社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－SE－157 (14)
2007／9／28

エージェントネットに基づくSOA検証用UMLシミュレータの提案

倉畑宏行† 藤井拓††
宮本俊幸† 熊谷貞俊†

近年，企業の競争力強化のために，様々な企業内のシステムをサービスとして統合するための技術であるSOA (Service Oriented Archteqture) が注目されている．サービスの動作やサービス間の相互作用が複雑であることから，これらと業務の要求との整合性を手動で確認することは容易ではない．本論文では，SOAに基づくシステムに対するシミュレータを提案する．このシミュレータはシステムの設計のハイレベルな動作やサービス間の相互作用のモデルと業務に対する要求との整合性を確認するためのものである．我々はシミュレータに関する主要な概念として以下の概念を提案する．(a) SOAに基づくシステムの設計のためのUMLモデルとしてROOMモデル (b) サービス間の相互作用のモデルからハイレベルなサービスの動作モデルの生成 (c) 拡張ROOMモデルの動作方法としてサービスの動作の設計からエージェントネットモデルへの変換ルール

A UML Simulator for SOA Based on Agent Net Model

Hiroyuki Kurahata ,† Taku Fujii ,†† Toshiyuki Miyamoto †
and Sadatoshi Kumagai†

Recently, Service Oriented Architecture (SOA) attracts attention as a technique to integrate various systems in the form of services to enhance business capability of companies. Since semantics of services and their interaction are complex, it is not easy to manually validate whether those semantics and interactions satisfy requirements based on business needs. In this article, we propose UML simulator for systems based on SOA. Our simulator makes it possible to validate that high level semantics of system designs and service interaction models conform business needs. We propose main concept of our simulator as followings : (a) Modified ROOM model for designing systems based on SOA, (b) generating high level service behavior models from service interaction model, (c) mapping rule of state machine model into Agent Net model as a execution method of Modified ROOM model .

1. 緒　　論

近年，情報通信技術の発展により企業間の競争が激化している．各企業は競争力を高めるために業務を市場の要求に迅速に対応させることが必要となった．このためには，短期間で業務のための情報システムを構築することが必要である．また，業務のための情報システムの開発をより低コストで実現することが求められる．このような背景から，短期間で柔軟に変更可能なシステムのアーキテクチャとしてSOA(Service Oriented Architecture)[1] が注目を集めている．

SOAとは大規模なシステムをサービスの集合として構築するアーキテクチャのことであり，サービスとは業務に必要な機能の実現の単位である．SOAに基づくシステムは一般的にWebサービスにより実現され，サービスのインターフェイスはWSDL(Web Service Description Language)[2]，サービス間の連携の記述にはBPEL4WS(Business Process Execution Langage for Web Service)[3] が利用される．SOAによりシステムを構築する場合，業務の要求に基づき必要なサービスをつなぎ合わせて開発を行う．このため，システムの設計者はサービスの接続のみを考えるだけでよく，迅速な開発が可能である．また，SOAの特徴のひとつとしてプラットフォーム非依存であることが挙げられる．WSDLやBPEL4WSはすべてXML形式であり，そのことからプラットフォームに依存しない実装が可能である．よって，Linuxベースのシステムとメインフレームのような異なったプラットフォームのシステムを統合可能となる．この特徴により，サービス

† 大阪大学大学院 工学研究科 電気電子情報工学専攻
Dept. of Ele., Elec. and Info. Eng. , Osaka University
†† 株式会社 オージス総研 技術部ソフトウェア工学センター
Software Engineering Center, Osaka Gas Info. Syst. Res. Inst.

として実装していない既存のシステムを新たに開発するシステムと容易に統合可能である．このことから，同様の機能を新たに開発することによる無駄を省くことが可能であり，開発コストを抑えることができる．

SOA に基づくシステムを開発する際の問題としてサービスの複雑さの問題がある．サービスは抽象度が高く，複雑な動作を行う．すなわち，サービスの持つ状態の数や状態から遷移する際の分岐が多い．よって，サービスの組み合わせにより発生する動作のパターンの数が膨大になる．設計に対する検証は現状では手作業により行われることが多い．SOA に基づくシステムの開発の際，サービスの複雑さから手作業による検証では検証漏れが起こる可能性がある．検証漏れにより設計に残った論理的な間違いや必要な機能の欠落は開発の段階が進むほど修正のためのコストの増加や開発の長期化を引き起こす可能性がある．これらの問題から，できるだけ早い段階でシステムに対する設計が顧客の求めるものかどうかを確認する必要がある．

