階層プランニング技術を適用した
適応型ソフトウェアの構築に関する研究

西村 一彦

電気通信大学大学院情報システム学研究科
博士（工学）の学位申請論文

2011年9月

階層プランニング技術を適用した
適応型ソフトウェアの構築に関する研究

博士論文審査委員会

主査	大須賀 昭彦	教授
委員	岡本 敏雄	教授
委員	渡辺 俊典	教授
委員	末廣 尚士	教授
委員	田原 康之	准教授

西村 一彦

2011年 9月

A Study on Applying Hierarchical Planning for Adaptive Software Systems

Kazuhiko Nishimura

Abstract

The purpose of this study is to establish a method for the software system to quickly adapt itself to changes throughout a software life-cycle, such as requirements change, system crashes or modification of operational conditions. This thesis proposes an approach to create an adaptive system based on AI planning. AI planning is aimed at finding an efficient way of problem solving. This thesis presents an automated approach to generating abstractions for the hierarchical planning. To evaluate the effectiveness of the approach, it is applied to the following three areas: (1) Web services composition, (2) Operational environment for computing systems, and (3) Software specification acquisition task, all of which require high adaptability.

Web services composition requires an efficient method to find the right web services to meet required goals. In this thesis, an automated composing method based on the hierarchical planning is proposed. Actions are formulated from existing web services definitions.

To cope with the operational management for complex computing systems, several approaches, such as self-managed system and autonomic computing, have been proposed. This thesis describes an approach for autonomic systems, which is based on a three-layered model Jeff Kramer proposed. In this study, a variety of individual operational services are defined, and the proper sequence of operational services is automatically obtained. The study also identifies required information with priorities. Therefore, it contributes to reduce the gathering activities of useless information.

In the software specification acquisition, where the tasks depend on the experience and skills of workers, appropriate guidelines and tools to support these tasks are needed. According to requirements change or modification, the process planner generates a plan for doing the specification activity effectively. This plan can be used to invoke associated tools and information. This thesis presents the experimental results of the application of the prototype system to develop business application software systems.

階層プランニング技術を適用した
適応型ソフトウェアの構築に関する研究

西村 一彦

概要

本研究の目的は, ソフトウェアシステムに対する仕様変更や, システム運用時の障害, 運用条件の変更など開発から運用にわたるライフサイクル全般における様々な状況変化に対して, ソフトウェアシステムがその変化に迅速に適応するための手法を確立することである. その実現に向けて, アクションを定式化し, 目標を達成するアクション系列を効率良く求めることを目的とした AI プランニング技術を応用する.

本研究では,AI プランニング技術の一つである抽象階層に基づく階層プランニング技術を拡張する. 従来のプランニング技術における課題を整理し, その解決方法として高速化に寄与する抽象階層を自動的に決定する方法を提案する. 本論文では, ロボットの行動計画問題を例に実証実験を行い, 従来の技術に比べて高速なプランニングを実現できることを示す.

次に,Web サービスの合成, 大規模システムの運用管理, そしてソフトウェアの仕様獲得の 3 つの領域において, 拡張した階層プランニング技術を適用する. これら 3 つの領域はいずれも, 変化に対して高い適応力が求められている. それぞれの分野における課題を整理し, 課題解決に向けた支援方法を提案し, 実験を通じて評価を行う.

Web サービスの合成に関する研究においては, 目標を達成する複数の Web サービス群を効率良く発見し, サービス系列を合成する手段が求められている. そこで, 与えられた目標に対して, 多数の利用可能なサービスから適切なサービスを選択し, 迅速かつ効率よくサービス系列を合成する方法を提案する. また, 既存の Web サービスの定義からプランニングに必要な情報を抽出する方法を示す.

