特集論文 「エージェント」

自律システム実現に向けたアーキテクチャの構築

An Architecture with Planning for Autonomic Systems

西村 一彦 Kazuhiko Nishimura
電気通信大学大学院 情報システム学研究科／株式会社ボイスリサーチ
Graduate School of Information Systems, University of ElectroCommunications / Voice Research, Inc.
knishimura@ohsuga.is.uec.ac.jp

中川 博之 Hiroyuki Nakagawa
電気通信大学大学院 情報システム学研究科
Graduate School of Information Systems, University of ElectroCommunications
nakagawa@is.uec.ac.jp

田原 康之 Yasuyuki Tahara
（同　上）
tahara@is.uec.ac.jp

大須賀 昭彦 Akihiko Ohsuga
（同　上）
akihiko@ohsuga.is.uec.ac.jp

keywords: agent, planning, autonomic computing, platform, goal management

Summary

Autonomic systems aim to reduce the configuration, operational, and maintenance tasks of large distributed applications. In order to implement autonomic systems, several approaches, such as Self-managed system and Autonomic Computing, have been proposed. This paper describes an architectural approach for autonomic systems, which is based on a three-layered model. In the uppermost layer, the planning function, which is an important part of this model, has to efficiently make an effective sequence of operational services to satisfy goals. In this paper, we propose an automated planning algorithm using hierarchy planning technique. Our planner composes the sequence of operational services in the most abstract space, and then it successively embodies the detail of them. The critical values, which determine the abstract space, are automatically discovered from the knowledge of operational services. We also present the experimental results to show the effectiveness of our method.

1. は じ め に

情報システムの大規模化，複雑化が進む中，アクセス数の急激な増大や何らかの原因によって発生する不具合等に対して，システム運用者を介さず，自律的に適応するシステムの需要が高まっている [日本 06, Srivastava 04, Kramer 07, 木下 08, 西村 09].

図 1 は，Web アプリケーションの典型的なシステム構成を表している．「A. 通常運用時」に示すように，複数サーバ (Server-1 から Server-4) 上にデータベースや Web アプリケーションサーバといった異なるソフトウェアサービス群 (A，B などの丸印) が配備されている．そして，オペレーション管理サーバがこれらのサーバ群およびソフトウェアサービス群の動作状況をモニタリングする．なお，ここでは説明のためサーバを 4 台としているが，昨今のデータセンターにおいては，サーバが数百台になることは決して珍しいことではない．

これらのサーバ群において「B. 障害発生時」に示すようにサーバ Server-1 が何らかの原因で CPU の負荷が急激に高くなった場合，その状況をオペレーション管理サーバが検知する．そして，その検知した結果を受けて，どのようにしてサービスを安定した状態に切り替えるかをシステム運用者は考えなければならない．この例では，Server-4 で新たにサービス A を起動し，サービス A の動作確認を行った後，Server-1 上のサービス A を停止し，最終的に Server-1 を停止させることでサービスの継続運用が可能となる．

また，障害対応以外でも，運用中のシステム上で一部のソフトウェアをバージョンアップすることは良くあることであり，こういったことへの対応も自律システムにおいては重要な要件といえる．

しかも，大規模で複雑なシステムでは，こうした運用手順の立案と実行を運用者が手動で行うのではなく，システム自身が状況をすばやく，そして的確に把握し，自律的に対応することが求められる．

Kramer らはこうした自律性を備えるシステムを実現する方法として，3 層アーキテクチャモデルを提案している [Kramer 07]．その基本的な考えは，システムの状況変化に対し，単にその場しのぎの対応をするのではなく，システムの置かれた状況を予測しながら戦略的に対応することにある．Kramer らの 3 層アーキテクチャモデルは，自律型ロボットの制御を参考にして考えられたものであり，コンポーネント制御層，変更管理層，ゴール管理層から構成される (図 2).

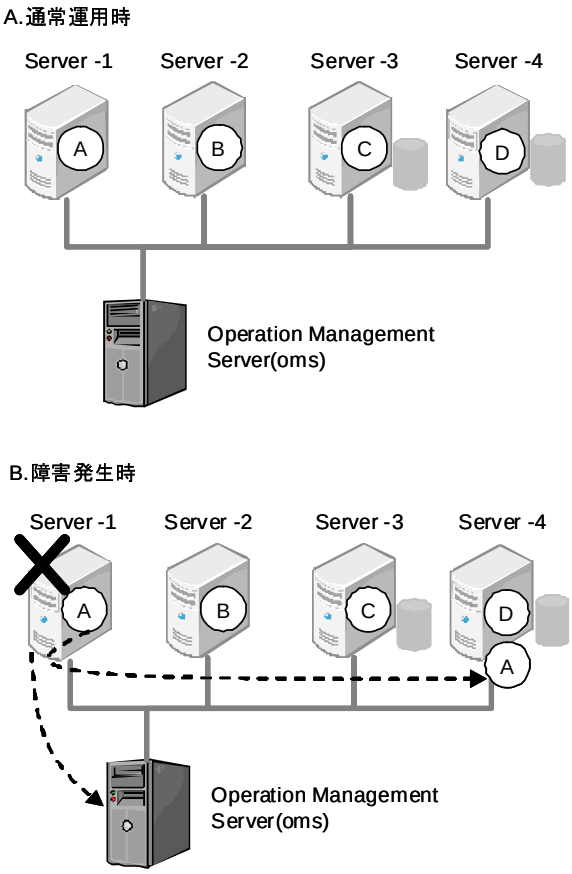

図 1 自律コンピューティングの例

コンポーネント制御層は，制御対象となるシステム固有の手続きを定義し，実行を制御する層である．たとえば，図 1 の Web アプリケーションシステムでは，Web サーバの起動コマンドやパラメータに該当する．また，コンポーネント制御層は，アプリケーションの稼働状況をモニタリングし，その状況変化を上位層に伝える．