我々は開発対象のシステムを UML を用いて設計し，設計に対する論理的な間違いや必要な機能に対する欠落を発見するためのシミュレータに関する研究を行っている．提案するシミュレータでは UML によりシステムに対する要求に従いサービスの構成やサービスの動作を設計する．設計したモデルを実際に動作させることにより設計の論理的な間違いや設計に必要な機能の欠落を発見可能であると考える．つまり，要求に含まれる矛盾や機能の定義の不足を発見でき，実装までこれらが残ることを防ぐことが可能である．また，UML は実行セマンティクスを持たないことから動作させる方法を提案する．我々の研究では UML による設計をペトリネットなどの視覚的な実行可能モデルに変換する．視覚的なモデルを用いることで UML を簡単な変換規則により動作させることが可能となる．

我々の先行研究[4),5)] では，BPEL4WS の検証のためのシミュレータの開発を行った．このシミュレータは BPEL4WS を UML のアクティビティ図で表記し（BPEL/UML モデル），これを実行可能モデルであるアクティビティハイパーグラフに変換しモデルの実行を可能とした．しかし，BPEL/UML シミュレータには 2 点の問題があった．まず，BPEL/UML シミュレータにおける設計は BPEL4WS をベースとしている．BPEL4WS は特にサービス間の連携の実装に用いられる言語である．よって，サービス間の連携に対する記述が実装レベルの記述である．システムに対する要求から設計を開始する設計の初期段階では，サービスの複雑さからサービスの詳細な動作を把握することは難しい．そのため，設計の初期段階においてサービス間の連携の設計を実装レベルから開始することは困難であり，より簡易に連携を記述する方法が必要である．次に，BPEL4WS がサービス間の連携の実装に用いられる言語であることからサービスの詳細な動作の定義に必要なデータに関する演算のための構文を提供していない．シミュレーションによりシステムの動作を確かめるためにはサービスの持つデータの遷移を確認可能である必要がある．これらの問題は，この BPEL/UML シミュレータのモデリング言語が BPEL4WS に基づくことが原因である．我々はこれらの問題点を解決するため，新たに提案するモデル化手法に基づいたシミュレータを提案する．

本論文では，ROOM 法[6)] と UML のコミュニケーション図を組み合わせたモデル化の方法を提案する．提案する方法はコミュニケーション図を用いることにより，BPEL/UML モデルを用いた方法よりもサービス間の連携の設計をより容易に行う方法である．また，コミュニケーション図によるハイレベルなサービス間の連携のモデルから各サービスの大まかな動作モデルを自動的に生成する．これにより，詳細な設計を補佐すると同時により早い段階からのシミュレーションが可能である．また，提案するシミュレータでは実行可能モデルとしてエージェントネットモデルを用いる．エージェントネットモデルを用いる主な理由は BPEL/UML シミュレータにおける問題点解決のためにデータを扱う必要があることである．エージェントネットモデルはオブジェクト指向ペトリネットの一種であり，オブジェクト指向ペトリネットは拡張されたカラーペトリネットである．よって，カラーが扱えるため，データの演算が可能となる．以後，提案するシミュレータを “SOA シミュレータ”，提案するモデル化の手法を “拡張 ROOM 法” と呼称する．

以降，**2** ではシミュレータの概要を述べる．**3** では SOA に基づく業務システムの UML によるモデル化方法と設計の手順の詳細を述べる．**4** ではコミュニケーション図からステートマシーン図を生成する手法，ならびに提案した UML モデルのエージェントネットモデルの変換方法を述べる．**5** では，今後の課題としてシミュレータの検証のための機能としてシナリオの生成機能について述べる．**6** では，本研究と関連する研究，ならびに本研究の先行研究を挙げ本研究との違いを述べる．最後に，**7** で結論を述べる．