大規模システムの運用管理に関する研究においては, システム構成の変更や障害に対して, システム自身が適切な対処方法を決定し, 実行する自律システムの実現を目指し,Kramer が提唱する 3 層アーキテクチャモデルに基づき, システム運用サービスの自動合成を支援する. システム運用に関わる各種サービスを個別に定義し, システムの状況や達成目標に対して, 適切なサービス系列を迅速に得ることができる. また, 階層プランニングによって, サービス系列を得る上で必要な情報を優先度付きで明らかにすることができるので. 無駄な情報収集活動を抑えることが可能となる.

ソフトウェアの仕様獲得に関する研究においては, 作業者のスキルや経験に依存する仕様化作業において, 状況に合わせて適切なガイドやツールを提供するための支援技術の確立が求められている. 本研究では, 作業者が行う仕様化作業を定式化するソフトウェアプロセスモデルの技術と, 定式化されたプロセスモデルから状況に合わせた作業プロセスを合成する階層プランニング技術を融合した仕様化作業を支援する環境の実現を

目指す. 作業者の状況に応じた作業手順を立案し, 作業者に対して必要なノウハウを提供し, ツールの操作などを制御することができる. 試作ツールを用いて, 異なるスキルを有する開発者に試用実験を行い, その有効性を検証する.

目次

図 目 次

表 目 次

第 1 章

序論

1.1 本研究の背景

ソフトウェアシステムは社会活動を支える重要な基盤となり，ネットワークやインターネットの進化に合わせて，スケーラブルで可用性に優れたさまざまな新しいサービスが我々の日常生活の中に入り込んできている．一方，ソフトウェアシステムの開発や運用に対して，新たな要求も発生している．たとえば，システムのアクセス手段は，PC が中心だった時代から，今では携帯電話や携帯端末が主流となりつつある．また，システムはさまざまなコンポーネントを組み合わせて構築されており，複雑さが一層増している．さらに，多くのソフトウェアシステムに対して，拡張，改良などの新たな要求が発生したり，外部システムの変更が発生したり，セキュリティや性能といった非機能要件の変更が発生しており，一度構築されたシステムであっても運用中に改良，拡張が行われることは決してめずらしいことではない．

たとえば，以下に示すような要件がソフトウェアシステムに対して発生する．

- 拡張性：新しい機能をソフトウェアシステムに追加する．コンポーネントの追加，置き換え，不要なコンポーネントの削除が含まれる．
- 再構築：ソフトウェアシステムのコンポーネントとその間の関係を再定義する．あるコンポーネントが別なサブシステムに移動する．
- 移植性：さまざまなハードウェア，オペレーティングシステム，プログラミング言語などに適合させる．
- 保守性：バグの修正．エラーが見つかった際にソフトウェアシステムの補修を局所化でき，他の部分への影響を最小限にすることが求められる ．

こうしたさまざまな変更に対して，ソフトウェアシステムを維持管理するためのコストの増大も大きな問題である．なぜなら，こうした変更を受け入れるにはその状況に応じてシステムの構成を柔軟に変更する高度な技術が開発者や管理者に求められ，労働集約的な作業になっているからである．システムの開発や運用管理に携わるものに

とって，システムに関わる Total Cost of Ownership を減らし，Quality of Services を改善することで、いかにして ROI（投資対効果）を上げるかが重要な経営課題にもなっている．

ところで，ソフトウェアシステムを不安定にさせる原因は人に依るところが大きいと言われている．たとえば，2008 年に報じられた日経 BP 社の調査によると，システムダウンの発生原因のうち，49 %は運用操作の誤り，パラメータの設定漏れ，データの入力誤りなどの人為的ミスによるものと報告されている．最近のソフトウェアシステムは多数のコンポーネントを利用して構築されており，その多様な相互関係に起因する複雑さがシステムダウンを起こしやすくしている．運用も以前とは比較にならないくらい難しいものとなり、小さな人為的なミスが致命的な問題を発生させる要因にもなっている．この結果，開発・管理上のエラーの原因究明と解決に対してさらにコストがかかるという悪循環を生み出している．システムの運用管理は，すでに，人間が管理できる限界を超えているという指摘もあり，自動化も含めた支援が必要に迫られている．

また，システム障害の発生要因はシステム運用だけに限定されるものではない．経済産業省の「高度情報化社会における情報システム・ソフトウェアの信頼性及びセキュリティに関する研究会の中間報告書」（図 1.1）では，システム障害の原因を分析すると全体の 70%が運用・保守で発生しているが，30%近くは，システムの開発時にその要因が組み込まれていると報告されている [75, 76].