変更管理層は，ゴール管理層が生成するプランを解釈し，プランを構成するアクション群を順番に実行する層である．また，コンポーネント制御層から状況変化 (例：コマンドの失敗など) を受理した場合は，他のプランの適用可能性を評価して，再びコンポーネント制御層に通知する．しかし，代替案が見つからない場合には，上位のゴール層に対して，他のプランの生成を要求する．

ゴール管理層は，初期状態から目標状態に達するプランを立案する層である．初期状態や目標状態は情報提供者 (例：運用管理者など) から与えられる．この層では最もハイレベルなプランが得られ，たとえば，サーバプロセスをマイグレーションする際のサービス系列 (例：サーバを起動する，パラメータを設定する，アプリケーションプロセスを停止するなど) が立案される．

Kramer らは，これらの 3 つの構成要素の中でゴール管理層の高速化と不確定な状況への対応が重要な課題であることを説いている [Sykes 08]．ゴール管理層では下位

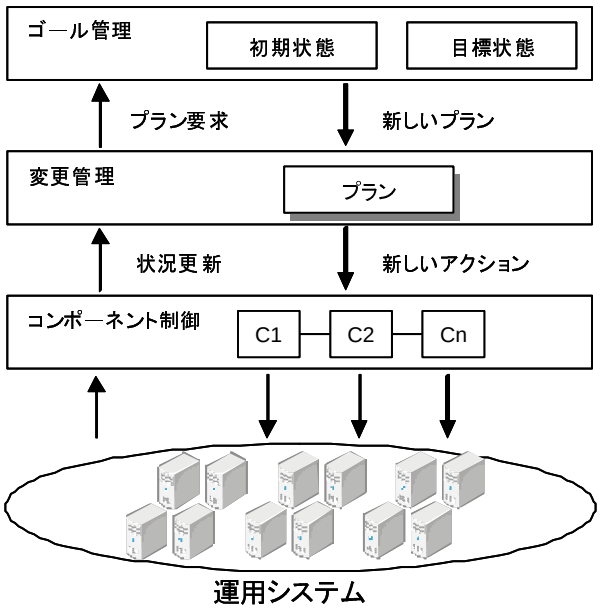

図 2 3 層アーキテクチャモデル

(変更管理層やコンポーネント制御層) に対する戦略や指針 (ポリシー) をプランとして提供する重要な部分である．一般に，プランニングには多くの時間と計算機リソースを必要とする．また，情報提供者は，プランニングに必要な情報を完全には提供できないため，ここでは，不完全な情報に基づいてプランニングを行わなければならない．さらに，プラン実行時に状況が変わってしまい，得られたプランが必ずしも実行できず，再プランニングを行うこともある．

本論文では，図 2 の 3 層アーキテクチャモデルを参照した自律システムの実現を念頭に，特に「ゴール管理層」に焦点をあて，以下の課題を解決する方法を提案する．

- 効率の良いサービス系列 (プラン) の合成
- 情報が不完全な状況下でのプランニング

以下，2 章では，関連研究の紹介を通じて自律システムとプランニング技術の関連および現在の課題を明らかにする．3 章では，その課題に対する解決方法として抽象階層を用いた階層プランニングを紹介し，4 章で，その階層を決める指標 (緊急値) の計算を詳しく説明する．5 章では，本手法を用いた実験とその結果に基づいた考察を述べ，6 章にてまとめとする．

2. 関 連 研 究

自律システムの代表例として「Autonomic Computing(自律コンピューティング)」がある [日本 06]．自律コンピューティングは自己構成，自己修復，自己最適化，自己防御という 4 つの機能から構成される．これらの 4 つの機能により，状況変化への対応が迅速になり，管理の複雑さの低下，信頼性や可用性の向上などが実現されるとして

いる．その適用事例には，オンラインの販売サイトや顧客管理システムなどの Web アプリケーションを中心とした企業内情報システムがある．

自律コンピューティングは，いくつかの層から構成されている．最下位は，管理対象となるリソースを管理する「管理対象リソース」層，その上位に管理対象リソースへのアクセスを一元化する「管理エンドポイント」層，さらにその上に自律コンピューティングの中核とも言える「オートノミック・マネージャ」がある．オートノミック・マネージャは，管理対象リソースに関する様々な情報を収集し，その情報に従って適切なアクションをとり，その結果を評価するという PDCA サイクルに基づいており，次の 4 つのタスクからなる．

監視 管理対象 (計算機システムなど) の情報の収集，集約，フィルタリング，および報告を行う

分析 各種モデルに基づき，得られたデータを分析して将来の状態予測を支援する

計画 現状から目標を達成するために必要なアクションやポリシーを合成する

実行 サービスの移動を実施するなど，動的な更新を考慮しながら計画の実行を制御する

これらのタスクの内，計画と実行を具現化する技術として，近年，AI プランニングの技術が見直され，その適用にあたって活発な議論が展開されている．また，前述の Kramer の 3 層アーキテクチャと対比すると，機能的にはゴール管理と変更管理層がオートノミック・マネージャに相当する．

AI プランニングは，初期状態から目標状態に至るアクションの系列を効率よく求める技術である．ロボットの行動計画から始まり [Fikes 71]，宇宙船の制御や自律ロボット [Nau 04]，電力系統システムの障害復旧の立案 [Bertoli 02, Thiebaux 01]，Web サービスの自動合成 [Carman 03, Oh 06, Peer 05, 小出 07] といったビジネスアプリケーションの応用事例も数多く報告されている．特に近年は，Web サービス合成問題の解決手段として，数多くの研究が行われている [Carman 03, Oh 06].