2. SOA シミュレータ概要

SOA に基づくシステムは基本的にサービス間の接続

関係，各サービスの動作によって設計される．ROOM法[6]はこれらの設計を可能とし，さらにUMLへのマッピングルール[7]が提案されているモデル化の手法である．ROOM法はモジュール間の接続関係とモジュールの動作によりシステムを設計する手法であり，UML2.0を用いたROOM法の表現ではSOAに基づくシステムはコンポジット構造図とステートマシーンモデルにより表現される．すなわち，サービス間の接続関係はコンポジット構造図，サービスの動作モデルはステートマシーンモデルにより表現される．また，サービスの動作をステートマシーンモデルにより表現することで先行研究の課題であったデータに関する記述が可能となる．UMLによるROOM法のうちコンポジット構造図はシステムに対する要求から容易に作成可能である．しかし，サービスの動作は複雑で，1つのサービスは入力やサービスの状態により様々な動作を行う．つまり，サービスの動作の設計は要求に含まれる多数のシステムの動作を同時に考慮して設計する必要がある．よって，サービスの動作であるステートマシーンモデルをシステムに対する要求から直ちに作成することは容易ではない．

我々は上記のSOAに基づくシステムのサービスの動作設計の難点を解決するため上記のROOM法を拡張した拡張ROOM法を提案する．拡張ROOM法ではROOM法による設計に加えて，コミュニケーション図によりサービス間のメッセージ交換の順序をケースごとに分けてモデル化する．ケースごとにサービス間の連携を記述することで先行研究の課題であった簡易な連携の記述が可能である．そして，複数のコミュニケーション図から大まかな各サービスの動作を表現したステートマシーンモデルを生成する．このように，ROOM法にサービス間の連携を設計する手順を加え，動作モデルを生成することでSOAに基づくシステムの設計の困難な点であるサービスの動作を容易に設計可能とした．また，拡張ROOM法に基づき作成したモデル（以後，拡張ROOMモデルと呼称）を実行するためのシミュレータとしてSOAシミュレータを提案する．SOAシミュレータはステートマシーンモデルをエージェントネットモデルに変換することによって拡張ROOMモデルを実行する．

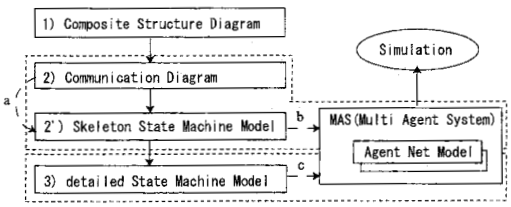


図 1 SOA シミュレータ概要

図1に拡張ROOM法による設計の手順と，エージェントネットモデルへの変換の関係をシミュレータの概要として示す．まず，ROOMモデルに基づきコンポジット構造図を作成する (1)．次に，サービス間の連携をケースごとにコミュニケーション図により設計する (2)．これは，一般にシステムに対する要求は機能ごとに入出力の条件やそれに対する動作のパターンが記述されていることから，システムの動作の各ケースにごとのサービス間の連携を容易に抽出可能であることが理由である．この手順では，システムに起こりうる各々の動作が複数のコミュニケーション図を用いて表現される．次に複数のコミュニケーション図から各サービスの大まかな動作を表現したステートマシーンモデルを生成する (2', a)．これは，サービス間の連携のモデルが各サービスの動作のモデルと関係することを利用している．つまり，サービスの動作はサービス間の連携の設計と矛盾しない設計である必要がある．この大まかな動作モデルであるステートマシーンモデルをスケルトンステートマシーンモデルと呼ぶ．スケルトンステートマシーンモデルの生成によりあらかじめ設計した複数のコミニケーション図をすべて実行可能なステートマシーンモデルが得られる．各サービスの詳細な動作は生成したスケルトンステートマシーンモデルにデータの演算などの記述を追加することによって設計する (3)．この詳細なステートマシーンモデルを詳細ステートマシーンモデルと呼ぶ．以上が拡張ROOM法である．また，SOAシミュレータはサービスの動作モデルであるステートマシーンモデルをエージェントネットモデルへ変換することにより拡張ROOMモデルを実行する (b, c)．エージェントネットを用いる理由は視覚的な実行可能モデルである点，データを扱うことが可能である点，モジュール性のあるモデルである点の3点である．言語により処理する場合と比較して，視覚的なモデルは同様に視覚的なモデルであるUMLのモデルと基本的な要素間の対応関係をとることで比較的変換しやすいという利点を持つ．また，サービスの動作をシミュレーションするためにはデータの定義やデータに関する演算を定義し，実行結果を確認する必要がある．更に，モジュール性のあるモデルであれば他のシステムの設計の際に特定のサービスの実行可能モデルを再利用可能である．SOAシミュレータは特にスケルトンステートマシーンの実行を可能としたことにより，サービスの動作を確認しながら動作の設計を詳細化可能とした．以上の

