightarrow (E_1 [] F_1)|[sync]|(E_2 [] F_2)|[a_i^1, b_j^2]|C$
$C = (\Sigma a_i^1; C_i | i \in I) [] (\Sigma b_j^2; C_j | j \in J)$
(* ③一つのノードでの内部アクション *)
$S_1 = \Sigma \mathbf{i}; a_i^1; A_i | i \in I,\ S_2 = \Sigma b_j^2; B_j | j \in J$
のとき
$S_1 [] S_2 \Longrightarrow ((E_1 [] F_1)[> F_1)|[sync]|((E_2 [] F_2) |[a_i^1, b_j^2]|C$
$C = (\Sigma \mathbf{i}; a_i^1; C_i | i \in I) [] (\Sigma b_j^2; C_j | j \in J)$
(* ④両方のノードでの内部アクション *)
$S_1 = \Sigma \mathbf{i}; a_i^1; A_i | i \in I,\ S_2 = \Sigma \mathbf{i}; b_j^2; B_j | j \in J$
のとき
$S_1 [] S_2 \Longrightarrow ((E_1 [] F_1)[> F_1)|[sync]|((F_2 [] E_2) [> E_2)|[a_i^1, b_j^2]|C$
$C = (\Sigma \mathbf{i}; a_i^1; C_i | i \in I) [] (\Sigma \mathbf{i}; b_j^2; C_j | j \in J)$ □

②〜④において, 単純な選択合成では弱双模倣性が保存されないため, コントローラプロセスを導入している. アルゴリズム中, $C_i$ と $C_j$ は, それぞれ, $A_i$ と $B_j$ が何であるかに依存して決まる. もし $A_i$ が **stop** ならば $C_i$ は **stop**, $A_i$ が **exit** ならば $C_i$ は **exit**, $A_i$ が $S_i$ ならば, つまり, 再帰ならば $C_i$ は $C$ である. $A_i$ がアクションを持つならば, $A_i$ が **stop**, **exit** あるいは $S_i$ であるまでスキップされる. $C_j$ と $B_j$ についても同様である.

### 2.3.2 順次的合成

順次的に結合される機能を持つ仕様の合成には, 以下の合成アルゴリズムを適用する. 但し, $S_1$ は $B[]\mathbf{exit}$ の形をしていないと仮定する.

**アルゴリズム 2**

$S_1 = \Sigma a_i^1; A_i | i \in I,\ S_2 = \Sigma b_j^1; B_j | j \in J$
のとき
$S_1 >> S_2 \Longrightarrow (E_1 >> F_1)|[sync]|(E_2 >> F_2)$ □

このアルゴリズムにおいて, $S_1$ と $S_2$ の初期アクションがどのノードで生起するかは問題にならない.

### 2.3.3 並列的合成

並列的に結合される機能を持つ仕様の合成には, 以下の合成アルゴリズムを適用する. 但し, $G = Act(S_1) \cap Act(S_2)$ とする.

**アルゴリズム 3**

(* ① $S_1$ と $S_2$ が同期 *)
$S_1 = \Sigma a_i^1; A_i | i \in I,\ S_2 = \Sigma b_j^1; B_j | j \in J,$
$G = Act(S_1) \cap Act(S_2) \neq \emptyset$
のとき

$S_1|[G]|S_2 \Longrightarrow (E_1|[Act(E_1) \cap Act(F_1)]|F_1)$
$|[sync]|(E_2|[Act(E_2) \cap Act(F_2)]|F_2)$
(* ② $S_1$ と $S_2$ が非同期 *)
$S_1 = \Sigma a_i^1; A_i | i \in I$, $S_2 = \Sigma b_j^1; B_j | j \in J$,
$G = Act(S_1) \cap Act(S_2) = \emptyset$
のとき
$S_1|||S_2 \Longrightarrow (E_1|||F_1)|[sync]|(E_2|||F_2)$ □