Srivastava らは，大規模システムにおけるサーバ運用を想定し，前述のオートノミック・マネージャの計画・実行タスクに対して，AI プランニング手法を利用した Java ベースの汎用フレームワークを開発した [Mediratta 06, Srivastava 04]．このフレームワークではいくつかのプランニングアルゴリズムを実装している．適宜これらのアルゴリズムを切り替えることで，自律システムの柔軟性が向上するとしている．本論文で述べる階層プランニング手法は，このプランニングアルゴリズムの一つに位置づけられる．

Peer は，AI プランニング手法を Web サービス合成問題に適用する際の課題を整理している [Peer 05]．Web サービス合成は自律システムと共通な問題を含んでいる．たとえば，不確定な情報に関わる問題である．制御対象となるシステムはコンピュータなどの様々な機器のみならず，利用者 (エンドユーザやシステム運用者など) も当然のことながら含まれれる．制御対象は常に変化しており，プランニングの段階でその変化をすべて予測することは事実上困難である．そこで，実行時の偶発的な出来事を予め想定したり，実行が失敗した際にいつどのように再プランニングをするかといった高度な判断が求められる．本研究では，抽象階層に基づく階層プランニングの手法を用いることによって，ある程度の再プランニングの効率化が可能となる．具体的には 5 章で説明する．

3. プランニングによるゴール管理層の実現

本章では，3 層アーキテクチャモデルの最上位に位置するゴール管理層の実現方法を説明する．ここで中核となる技術は階層プランニングである．

我々はこれまでに汎用的な問題解決システムを効率化するメカニズムの研究に取り組み，抽象階層を利用した階層プランナの効率化 [西村 91] や Web サービス合成問題への適用可能性を探ってきた [西村 08].

そこで，まず初めに階層プランニングの構成要素と基本的なメカニズムを説明する．

3·1 サービス定義

運用対象システムに対する各種サービス (プロセスの起動や停止，プロセスマイグレーション，サーバ構成情報の取得，監視エージェントの設定など) の定義は次のように「Opname」,「Precondition」,「Delete」,「Add」から構成される．

```
Opname::= サービス名
Precondition::= [
        入力データおよび
        サービス適用時に成立すべき条件]
Delete ::= [
        消費されるデータおよび
        サービス適用後の状態で成立しない条件]
Add ::= [
        出力データおよび
        サービス適用後の状態で成立する条件]
```

これは，STRIPS などの AI プランナで採用された定義方法である [Fikes 71].「Opname」 にはサービス名を定義する.「Precondition」リストには，サービスの実行に必要な入力データやサービスを適用するための前提条件を定義する.「Delete」リストはサービスの実行により消費されるデータ (すなわち，出力データとしては得られない) およびサービス適用後の状態で成立しない条件を定義する.「Add」リストはサービスの実行結果として新たに得られる出力データおよびサービス適用後の状態で成立する条件を表す．なお，各データや各条件は,「名称 (引数 1, 引数 2, ... , 引数 n)」の形式で定義される (以降, 属性と呼ぶ)．ここで引数は変数または値である．変数は

```
// 監視エージェントのサーバ間移動
Opname #
    move_agent(agent,OMSX,SRVY,SRVX) ::
 Precondition # [
      type(OMSX,oms), type(SRVX,server),
      type(SRVY,server), connects(OMSX,SRVX,SRVY),
      status(OMSX,active), inserver(agent,SRVY)] ::
 Delete # [inserver(agent,SRVY)] ::
 Add # [inserver(agent,SRVX)].

// ソフトウェアコンポーネントのサーバ間マイグレーション
Opname #
    migrate_sw(agent,SWX,OMSX,SRVX) ::
 Precondition # [
      type(OMSX,oms), type(SRVX,server),
      type(SRVY,server), type(SWX,software),
      installed(SWX), connects(OMSX,SRVX,SRVY),
      status(OMSX,active), inserver(SWX,SRVY),
      inserver(agent,SRVY)] ::
 Delete # [inserver(SWX,SRVY), inserver(agent,SRVY)] ::
 Add # [inserver(SWX,SRVX), inserver(agent,SRVX)].
```

図 3 サービス定義の例

アルファベットの大文字で始まる文字列，値は数字もしくはアルファベットの小文字で始まる文字列とする．

図 3 に 2 つのサービス定義を例示する．「監視エージェントのサーバ間移動サービス (move_agent)」の前提条件 (Precondition リスト) は 6 つの属性から成る．最初の 3 つの属性 (「type(OMSX, oms)」，「type(SRVX,server)」，「type(SRVY,server)」) は，サービスの操作対象物のタイプを表している．このケースでは，OMSX はオペレーション管理サーバ (oms) であり，ここで，OMSX，SRVY，SRVX といった変数はプランニングの過程で実際の値に置き換えられる．SRVX と SRVY はサーバであることを表している．次に「connects(OMSX,SRVX,SRVY)」という属性は，OMSX が異なるサーバ (SRVX と SRVY) 間を仲介していることを表す．続いて「status(OMSX, active)」はオペレーション管理サーバが稼働中であることを表す．最後の属性「inserver(agent, SRVY)」は管理エージェント (agent) がサーバ (SRVY) に存在することを表す．このサービスが適用されると管理エージェントがサーバ間を移動することになるので，現在の状態からは管理エージェントの位置が削除され (Delete リストの「inserver(agent,SRVY)」)，移動先のサーバの情報 (Add リストの「inserver(agent,SRVX)」) が追加される．

もう一つの例は「ソフトウェアコンポーネントのサーバ間マイグレーションのサービス定義 (migrate_sw)」である．対象システムに障害が発生した際に，バックアップ系列に切り替えるために定義されたサービスである．その前提条件 (Precondition リスト) は 9 つの属性から成る．最初の 4 つの属性 (「type(OMSX, oms)」，「type(SRVX,server)」，「type(SRVY,server)」，「type(SWX,software)」) は，サービスの操作対象物のタイプを表している．このケースでは，OMSX はオペレーション管理サーバ (oms) であり，SRVX と SRVY はサーバ，SWX はソフトウェア (software) であることを表している．次の「installed(SWX)」という属性は，ソフトウェア SWX がインストールされた状態であることを表す．「connects(OMSX, SRVX, SRVY)」は，OMSX が異なるサーバ (SRVX と SRVY) 間を仲介していることを表す．続いて「status(OMSX, active)」はオペレーション管理サーバが稼働中であることを表す．続く属性「inserver(SWX, SRVY)」および「inserver(agent, SRVY)」は，ソフトウェア (SWX) と管理エージェント (agent) がいずれもサーバ (SRVY) 上に存在することを表す．以上の条件が満足されると，その適用結果として，Delete リストに示すようにソフトウェア (SWX) と管理エージェント (agent) がサーバ (SRVY) に存在するという属性が現在状態から削除され，Add リストに示すように，新しいサーバ (SRVX) にソフトウェアと agent が存在することを表す属性が次の状態で追加される (「inserver(SWX, SRVX)」と「inserver(agent, SRVX)」)．