表 1　SOA モデルの作成手順

a		サービスの構成と接続関係
	1.	システム構成を表現するコンポジット構造図を定義
	2.	各サービスを表現するクラスを定義（以下サービスと呼称）
	3.	サービスの子要素としてサービスの持つインターフェイス，ポートを定義
	4.	サービスの持つポートとインターフェイス間の実現関係を定義
	5.	あるサービスの持つポートと他のサービスの持つインターフェイスの間の依存関係を定義
b		サービスの連携順序の設計
	1.	インターフェイスの子要素としてサービスの持つ操作を定義
	2.	操作の入出力を表現する属性を操作の子要素として定義
	3.	サービス間の連携を表現するコミュニケーション図を定義
	4.	各サービスのオブジェクトをライフラインとして定義
	5.	各サービスの操作の呼び出しをメッセージとして定義
	6.	3. - 5. を連携のケースの数だけ行う
	7.	定義した複数のコミニケーション図から各サービスのスケルトンステートマシーン図を生成する
	8.	コミニケーション図により作成されたステートマシーン図にガード条件を追加し動作の確認を行う
c		サービスの詳細動作の設計
	1.	コンポジット構造図上においてサービスの振る舞いの要素としてステートマシーンモデルを明示
	2.	同図上で状態機械にポートを定義し，サービスのポートと依存関係を定義
	3.	ステートマシン図内で使用する変数をサービスの子要素として定義
	4.	サービスの持つ操作の状態，状態遷移，変数の演算などを記述
	5.	サービスの詳細な動作を確認

ことから，SOA シミュレータによる設計は以下の特徴を持つ．

A)　サービス間の連携を定義

B)　サービス間の連携モデルからサービスの大まかな動作のモデルを生成

C)　エージェントネットモデルによる動作モデルの実行

3. 拡張 ROOM モデルの詳細

拡張 ROOM モデルの作成手順の詳細を表 1 に示す．コンポジット構造図では各サービスはクラスで表現し，サービス間の接続はポートとインターフェイスの依存関係で表現する．また，コミュニケーション図ではサービス間の連携順序はメッセージによって表現し，メッセージはシーケンス番号によって全順序で与えられる．更に，動作モデルであるステートマシーンモデルではサービスの子要素またはステートマシーンモデルの子要素として定義した変数を用いてデータに関する演算が定義可能である．

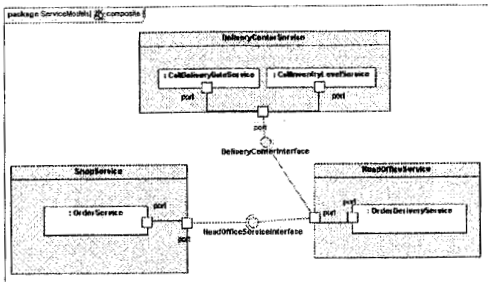


図 2　コンポジット構造図

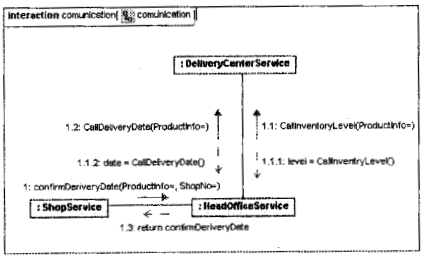


図 3　コミュニケーション図：在庫あり

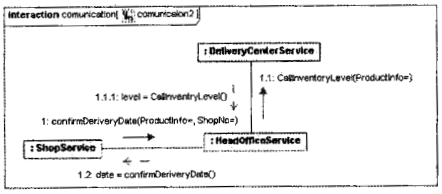


図 4　コミュニケーション図：在庫なし

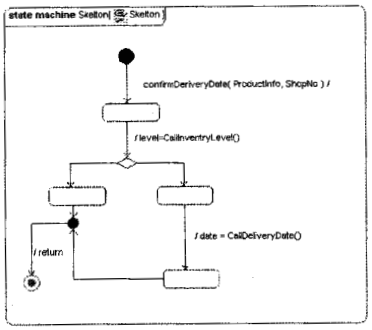


図 5　HeadOfficeService のスケルトンステートマシーンモデル

表 1 に示した手順による実際の SOA に基づく業務システムの設計の具体例として，図 2 から図 6 に物流システムの拡張 ROOM モデルを示す．ここで，設計対象の物流システムは店舗，本社，配送センターの 3