今後，社会基盤における IT 依存度が高まる中、システムの開発から運用・保守フェーズまでをカバーする支援技術の重要性が増すことになる。

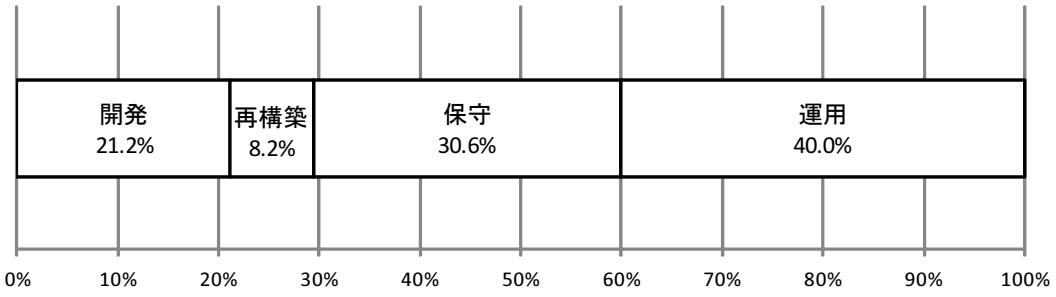


図 1.1 システム障害の発生原因

しかしながら，一方で，ソフトウェアの開発や運用は意思決定を伴う作業であることから，完全に人間を排除し，自動化することは難しい．そこで，少しでも人間の負担を軽減しようという目的で，さまざまな技術が提供されている．

たとえば，ソフトウェア工学の一つの分野として，ソフトウェアプロセスという研究が 1990 年頃から行われている．ソフトウェアの大規模化に伴い，ソフトウェア開発プロジェクトの管理という新たな課題が発生し，その解決策として,「ソフトウェアプロセス」の重要性が強調された [14]．L.Osterweil による「ソフトウェアプロセスもソ

フトウェアである」という提唱をきっかけにして，ソフトウェアを開発するプロセスを手続き的なプログラムとして記述し,「実行」しようという研究が始まった．それ以来，ソフトウェアの開発プロセスを記述する形式言語やモデル，さらにはこの形式化されたプロセスを再利用するための支援環境が数多く提案されている．たとえば、開発者のスキルに合わせて適切なアドバイスを行う支援システムや開発者の作業履歴を元に新たな開発方法論を作成する試みなどがある．

システムの運用管理の観点からは，自己適応型システム (Self-Adaptive System) やオートノミックコンピューティングといった研究が活発に行われている．

図 1.2 は，自己適応型システムを概念的に表したものである．自己適応型システムでは，ターゲットシステムに何らかの変化が起きた場合，その変化を検知し（変化の観測），利用者から与えられる目標との差分を明らかにし，その差分を埋めるアクション系列を求めて（アクション列を決定），ターゲットシステムに対して実行する（アクション列の実行）．利用者が目標を与えることにより，システム自身が能動的に変化に対応することになる．