表 1 属性の例

属性	説明
inserver(Obj,Svr)	ソフトウェアや監視エージェントなどの操作対象物 (Obj) が指定サーバ (Svr) 上にいる状態を表す
accessto(Obj1,Obj2)	操作対象であるソフトウェアないしは管理エージェント (Obj1) が別のソフトウェア (Obj2) に対して接続可能な状態であることを表す
type(Obj,Type)	操作対象物 (Obj) タイプ (Type) を表す．Type の値は server, software, agent, oms のいずれかである
status(OMS,Sts)	オペレータ管理システム (OMS) の状態 (Sts) を表す．Sts の値は active か inactive のいずれかである．
connects(OMS,S1,S2)	異なるサーバ (S1 と S2) がオペレーション管理サーバ (OMS) を介して接続していることを表す
installed(SW)	ソフトウェア (SW) がインストール済みであることを表す
reset_mode(Svr,Sts)	サーバ (Svr) がリセット可能かどうか (Sts) を表す (一部サーバのみにある)．Sts の値は, active もしくは inactive のいずれかである．
process_stats(SW,Sts)	ソフトウェア (SW) の稼動状態 (Sts) を表す．Sts の値は run か stop のいずれかである

表 1 は，図 1 の問題においてサービス定義の各リストに表れる属性の例である．

3·2 階層プランニング

目標を達成するサービス系列は階層プランニングの手法を用いて獲得する．階層プランニングとは，解くべき問題を複数レベルに抽象化し，抽象度の高いレベルで骨格となるサービス系列を定め，この骨格に従ってサービス系列の細部を抽象度の低いレベルに向かって順次埋めていくという戦略に従う．上位レベルで得られたサービ

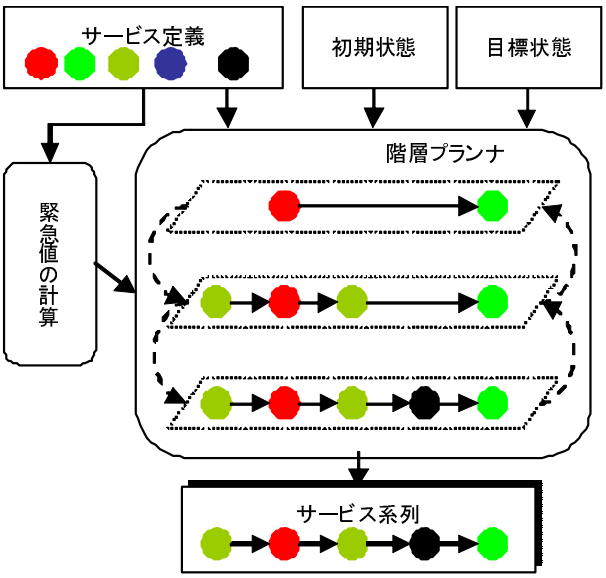

図 4 階層プランナの構成

```
problem(
[
    type(agent,agent), type(oms1,oms),     # 初期状態
    type(server-1,server), type(server-2,server),
    type(server-3,server), type(server-4,server),
    type(serviceA,software), type(serviceB,software),
    type(serviceC,software), type(serviceD,software),
    connects(oms1,server-1,server-2),
    connects(oms1,server-2,server-3),
    connects(oms1,server-3,server-4),
    process_stats(serviceA,stop),
    inserver(agent,server-1), status(oms1,inactive)],
[
    inserver(agent,server-4),                  # 目標状態
    inserver(serviceA,server-4), process_stats(serviceA,run),
])
```

図 5 初期状態と目標状態の記述例

ス系列が，下位レベルのサービス系列に対して自然なスコープ (探索領域を決定する条件) を与えるため，サービス系列を生成するための探索領域が劇的に減少し，効率の良い探索が実現できる．

Sacerdoti は，Precondition リストに含まれる属性に対して緊急値という概念を導入し，問題領域を詳細度の異なるいくつかの階層に分割する方法を提案した [Sacerdoti 74]．プランナは，最も高い緊急値が割り当てられた属性のみを考慮した抽象階層において骨格となるサービス系列を推論する．次いで，緊急値を一つずつ下げ，その時の緊急値を下回る属性を無視した抽象階層において，上位層で得られたサービス系列の細部をうめていく．最終的には緊急値が 1 以上の属性，すなわちすべての属性を考慮し，サービス系列を求める．緊急値の計算については後述する．

図 4 は，緊急値を用いた階層プランナの構成要素を表している．階層プランナに対する入力は,「サービス定義」,「初期状態」,「目標状態」であり，その出力は初期状態から目標状態へ至るサービス系列である．ここで，階層プランナは，後述の緊急値が決定する最上位の抽象階層 (その緊急値を下回る属性を無視して構成される探索空間) からプランニングを開始する．そして，上位階層で得られた骨格となるサービス系列を下位階層で順次補完する形でプランニングを進め，最終的に所望のサービス系列を出力する．なお，途中の階層で探索に失敗した場合には，一つ上の階層に戻り，別なサービス系列を探索する．

各抽象階層でのサービス系列の推論は，STRIPS[Fikes 71] で用いられた以下に述べる手段目的分析の方法に従う．

(1) 目標状態と現在状態を比較する．プランニングの開始時では，初期状態が現在状態となる．目標状態として与えられるすべての属性が現在状態で成立しているならば，プランニングの処理を終了する