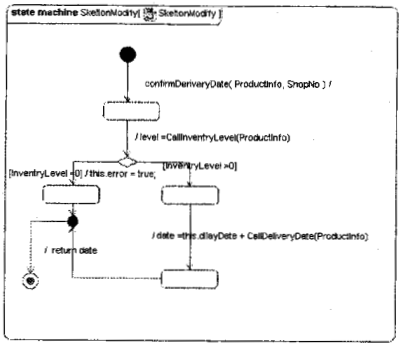


図 6 HeadOfficeService の詳細ステートマシーンモデル

つの支部のシステムからなるものとする．この物流システムにおいて，商品の取り寄せ機能をサービスとして設計することを考える．ただし，商品取り寄せ機能とは店舗からの商品の配送注文を本社が受付け，配送センターへ配達を手配する機能である．

- 図 2 のコンポジット構造図では，設計対象のシステムのサービスとサービスの接続を定義する．図ではサービスとして *ShopService*, *HeadOfficeService*, *DeliveryCenterService* を定義した．また，インターフェイスとポートを依存関係により接続し，サービス間の接続を表記している．
- 図 3, 4 のコミュニケーション図で設計する主な要素は，各インターフェイスに定義された操作の呼び出しメッセージと返信メッセージである．これは抽象度の高いのサービス間の連携順序の 1 つのケースであり、詳細設計に移る前段階として設計する．図 3 は在庫がある場合の，図 4 は在庫が無い場合の動作である．これらの図から各操作に対応するスケルトンステートマシーン図を生成する．
- 図 5 に生成した *HeadOfficeSercice* のスケルトンステートマシーンモデルを示す．このスケルトンステートマシーン図をエージェントネットへ変換することでここまでの大まかなシステム設計の動作確認を行う．次に，このスケルトンステートマシーン図を詳細化し更に詳細なシミュレーションを行う．
- 図 6 に詳細化した *HeadOfficeService* のステートマシーンモデルを示す．ここでは，分岐の際に図 5 では定義されていなかったガードの追加や変数に対する演算を追加している．例えば，以下の記述は *HeadOfficeService* の持つ *DeliveryDate* というデータに *DeliveryCenterService* の持つ操作

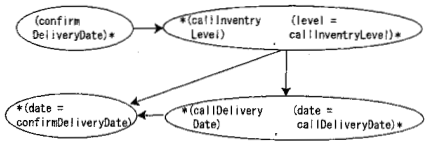


図 7 メッセージグラフ

CallDeliveryDate() の帰り値と *delay* を加えて代入するという意味である．

```
/ date=this.delayDate
    +CallDeliveryDate(ProductInfo);
```

また，ここではスケルトンステートマシーンモデルから構造を変えていないが，状態の階層化などを用いて状態をより詳細化することも可能である．

4. 変換規則の詳細

4.1 スケルトンステートマシーンモデルの生成

前章で述べたように複数のコミニケーション図から各サービスのステートマシーンモデルを生成する．本章では，この変換の規則を述べる．変換は以下に示す手順で行う．

1. 各サービスについて入出力メッセージのシーケンス番号による全順序集合を作成する
2. 各全順序の共通部分をマージしたグラフ (メッセージグラフと呼ぶ) を作成
3. 後述する変換規則に従いメッセージグラフをステートマシーン図へ変換

この手順を前述した図 3, 4 を用いて説明する．ここでは特に，*HeadOfficeService* に注目することとする．まず，1. により以下の 2 つの全順序を得る．ただし，*HeadOfficeService* への入力を (operationName)* ，出力を*(operationName) と表現している．

- (confirmDeliveryDate) * → *(callInventryLevel) → (level = callInventryLevel)* → *(callDeliveryDate) → (date = callDeliveryDate)* → *(return = confirmDeliveryDate)
- (confirmDeliveryDate) * → *(callInventryLevel) → (level = callInventryLevel)* → *(return = confirmDeliveryDate)

次に，2. では全順序に対してメッセージグラフを作成する．この際，同期呼び出しに対する返信メッセージの受信は同一ノードにマージする．上記の全順序から作成されるメッセージグラフを図 7 に示す．図では，callDeliveryDate の呼び出し部分が分岐となる．また，同期呼び出しである callInventryLevel, callDeliveryDate は単一ノードにマージされる．