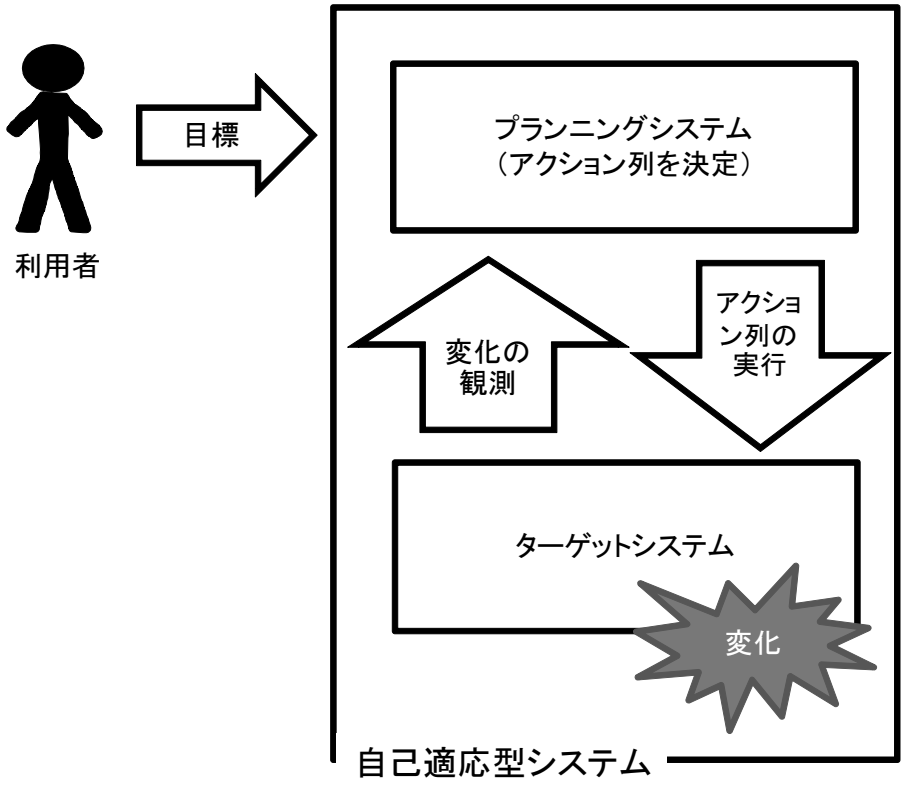


図 1.2 自己適応型システム

オートノミックコンピューティングは，2001 年の National Academy of Engineers 会議の基調講演で IBM の Paul Horn が「Autonomic Computing Manifesto」というプレゼンテーションを行い、一躍脚光を浴びた考え方である．システム運用を 監視，発見，計画，実行の 4 つのサブプロセスからなる Closed Loop システムとしてとらえ，これらのサブプロセスを順次行うことによってターゲットシステムを自律的に管理する．オー

トノミックコンピューティングでは，この制御を Monitor-Analysis-Plan-Execute という要素から構成される MAPE-K ループと呼んでいる．この考えに基づくことで，システムの構成変更や，障害発生時の対応を可能な限り自動化する試みがなされている．

1.2 本研究の課題

自己適応型システムは，主にシステム運用作業の自動化を目的としたものだが，この考え方は，システムのライフサイクル全体に適用できる考え方である．また，ソフトウェアシステムの開発環境そのものもソフトウェアの一つであることから，開発環境を自己適応型システムの一つとみなすことができる．

当然のことながら，システムの開発時と運用時では作業の内容も異なれば,「変化」の質や量も異なる．しかし，変化を検知し，それに対して何かしらのアクションを取るという点ではシステム開発時も運用時も共通に行われることである．

図 1.2 に示したように，自己適応型システムには３つの主要な機能があるが，本研究では，これらの３つの機能のうち，特に，アクション列を決定する部分の実現に焦点をあて，その課題解決に向けた方法を提案する．

このアクションに関する研究として，AI プランニングがある．AI プランニングはロボットの行動計画から始まった研究であり，アクションをどのように定式化し，目標を達成するためのアクション系列を効率良く求めることを目的とした研究である．

AI プランニング技術を利用して，自己適応型システムのプランニングシステムを実現するにあたり，具体的な課題は以下の通りである．

- アクションの表現：利用者（システム開発者やシステム運用管理者）が実際にどのようなアクションを行っているかを表現しなければならない．通常，システム開発方法論や運用管理マニュアルなどで利用者が行うべきアクションは規定されているが，アクションを適用するための前提条件や適用結果は十分に整理されているとは言い難い．明文化されていないことや，利用者の経験的知識に依存する部分もあり，こうした隠れた情報を明らかにする必要がある．