(2) 各サービスの Add リストと目標を比較し，Add リスト中の少なくとも一つ以上の属性が目標に含まれるようなサービスを求める．求めたサービスが複数存在する場合は，それぞれのサービスの Precondition リストと現在状態をマッチさせる．その結果，Precondition リスト中の属性群の中でマッチしない属性の数が最も小さいサービスを一つ選択する．

(3) (2) で選択したサービスの Precondition リストが現在状態で成立しているならば，そのサービスの Delete および Add リストを用いて現在状態を新しい状態に更新して，(1) に戻る．しかし，選択されたサービスの Precondition リストが現在状態で成立していない場合は，Precondition リストの中で成立していない属性を新たなサブゴールとして目標状態に追加し，(1) に戻る

3·3 初期状態と目標状態の表現

サービス系列を推論するために与えられる初期状態と目標状態は，対象システムの構成に関する情報や構成要素の状態に関する情報からなり，運用管理者が与えるものと，運用監視システムから自動的に得られるものがある．具体的には表 1 に示す属性の集合から構成される．

たとえば，図 5 は，図 1 に示した 4 台構成のシステムで障害発生時 (B) のサーバが停止した状況 (サービス A が停止) を初期状態とし，サービス A を別サーバ (この場合 Server-4) で再起動した状態を目標状態にするという記述を表している．

4. 緊急値の計算

前章で説明した階層プランニングにおいて，抽象階層を決定する緊急値の求め方について，Sacerdoti は経験則を示している．しかし，この方法は必ずしも探索効率の良い指標ではないことが指摘されている [Andersen 88]．また，実際には計画立案者がこの方針に従って事前に緊

急値を決定しなければならない．

そこで，本論文では，緊急値をサービスの関する情報から自動的に決定し，かつ Sacerdoti の方法よりも探索効率を向上させることができる方法を提案する．前述のサービス定義を利用して，次の手順に従い，決定する．

(1)サービスの分類木の作成
(2)サービスの分類木から暫定緊急値を計算
(3)属性の出現頻度により，最終決定

4·1 分類木の作成

サービス定義から分類木を作成する．具体的には，各サービス定義の Add リストに含まれる属性を比較して，同一なものがあるかどうかを調べる．もし Add リストに同一属性があれば，各サービス定義の Precondition リストの中で同一属性があるかどうかを調べる．そして，Add リストおよび Precondition リストで同一属性から成る「抽象サービス」を求める．ここで属性が同一とは，属性名が同じでかつその引数の数が同じ場合をいう．ここで，引数は属性変数であっても，属性値のどちらであっても良い．抽象サービスが複数得られた場合は，抽象サービス同士を比べ，これ以上抽象サービスが得られなくなるまでこの比較を繰り返す．

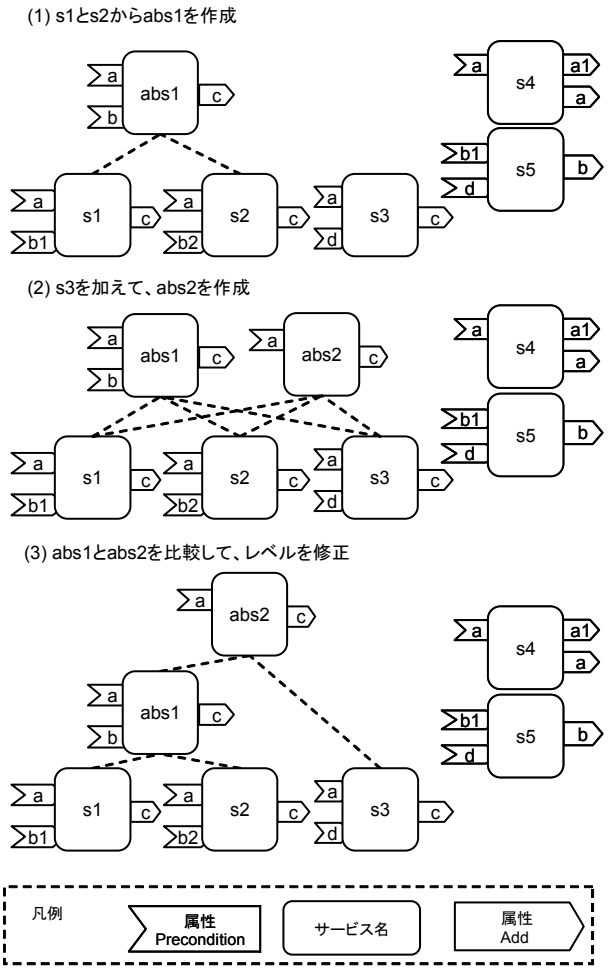

図 6 サービスの分類木

図 6 は，5 つのサービス定義 (s1，s2，s3，s4，s5) に対するサービス分類木の作成過程を表している．なお，ここでは説明のため，サービス定義は簡略化しており，実際には図 3 に示すサービス定義が分類の対象となる．また，図 6 では，各々のサービスの Precondition リストと Add リストに出現する属性名のみを記している．Delete リストはこの図の中では省略している．ここで，アルファベットのみの属性とアルファベットと数字を組み合わせた属性の 2 種類が表れているが，この違いは引数に相当する属性変数 (または値) の違いを示しているものであり，分類木を作成する上では同一属性として解釈する．

Add リストに同一属性をもつサービスは，s1，s2，s3 である．s1 と s2 それぞれの Precondition リストを比較する．s1 の Precondition リストは属性 a と b1 から構成され，s2 の Precondition リストは属性 a と b2 から構成されている．ここでは，b1 と b2 は同一と解釈する．以上から， Add リスト に属性 c を有し，Precondition リスト に属性 a と b (b は b1 と b2 を抽象化した属性とする) を有する抽象サービス abs1 が構成される．