最後に，3. ではメッセージグラフをスケルトンス

表 2　メッセージグラフからステートマシーンモデルへの変換

Message Graph	State Machine Model
receiveNode (X)*	1) if |(Node)*|<=1 && |*(Node)|<=1 service.X() /
	2) if |(Node)*|==1 && |*(Node)|>1 service.X() /
	2) if |(Node)*|==1 && |*(Node)|>1 service.X() /
syncNode *(X) , x=(X)*	1) if |(Node)*|<=1 && |*(Node)|<=1 / x = service.X()
	2) 3) same as receiveNode
replyNode *(data =X)	1) if |(Node)*|<=1 && |*(Node)|<=1 / return
	2) 3) same as receiveNode
invokeNode *(X)	1) if |(Node)*|<=1 && |*(Node)|<=1 / X()
	2) 3) same as receiveNode
Arc	state

テートマシーンモデルへ変換する．変換規則を表 2 へ示す．表 2 ではノードは遷移に，アークは状態に変換している．また，ノードの入力アークの数を $|*node|$，出力アークの数を $|node*|$ と表現している．ただし，変換ルールには以下の条件を追加する．

- 初期状態，終了状態はスケルトンステートマシーンモデルを作成した段階で定義することとする．
- ステートマシーンモデルに変換した際，入力状態の無い遷移は初期状態から，出力状態の無い遷移は終了状態を出力先に指定する．

この変換規則により図 7 を変換すると図 5 となる．このように生成されたスケルトンステートマシーンモデルの分岐にガード条件を追加する．その後，次節で述べるエージェントネットへの変換し，動作の確認を行う．

4.2　エージェントネットモデルへの変換

エージェントネットモデルは通常のカラーペトリネットの定義に加えて，メソッド，R 関数，という特殊な要素を用いることにより分散性を実現したペトリネットである．メソッドは他のエージェントネットからトークンを受け取るための特殊なトランジションである．また，R 関数はメソッドにトークンを渡すためのプレースに与える関数である．このエージェントネットを図的に表したものを図 8 に示す．エージェントネットを図的に表す場合，メソッドはトークンと同じ外形である長方形の内部を黒く塗りつぶしたもので表現する．図 8 の動作の例を述べる．図では同様の構造を持った *Agent1*, *Agent2* があり，まず *Agent2* の *t2* が発火し，*p3*, *p4* にトークンが入る．次に，*p4* に R 関数：$R(p4) = (m1, p2)$ があるので，このプレースから *m1* という名前を持つメソッドの入力プレースである *p2* へトークンが渡される．よって，*Agent1* の *p2* へトークンが渡り，*Agent2* の *t1* を発火すると，はじめの状態から *Agent1* と *Agent2* を入れ替えたマーキングとなり，以降同様な動作を行う．また，図では int 型のカラー A を定義し，カラー A の変数 X を定義している．ここで，例えば *t1* にアーク関数として $X = X + 1$ のように記述すると，*t1* の発火ごとに変数 X の値が 1 増えるモデルが定義できる．

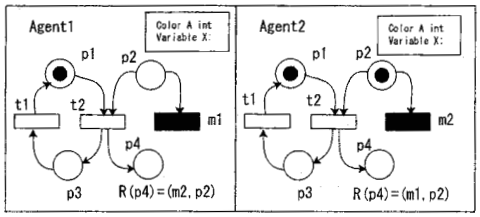


図 8　エージェントネット

以上のように，マルチエージェントネットはカラーとアークエクスプレッションによるデータ定義と演算が可能であり，R 関数とメソッドによるモジュール性を持ったモデルである．よって，SOA に基づくシステムのシミュレーションを行うための実行モデルに適していると考える．

ここで，拡張 ROOM モデルのエージェントネットへの変換規則を示す．拡張 ROOM モデルからエージェントネットモデルへの変換は以下の手順で行う．

1. 各サービスの持つ属性，操作の子要素である入出力属性をカラーに変換
2. 各サービスの持つ各ステートマシーンモデルの要素をエージェントネットモデルの要素へ変換
3. サービスの持つすべてのステートマシーンモデルから変換されたエージェントネットモデルの要素をマージして 1 つのエージェントネットモデルを定義．

1. で行うカラーへの変換ルールを前章で作成した拡張 ROOM モデルの例を用いて図 9 に示す．図は，サービスの子要素として定義した属性がネットの持つカラー，ステートマシーンの持つ属性がローカルカ