- アクション列の推論：与えられる目標に対して，アクション列を効率良くかつ迅速に得るにはどのようにすれば良いかはプランニングにおいて本質的な課題といえる．単独のアクションが解となることは稀で，通常は複数のアクションを組み合わせることが必要である．すなわち，求められる組合せをいかにしてすばやく求めるかが，推論の重要な役割となる．変化を検知し，アクション列を決定し，実行するまでの時間は短いほど良い．しかしながら，高速化を追求するあまりに，メカニズムが複雑になることは避けなければならない．シンプルで分かりやすい方法が望まれる．

- 利用者への意義のある情報提供：ターゲットシステムから得られる変化の内容や利用者から与えられる目標が完全な形で得られない場合，利用者と協調しながら

問題を解決することが望まれる．その際，利用者に対してどのような情報を提供すべきが重要である．たとえば，アクション列を決定する上で不足している情報が何か，適用可能なアクション列が複数あった場合にどれを有効とするかといった情報を利用者へ提供することは，利用者の意思決定に強く影響する．

1.3 本研究の目的

本研究の目的は，1.2 節で述べた課題の解決に向けて，AI プランニングの一種である階層プランニング技術を適用し，その有効性を示すことにある．その具体的な手段として，抽象階層に基づく階層プランニング技術を利用する．

さらに，提案する階層プランニング手法が適応力のあるソフトウェアシステムの実現において，実際にどのように適用できるかを，3 つのソフトウェアシステムを例にして検証する．ここで取り上げる例は，Web サービスの合成，システムの運用管理，そしてソフトウェアシステムの仕様獲得である．いずれも，変化に対して高い適応力が求められている分野である．それぞれの分野における課題を整理し，その課題解決に向けた支援方法を提案し，実験を通じて評価を行う．

1.4 本論文の構成

図 1.3 に本論文の構成を示す．以下，本図に沿って本論文の章構成と概要を述べる．

第 2 章では，本研究の基本技術となるプランニング技術の動向について触れ，抽象階層プランニング技術について述べる．ここではプランニングの高速化に向けた課題を整理し，その解決方法として，アクションを定義するオペレータからプランニングを効率化するための抽象階層を自動的に得る方法を述べる．本方法は，オペレータをその適用結果の類似性によって分類木としてまとめる．そして，オペレータの前提条件の達成の難易度を分析し，その結果を反映することによって抽象階層を得る．ロボットの行動計画問題を例にして実証実験を行い，本手法の有効性を評価する．

第 3 章から第 5 章では，第 2 章で提案する階層プランニング技術を異なる 3 つの分野に適用し，その有効性を評価する．

第 3 章では，Web サービス合成問題に対して AI プランニング技術を適用した結果を述べる．与えられた目標に対して，多数の利用可能なサービスから適切なサービスを選択し，迅速かつ効率よくサービスを合成する方法を提案する．既存の Web サービス定義（WSDL 文書）に対して，若干の情報を追加することによって，比較的容易にプランニング技術を適用することができる．

第 4 章では，状況の変化を検知し，自律的に振舞うソフトウェアシステムの実現に向けたアーキテクチャの構築について論じる．障害発生など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 らが提案する 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