# 3 合成支援環境

本節では，前節の合成アプローチに基づき，2つのサービス仕様を入力し，プロトコル仕様へ分解し，それらを合成する合成支援環境の設計と実装について述べる．

## 3.1 設計

### 3.1.1 機能要件

以下に，合成支援環境の具備すべき機能要件を整理する．

1. **サービス仕様入力** ユーザからサービス仕様を入力させる．サービス仕様は，2つ入力される．
2. **構文解析** 入力したサービス仕様がLOTOSのアクションプレフィクス形式に基づいているか解析する．解析によって構文が正しくないと判断された場合は再度仕様の入力を行わせる．
3. **プロトコル仕様への分解** 入力した2つのサービス仕様を文献[6]の方法で，それぞれプロトコル仕様へ分解する．分解されたプロトコル仕様は，元のサービス仕様と弱双模倣等価である．
4. **仕様表示** 入力した2つのサービス仕様と分解した2つのプロトコル仕様を表示する．
5. **合成方法の選択** 選択的，順次的，並列的の3つの合成方法から1つを選択可能とさせるインタフェースを提供する．
6. **合成** 合成支援環境のメイン機能である．サービス仕様とプロトコル仕様を前節で述べた合成アプローチに基づいて，それぞれ合成し，それらの結果を表示する．
7. **ファイル管理** サービス仕様，プロトコル仕様をファイルとして管理する．
8. **プロトコルモデル表示** プロトコルのモデルを図式的に示すとともに，合成の流れの中，ユーザが現在どの段階にあるかを強調する．

### 3.1.2 機能制限と設定

以下，合成支援環境の制限，及び設定を整理する．

1. **サービス仕様の入力**
   アクションについてはアクション名，ノードという形で入力する．ノードは1もしくは2を {} にいれてアクション名のすぐ後ろに入力するということを設定する．例えば，a {1} はアクション名がaで，それがノード1で生起することを示す．
   テキスト形式で保存したサービス仕様ファイルの読み込み／保存を可能にする．
2. **構文解析について**
   アクションプレフィクス形式ではあるがサービス仕様として意味がない仕様に付いては構文が正しくないと判断する．例えば **stop[]exit** などは正しくないと判断される．

### 3.1.3 合成の流れ

以下に合成の流れを説明し，図2に示す．

第一段階は，ユーザがサービス仕様をテキスト形式で入力することである．入力したサービス仕様はファイルとして保存することが可能である．又，作成したファイルの読み込みも可能であり，編集することも出来る．このサービス仕様入力を2回行う．

第二段階は，これらの2つのサービス仕様を構文解析することである．正しく構文解析がされている場合は，次の段階へ進み，されていない場合は再びサービス仕様入力段階へと戻り，仕様を再度入力する．

第三段階は，構文解析が正しくされた2つのサービス仕様を2つのプロトコル仕様へ分解し，テキスト形式で表示する．分解されたプロトコル仕様は，保存が可能である．

第四段階は，2つのサービス仕様と2つのプロトコル仕様を合成するための合成方法を選択することである．選択的，順次的，並列的の3つの合成方法から1つを選択する．

第五段階は，合成されたサービス仕様とプロトコル仕様をそれぞれテキスト形式で表示する．これらの仕様もまたファイルとして保存することが可能となる．

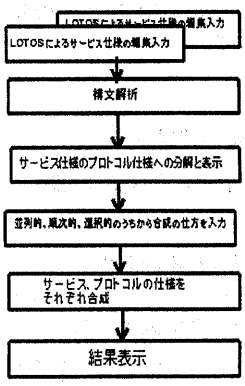


図 2: 合成の流れ

### 3.1.4 ソフトウェア構造

図 3 にソフトウェア構造を示す.

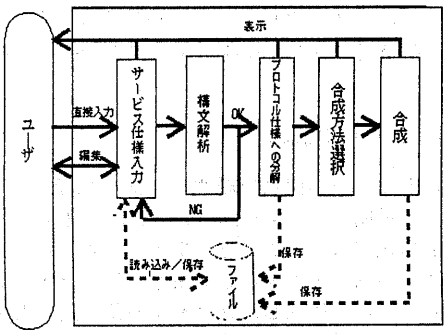


図 3: 支援環境のソフトウェア構造

## 3.2 実装

上記機能要件, 及び制限, 設定の下, 合成支援環境を VisualBasic を用いて実装を行っている. 以下に示す簡単な 2 つのサービス仕様を例にとり, 本合成支援環境の実装内容と動作について記述する.

$$
\begin{aligned}
S_1 = \ & ConReq^1; ConInd^2; \\
& ConRes^2; ConCnf^1; \mathbf{stop} \\
S_2 = \ & ConReq^2; ConInd^1; \\
& ConRes^1; ConCnf^2; \mathbf{stop}
\end{aligned}
$$

$S_1$ は, ノード 1 でコネクション確立要求後, ノード 2 でコネクション確立指示, コネクション確立応答をし, 最後にノード 1 でコネクション確立確認をして終了するコネクション確立サービスの例である. $S_2$ は, ノード 2 側から確立要求を行うということを除いて $S_1$ と同じである.