同様にしてサービス s3 を s1 および s2 と比較すると抽象サービス abs2 が構成される．s1，s2，s3 からはこれ以上抽象サービスは構成できないため，次に抽象サービス同士を比較する．すると abs1 と abs2 からは Add リストに属性 c からなり，Precondition リストは属性 a からなる抽象サービスが構成できる．しかしながら，これは abs2 に他ならず，そこで abs2 を abs1 の上位として分類木を修正する．なお，元々の比較対象であるプリミティブなサービスは，常に最下位にあるものとする．

最終的には図 6 の (3) に示す分類木となる．

4·2 緊急値の割当

サービス定義の Precondition リストに出現するすべての属性に対して，サービス分類木をもとに，暫定的な緊急値を割り当てる．Algorithm 1 の assign_critic で示すように，最上位の抽象サービスの Precondition リストに出現する属性には，最も高い緊急値 (C_v:初期値は分類木の高さである HL) を割り当てる．次に，分類木の階層レベルを一つ下げて，その階層レベルに存在する抽象サービス群の Precondition リストに出現する属性を調べる．そして，上位階層に出現した属性は除いて，この階層レベルで初めて出現する属性に $C_v - 1$ の緊急値を割り当てる．以降，階層レベルを一つずつ下げて，最も下位の層になるまで繰り返す．最終的には，すべての Precondition リスト中の属性に対して緊急値を割り当てる．

前述の例では，5 つのサービスの Precondition リストには 4 つの属性 a，b1，b2，d がある．まず，最上位の抽象サービス abs2 の Precondition リストには属性 a があるのでこの分類木の高さである 3 を緊急値として割り当てる．次に，分類木のレベルを一つ下げる．すると，抽象

Algorithm 1 assign_critics(SH, PL, HL)

Input: SH:サービス分類木, PL:Precondition リスト中のすべての属性, HL:SH の高さ;
Output: HP:緊急値が付与された属性リスト;

```
L ← 1;
C_v ← HL;
HP ← [];
while L ≤ HL do
  PR : SH の上から L 番目にある各サービスの Precondition リストに出現するすべての属性;
  if L < HL then
    foreach P_a ∈ PR do
      if P_a ∈ PL then
        P_a に C_v を割当て, HP に追加;
        PL ← PL − P_a;
      end if
    end foreach
  else
    foreach P_b ∈ PL do
      P_b に 1 を割当て, HP に追加;
    end foreach
  end if
  L ← L + 1;
  C_v ← C_v − 1;
end while
```

注：SH のトップのサービス集合を 1 番目のレベルとする

サービス abs1 がある．抽象サービス abs1 の Precondition リストは，属性 a と b だが，b はもともと b1 と b2 を抽象化したものなので，b1 と b2 に緊急値 2 をセットする．最後に残りの d に 1 を割り当てる．

次に暫定的な緊急値を補正する．まず，すべてのサービス定義の Precondition リストの中に出現する属性の出現頻度を計算する．頻度が多いものほど高い緊急値とする．出現頻度が多い属性は，現在状態でその属性が満足されているかどうかを確認する手間が多く，より上位で確定すべきと考えた．この補正作業を暫定結果に対して行い，最終的な緊急値を確定する．

この例では，属性 b1 はサービス s1 と s5 の Precondition リストに出現し，属性 b2 はサービス s2 のみに出現する．そこで，属性 b1 は b2 よりも高い緊急値 (ただし，a よりも低い値) を割り当てる．

最終的には，属性 a に 4，属性 b1 に 3，属性 b2 に 2，属性 d に 1 が割り当てられる．

5. 評価と考察

前節で述べた階層プランニングの方法が従来手法に比べて効率が良いことを示すために実データを用いた性能評価を行う．ここでは，図 1 で例示した Web アプリケーションシステムにおいて，システム障害が発生した場合に，どのようにして復旧をするかというシナリオで評価を行う．具体的には以下のようなシナリオである．

(シナリオ 1) 障害が起きたサーバから代替サーバへ，ソフトウェアサービス (例：Web アプリケーションサーバ) をマイグレーションする

(シナリオ 2) ソフトウェアサービスを複数のサーバに予め冗長配備しておき，障害発生時には，必要分のソフトウェアサービスを初期化・再起動する

まず，(シナリオ 1) に関しては，10 台のサーバにおいて，5 種類のソフトウェアサービスをそれぞれ別々のサーバ上にマイグレーションするというケースで測定を行う．(シナリオ 2) に関しては，サーバ台数を 10，20，30，50 台と変化させたケースで測定を行う．測定内容は，結果を得るまでにかかった時間および探索空間のサイズとする．なお，本システムの実験環境は，ハードウェアは Intel Core Duo プロセッサー T2300/1.66GHz 搭載の Toshiba Dynabook, オペレーティングシステムは Windows XP, 実装言語は SWI-Prolog 5.6 である．

表 2 は，本手法による緊急値の割当と Sacerdoti の方法 (Abstrips) による緊急値の割当を示す．数字が大きいほどより上位の抽象階層に位置づけられる．

今回の実験では，最短のサービス系列（最適解）でなく，実行可能なサービス系列を迅速に得ることを目的とした．そこで，各ケースに対応する初期状態と目標状態を階層プランナに与え，階層プランナが最初にサービス系列を生成するまでの過程を測定した．1 つのケースに対し 3 回の測定を行い，平均値を算出した．なお，今回の実験では，いずれのケースも階層プランナが最初に生成したものは最適解であった．

表 3 は，2 つのシナリオにおけるサービス系列の合成時間 (CPU time) を表している．(シナリオ 2)(50 台) につ

表 2 緊急値の割り当て

属性名	本手法	Abstrips
inserver	7	3
accessto	6	1
type	5	4
status	4	2
connects	3	4
installed	2	4
reset_mode	1	1
process_stats	1	1

表 3 サービス系列を得る時間 (本手法と Abstrips)

ケース	本手法 (msec)	Abstrips(msec)
(シナリオ 1)	2,360	22,343
(シナリオ 2)(10 台)	672	4,109
(シナリオ 2)(20 台)	5,313	53,344
(シナリオ 2)(30 台)	21,703	259,922
(シナリオ 2)(40 台)	141,687	測定不能