ラー，インターフェースの持つ入出力属性がオペレーションカラーとしてインプットカラーとアウトプットカラーにそれぞれ定義されることを示している．作成した，これらのカラーはステートマシーンモデルからエージェントネットの要素への変換の際，プレースのカラー定義やエージェントネットのカラーの定義に用いる． 2. ではステートマシーンモデルの各要素を図3のルールに従って，エージェントネットモデルの要素へ変換する．基本的に，状態はプレースに，遷移はアークとトランジションのセットに変換される．また，外部操作の呼び出しはR関数，保持操作の呼び出しはメソッドにより対応している．また，変数に対する操作は同様のセマンティクスを持つアーク関数に変換される．更に，ローカルな変数や外部操作を呼び出す際の変数の値を保持するためのバッファとして，ローカルカラーとオペレーションカラーの組をすべて含んだカラーを持つプレースを1つ定義する． 3. は 2. で変換したエージェントネットモデルの要素をコンポジット構造図で定義した各サービスの状態機械ごとにまとめる操作である．よって，複数の状態機械を持つサービスは 2. で生成されたエージェントネットを部分ネットとして持つエージェントネットとなる．

変換の例として，図11にHeadOfficeServiceのエージェントネットを示す．これは，図6を変換して作成したエージェントネットモデルである．チョイスによる分岐はp6, 連結点はp 10である．また，p_Valiableは変数のバッファである．

5. 今後の課題

提案した各段階においてより効率的にシミュレーションを行うためにはテストケースの生成や要求との整合の確認を自動化する仕組みが必要であると考える．

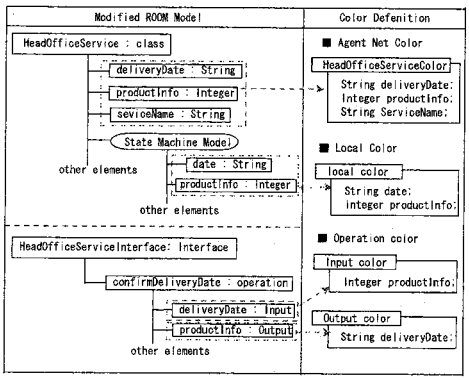
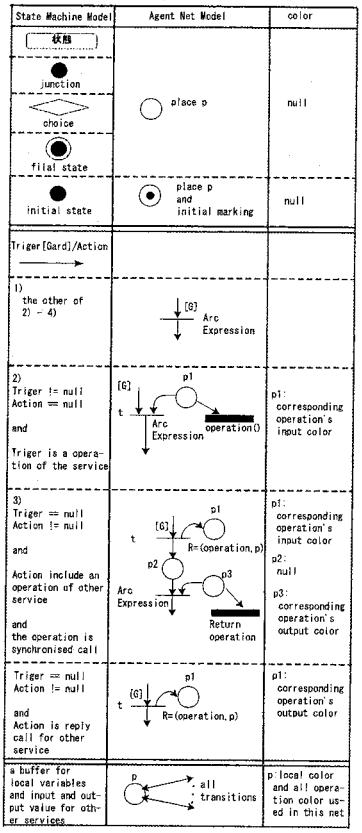


図 9　拡張 ROOM モデルからカラーの生成

表 3　拡張 ROOM モデルからエージェントネットへの変換規則



現状では各サービスの持つ変数の値または入力値は手作業で定義せざるおえない．これらの値を人が考えるには時間がかかる．また，これらをひとつひとつ確認する作業にも時間がかかる．これらの作業は開発の効率化の妨げになる可能性がある．我々はシミュレータの機能としてにシナリオの生成機能を提案する予定である．図11に実現予定のテストシナリオの概念を示す．シナリオは，テストケース，予測状態遷移，予測終了状態からなる．テストケースはサービスの持つデータの初期値である．また，予測状態遷移はあるテストケースにおける実行されるべきシステムの状態遷移である．予測終了状態はあるテストケースにおけるシステムの終了時点の状態やデータの値である．テストケースを実際に実行し比較することで要求との整合性を確認する．このようなシナリオのセットを生成することによりより効率的な開発が可能であると考えている．このシナリオの生成機能を今後解決する予定である．