1. サービス仕様 $S_1$ を入力する (図 4). ここでは, キーボードから直接入力したり, ファイルからテキスト形式の仕様の入力を行う. 入力した仕様の保存も可能である. また, プロトコルモデルも同時に表示される (図 5). この表示はモデル中のノードやエンティティなどが点滅し, 現在合成支援がどの段階にあるかを示している. このプロトコルモデルは以下で説明するすべての段階で表示されている. 仕様を入力してメニューバーをクリックすると入力したサービス仕様がアクションプレフィクス形式であるかどうか判断する. 図 4 で入力されているサービス仕様はアクションプレフィクス形式であるため, 次の段階に進むことが出来る. アクションプレフィクス形式でない場合は, エラーの表示がなされ, 仕様の再入力をすることになる.

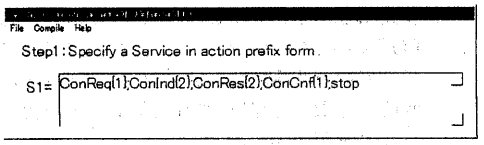


図 4: サービス仕様入力 1

2. 2 つめのサービス仕様 $S_2$ についても同様である (図 6).
3. 入力された 2 つのサービスを分解法に基づいて分解する (図 7). $S_1$ と分解された $P_1$, $S_2$ と分解された $P_2$ はそれぞれ弱双模倣等価である.

$$
\begin{aligned}
S_1 &\approx \mathbf{hide}\ sync\ \mathbf{in}\ P_1 \\
&\equiv \mathbf{hide}\ sync\ \mathbf{in}\ E_1 |[sync]| E_2
\end{aligned}
$$

$$
\begin{aligned}
S_2 &\approx \mathbf{hide}\ sync\ \mathbf{in}\ P_2 \\
&\equiv \mathbf{hide}\ sync\ \mathbf{in}\ F_1 |[sync]| F_2
\end{aligned}
$$

ここで, $E_1$, $E_2$, $F_1$, $F_2$ はいかの通りで

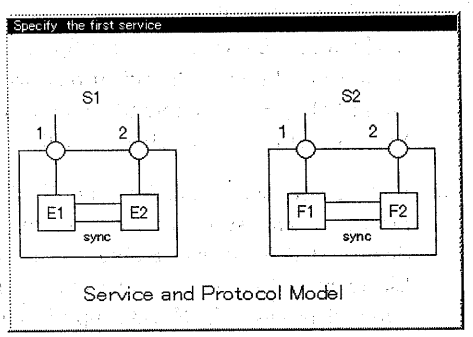


図 5: プロトコルモデル

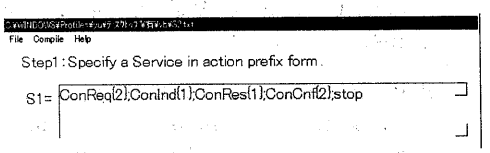


図 6: サービス仕様入力 2

ある.

$$
\begin{aligned}
E_1 = \ & ConReq^1; sync!ConReq^1; \\
& sync!ConRes^2; ConCnf^1; \mathbf{stop} \\
E_2 = \ & sync!ConReq^1; ConInd^2; \\
& ConRes^2; sync!ConRes^2; \mathbf{stop} \\
F_1 = \ & sync!ConReq^2; ConInd^1; \\
& ConRes^1; sync!ConRes^1; \mathbf{stop} \\
F_2 = \ & ConReq^2; sync!ConReq^2; \\
& sync!ConRes^1; ConCnf^2; \mathbf{stop}
\end{aligned}
$$

4. 選択的，順次的，並列的の 3 つの中から合成方法を選択する (図 8) . この例で，$S_1$ と $S_2$ は選択的な機能であるため，選択的合成方法を選択する（図 8 の CHOICE）.
5. 前節で述べた合成法に基づき合成を行い，結果を表示する (図 10). 合成されたサービス仕様は $S_1[]S_2$ であり，合成されたプロトコル仕様は，以下のようになる.

$$(E_1[]F_1)|[sync]|(E_2[]F_2)|[G]|C$$

ここで，C はコントローラであり，以下のようになる.