表 4 (シナリオ 1) における階層別の探索空間のサイズ (S) とサービス系列の長さ (L)(本手法)

階層	S	L
1	21,409	59
2	3,698	123
3	0	123
4	1	124
5	0	124
6	0	124
7	0	124

表 5 (シナリオ 1) における階層別の探索空間のサイズ (S) とサービス系列の長さ (L)(Abstrips)

階層	S	L
1	15	5
2	998	59
3	1	60
4	7,712	124

いては，Abstrips が規定時間内 (10 分) に終了しなかったため，測定不能とした．いずれも本手法が Abstrips に比べて短時間でサービス系列を得ることができた．これは，本手法による緊急値が，Abstrips オリジナルの緊急値に比べ有効であることを示している．

表 4，表 5 は，(シナリオ 1) における探索空間のサイズ (S) とサービス系列の長さ (L) を階層別に測定した結果である．同様に，表 6，表 7 は (シナリオ 2) の結果である．なお，いずれのシナリオも，各ケースにおいて本手法と Abstrips で得られたサービス系列の長さおよび内容は同一であり，さらには，階層プランナが最初に生成したサービス系列は最適なものであった．しかしながら，これは階層プランナが常に最適解を最初に生成することを保証するものではない．最適なサービス系列を常に最初に得られるかどうかについては今後の研究課題である．

表 4 から表 7 に示す探索空間の大きさを比べると，下位層での探索空間のサイズに違いがある．本手法は第 1 層から第 3 層まで (表 4 においては第 2 層まで) の探索空間が大きく，それ以下は小さい．これは，本手法が，上位層で得られた抽象サービス系列をガイドとしてうまく下位層に引き継いでいることを示している．一方，Abstrips は，最下位層である第 4 層の探索空間のサイズが第 3 層より上の層に比べてかなり大きい．Abstrips は上位層の結果が下位層に対して有効に働かず，結果的に全体のプランニング時間に影響している．

表 6 (シナリオ 2) における階層別の探索空間のサイズ (S) とサービス系列の長さ (L)(本手法)

ケース	10 台		20 台		30 台	
階層	S	L	S	L	S	L
1	225	28	950	58	2,175	88
2	341	56	1,281	116	2,821	176
3	270	56	1,140	116	2,610	176
4	0	57	0	117	360	177
5	0	57	0	117	265	177
6	50	57	100	117	250	177
7	0	57	0	117	375	177

表 7 (シナリオ 2) における階層別の探索空間のサイズ (S) とサービス系列の長さ (L)(Abstrips)

ケース	10 台		20 台		30 台	
階層	S	L	S	L	S	L
1	18	10	63	20	2,825	30
2	165	28	540	58	1,441	88
3	150	29	736	59	1,535	89
4	560	57	2,829	117	4,742	177

次に，各層で得られたサービス系列を分析する．表 7 より，Abstrips では，最下位層に至るまでは最終的なサービス系列を得ることができなかった．これは，サービス系列を得るにはすべての属性を考慮しなければならないことを示している．

しかし，評価実験限定ではあるが，本手法では，第 4 層ですでに最終的なサービス系列が合成されている (表 6)．すなわち，緊急値 3 以下の属性を無視しても，所望のサービス系列を得られたということになる．

Srivastava らは，自律システムが対応しなければならない不完全な状況を，(1) その世界の状態定義 (初期状態や目標状態を含む) が不完全なとき，(2) アクション (本論文で言うところのサービス) の前提条件が不完全なとき，(3) アクションの結果が不完全なときといった分類をしている．今回の実験では，緊急値 3 以下の属性が初期状態として与えられない場合でもサービス系列を得ることができた．実験のケースに限定されるが，見方を変えると，本手法が一部の不完全な状況 (すなわち (1) に該当) にも対応できる可能性を示している．実際のシステム運用では，必要な情報が得られないことは多々ある．たとえば，停止したソフトウェアの構成情報が資産管理システムから消失していたり，システムの監視情報が何らかの原因で記録されていなかったということもある．こうした状況下でも自律システムは何らかの対応が求められ，本手法がこうした課題を解決する一つの可能性となりうることを今回の実験により示すことができた．

システムの復旧手順を立案する際，最も重要な情報は何か，すぐに必要ではないがあれば良い情報は何かを判断することは重要な作業である．本論文で述べた緊急値は復旧手順の立案において重要な情報とそうでない情報を分ける一つの判断基準となりうる．また，立案した手順に基づいてサービスを実行する時に状況が変わり，再プランニングが必要となった場合，もし緊急値の低い属性に変化があったことがわかれば，再プランニングの効率化にもつながる可能性がある．すなわち，変化した属性に付与された緊急値＋１の抽象階層で得たサービス系列はそのまま再利用できる可能性が高く，最初から改めてプランニングを行わずに済むかもしれない．この点に関しては，今後実証を進めたい．

6. ま と め

障害発生等のシステムの変化に対して適切な対処方法をシステム自身が決定および実行する自律システムの実現へ向けて，Kramer らの 3 層アーキテクチャモデルを参照し，特にシステムがどのように対処するかをプランニングする部分に焦点をあてて，階層プランニング手法を応用した運用サービス自動合成の方法を述べ，実験によりその有効性を確認した．

本手法では，階層化に必要な緊急値を運用に関するサービス定義から自動的に発見し，複雑なサービス合成を効率化できる．また，実験から不完全な情報の下でのサービス合成の可能性をあわせて見出すことができた．

今回は，サービス系列を迅速に得ることに主眼においたが，実際に得られるサービス系列の質を高めることにも今後注力していきたい．たとえば，現在は，サービス系列を合成するために，インスタンスレベルでの初期状態および目標状態を提供する必要があるが，本格的なシステム運用を想定した場合，例外処理や性能，さらには信頼性といった様々な条件を考慮していく必要がある．なお，今回はゴール管理層のみであったが，変更管理やコンポーネント制御といった他の機能との連携も今後視野に入れて研究を進める予定である．