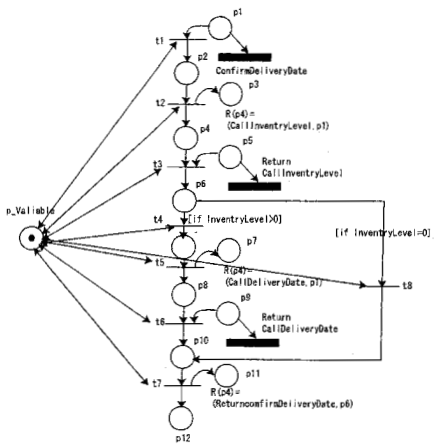


図 10 HeadOfficeService のエージェントネット

6. 関連研究

Yan[10] らは Web サービスシステムを色つきペトリネットを用いてモデル化し，CPNTools を用いて検証を行う手法を提案している．また Schlingloff[11] らは BPEL4WS を用いて記述された Web サービス間の連携をペトリネットモデルへ変換しペトリネットベースのモデル検査器を用いて検証する手法を提案している．BPEL4WS や WSCI は Web サービスの実装に用いられる言語である．よって，これらは実装レベルのサービスの連携手順をペトリネットを用いてモデル化し，形式的に検証を行うことを目的とした研究である．

これらの研究がサービス間の連携順序を単一の色付きペトリネットを用いているのに対して，我々の研究では SOA のモジュール性を表現可能なオブジェクト指向ペトリネットを用いてモデルを動作させている．また，我々の開発するシミュレータは，SOA に基づくシステム開発の設計初期から用いるためのものである．よって，対象として SOA を設計するためのモデルの提案とシミュレーション方法の提案を行っている．

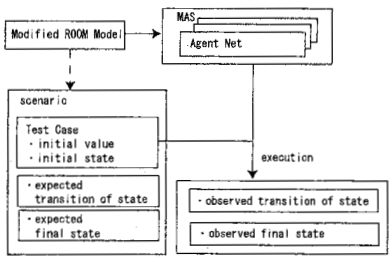


図 11 シナリオの生成機能

設計したモデルを動作させることで要求に含まれる矛盾や機能の欠落などに対する確認も可能である．

7. 結　論

本論文では SOA に基づく業務システム開発のための UML シミュレータの提案を行った．シミュレータの特徴は段階的なシステムのモデル化と段階的なシミュレーションである．段階的なシステムの設計のためのモデルとして拡張 ROOM モデルによるモデル化手法を提案した．また，拡張 ROOM モデルの各段階でシミュレーションを行うための手法として，コミニケーション図からステートマシーンモデルの生成方法，ステートマシーンモデルからエージェントネットモデルの変換方法を提案した．今後の課題として，**5.** で述べたシナリオの生成機能の提案を行う予定である．また，これらを基にシミュレータを実装し，シミュレータの有効性の確認を行う予定である．

参 考 文 献

1) E. Thomas. Service-Oriented Architecture, PRENTICE HALL, (2004).
2) http://www.w3.org/TR/wsdl
3) ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
4) E. Satoru, M. Toshiyuki, K. Sadatoshi, "On Simulation of BPEL4WS/UML Descriptions", Proc. of APSEC, pp. 103-109, (2005).
5) H.Kurahata, T.Fuji, T.Miyamoto, S.Kumagai. "A UML Simulator for Behavioral Validation of Systems Based on SOA", Proc. Next Generation Web Services Practies, pp 3-9, (2006).
6) B.Selic, G.Gullekson, P.T.Ward REAL-TIME OBJECT-ORIENTED MODELING, WILEY PROFESSIONAL COMPUTING, (1994).
7) B.Selic, J.Rumbaugh, Using UML for Modeling Complex Real-Time Systems, Rational Software Corporation (1998)
8) T. Miyamoto, S. Kumagai, "A Multi Agent Net Model and the Realization of Software Environment", Proc. of Workshop of Petri Nets to intelligent system development in ICATPN'99, pp.83-92,(1999).
9) http://www.daimi.au.dk/CPnets/cpngroup.html
10) Y.-P. Yang, Q.-P. Tan, Y. Xiao, "Verifying Web Services Composition Based on Hierarchial Colored Petri Nets". *Proc. of IHIS '05*,(2005).
11) H. Schlingloff, A. Martens, and K. Schmidt, "Modeling and Model Checking Web Service", *Electronic Notes in Theoretical Compute Science*, 126, pp 3-26, (2005).