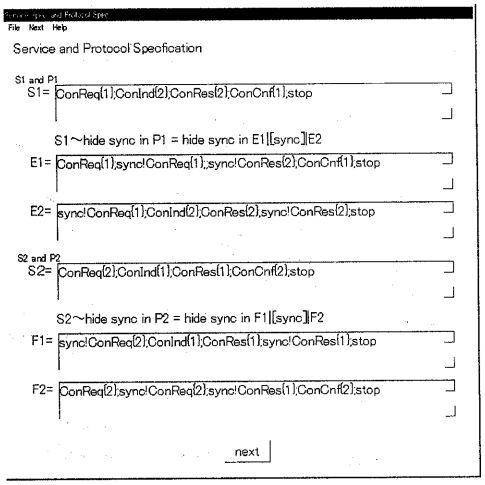


図 7: プロトコル仕様表示

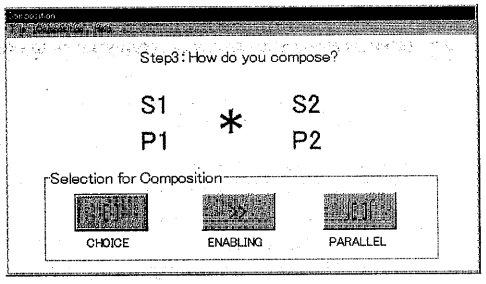


図 8: 合成方法の選択

$$ConReq^1; \mathbf{stop}[]ConReq^2; \mathbf{stop}$$

合成されたサービス仕様とプロトコル仕様は図 9 に示すように，弱双模倣等価となる.

# 4 むすび

本論文では，プロトコル仕様を，与えられたサービス仕様から Langerak[6] の方法を用いて観測等価分解 (弱双模倣等価分解) して導出し，それら仕様群を合成して (サービス仕様群の合成とプロトコル仕様群の合成)，再び等価なサービス仕様とプロトコル仕様を得るソフトウェア環境の設計と実現について述べた. 本合成支援環境の今後の課題は以下の通りである.

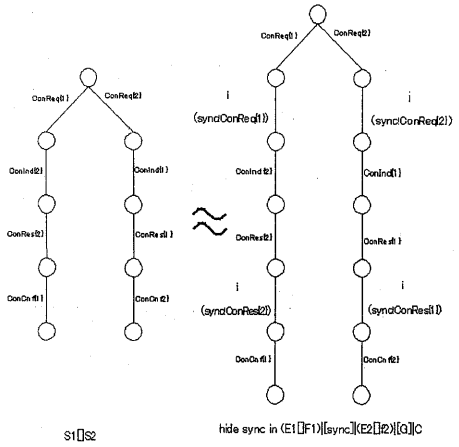


図 9: サービス仕様とプロトコル仕様の遷移システム

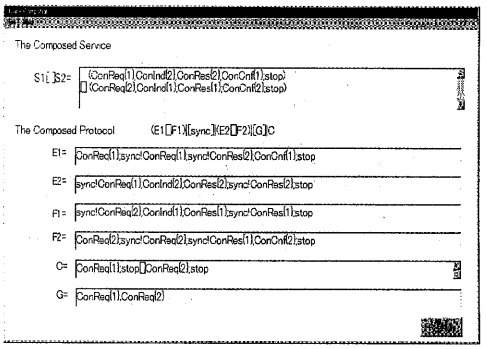


図 10: 合成結果表示

- 割り込み的合成機能の追加
- サービス仕様とプロトコル仕様の図式的視覚化
- 内部イベントを考慮した仕様の合成

## 参考文献


[1] T. Higashino, "Service Specification and Its Protocol Specifications in LOTOS - A Survey for Synthesis and Execution-" IEICE Trans. Fundamentals, Vol.E75-A, pp. 330-338, 1992.

[2] C. Kant, T. Higashino, and G. v. Bochmann, "Derving Protocol Specifications from Service Specifications written in LOTOS," Distributed Computing, Vol. 10, No. 1, pp. 29-47, 1996.

[3] B. B. Bista, K. Takahashi, and N. Shiratori, "Composition of Protocol Functions," IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, Vol.E81-A, pp.586-595, 1998.

[4] H.-A. Lin, "Constructing Protocols with Alternative Functions," IEEE Transactions Computers, Vol. 40, pp. 376-386, 1991.

[5] B. B. Bista, K. Takahashi, and N. Shiratori, "A Compositional Approach for Constructing Communication Services and Protocols," IEICE Trans. Fundamentals," Vol.E82-A, No.11, pp.2546-2557, 1999.

[6] R.Langerak, "Decomposition of Functionality : A Correctness Preserving LOTOS Transformation," in Protocol Specifications, Testing and Verification, pp.203-218, 1990.

[7] ISO, "Information Processing Systems - Open Systems Interconnection- LOTOS-A Formal Description Technique based on the Temporal Ordering of Observational Behaviour," ISO 8807, 1989.