◇ 参 考 文 献 ◇

[Andersen 88] Andersen, J.: Plan Abstraction Based on Operator Generalization, *Proc. 7th National Conference on Artificial Intelligence(AAAI-88)*, pp. 100–104(1988)

[Bertoli 02] Bertoli, P., Cimatti, R., Slaney, J., and Thiebaux, S.: Solving Power Supply Restoration Problems with Planning via Symbolic Model Checking, *Proc. 15th European Conference on Artificial Intelligence(ECAI-2002)*, pp. 576–580 (2002)

[Carman 03] Carman, M., Serafini, L., and Traverso, P.: Web Service Composition as Planning, *ICAPS '03 Workshop on Planning for Web Services* (2003)

[Fikes 71] Fikes, R. and Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving in Problem Solving, *Artificial Intelligence*, Vol. 2, pp. 189–208 (1971)

[木下 08] 木下哲男：発展型エージェントシステムの動作状況認識機能, 合同エージェントワークショップ＆シンポジウム 2008, pp. 1–6 (2008)

[小出 07] 小出 誠二, 武田英明：階層型計画による Web サービス分解実行, 人工知能学会全国大会 (第 21 回) 論文集, 1D1-1 (2007)

[Kramer 07] Kramer, J. and Magee, J.: Self-Managed Systems: an Architectural Challenge, *2007 Future of Software Engineering*, pp. 259–268 (2007)

[Mediratta 06] Mediratta, A. and Srivastava, B.: Applying Planning in Composition of Web Services with a User-Driven Contingent Planner, *IBM Research Report* (2006)

[Nau 04] Nau, D., Ghallab, M., and Traverso, P.: *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc. (2004)

[日本 06] 日本 IBM：オートノミック・コンピューティングのアーキテクチャーに関するブループリント (2006)

[西村 91] 西村 一彦, 松本 一教：階層型問題解決システムにおける抽象階層の決定方法, 人工知能学会誌, Vol. 6, No. 5, pp. 701–709 (1991)

[西村 08] 西村 一彦, 中川 博之, 中山 健, 田原 康之, 大須賀 昭彦：階層プランニングによる Web サービスの自動合成, ソフトウェアエンジニアリングシンポジウム 2008(SES2008), pp.139–146 (2008)

[西村 09] 西村 一彦, 中川 博之, 中山 健, 田原 康之, 大須賀 昭彦：自律システム実現に向けたアーキテクチャの構築, 合同エージェントワークショップ&シンポジウム 2009（JAWS2009）, pp. 524–531 (2009)

[Oh 06] Oh, SC, Lee, D and Kumara, S. R. T.: A Comparative Illustration of AI Planning-based Web Services Composition, *ACM SIGecom Exch.*, Vol. 5, No. 5, pp. 1–10 (2006)

[Peer 05] Peer, J.: Web Service Composition as AI Planning - a Survey *, Technical report Univ. of St. Gallen, Switzerland (2005)

[Sacerdoti 74] Sacerdoti, E.: Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence*, Vol. 5, pp.115–135 (1974)

[Srivastava 04] Srivastava, B., Bigus, J. P., and Schlosnagle, D. A.: Bringing Planning to Autonomic Applications with ABLE, *Proc. International Conference on Autonomic Computing (ICAC'04)*, pp. 154–161 (2004)

[Srivastava 05] Srivastava, B. and Kambhampati, S.: Challenges in Adapting Automated Planning for Autonomic Computing, in *Proc. International Conference on Automated Planning & Scheduling (ICAPS 2005)* ,pp. 41–44(2005)

[Sykes 08] Sykes, D., Heaven, W., Magee, J., and Kramer, J.: From Goals to Components: A Combined Approach to Self-Management, *Proc. 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, pp. 1–8 (2008)

[Thiebaux 01] Thiebaux, S. and Cordier, M.-O.: Supply Restoration in Power Distribution Systems - a Benchmark for Planning under Uncertainty, *Proc. 6th European Conference on Planning*, pp. 525–532 (2001)

〔担当委員：松原 繁夫〕

2009 年 12 月 18 日 受理

著 者 紹 介

西村 一彦（正会員）

1988 年東京理科大学理工学研究科経営工学専攻修士課程修了．同年 (株) 東芝入社．2003 年 (株) ボイスリサーチ設立．現在，同社取締役，および電気通信大学大学院情報システム学研究科博士後期課程在学中．2009 年合同エージェントワークショップ&シンポジウム 2009（JAWS2009）において優秀論文賞を受賞．情報処理学会，日本ソフトウェア科学会各会員．

中川 博之（正会員）

1974 年生．1997 年大阪大学基礎工学部情報工学科卒業．同年鹿島建設（株）に入社．2007 年東京大学大学院情報理工学系研究科修士課程修了，2008 年同大学院博士課程中退．同年より電気通信大学助教，現在に至る．エージェントおよび自己適応システム開発手法の研究に従事．情報処理学会，電子情報通信学会，IEEE CS 各会員．

田原 康之（正会員）

1966 年生. 1991 年東京大学大学院理学系研究科数学専攻修士課程修了．同年 (株) 東芝入社．1993～1996 年情報処理振興事業協会に出向．1996～1997 年英国 City 大学客員研究員．1997～1998 年英国 Imperial College 客員研究員．2003 年国立情報学研究所入所．2008 年より電気通信大学准教授．博士 (情報科学)(早稲田大学)．エージェント技術，およびソフトウェア工学などの研究に従事．情報処理学会，日本ソフトウェア科学会会員．

大須賀 昭彦（正会員）

1981 年上智大学理工学部数学科卒．同年 (株) 東芝入社．同社 研究開発センター，ソフトウェア技術センターなどに所属．1985～1989 年 (財) 新世代コンピュータ技術開発機構 (ICOT) 出向．2007 年より，電気通信大学 大学院情報システム学研究科 教授．工学博士 (早稲田大学)．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，人工知能学会，IEEE CS 各会員．