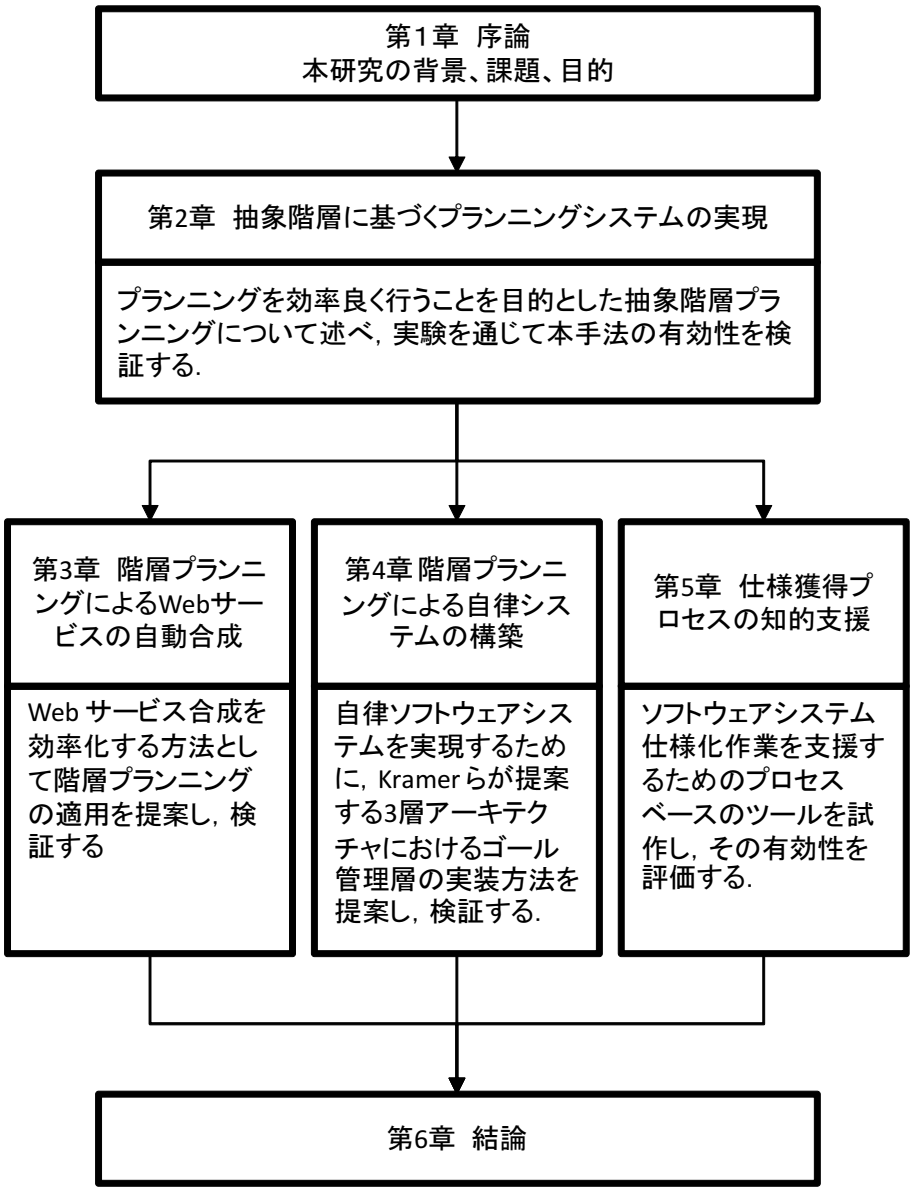

第1章 序論
本研究の背景、課題、目的
第2章 抽象階層に基づくプランニングシステムの実現
プランニングを効率良く行うことを目的とした抽象階層プランニングについて述べ，実験を通じて本手法の有効性を検証する.
第3章 階層プランニングによるWebサービスの自動合成
Web サービス合成を効率化する方法として階層プランニングの適用を提案し，検証する
第4章 階層プランニングによる自律システムの構築
自律ソフトウェアシステムを実現するために，Kramerらが提案する3層アーキテクチャにおけるゴール管理層の実装方法を提案し，検証する.
第5章 仕様獲得プロセスの知的支援
ソフトウェアシステム仕様化作業を支援するためのプロセスベースのツールを試作し，その有効性を評価する.
第6章 結論

図 1.3 本論文の構成

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る．システム運用に関わる各種サービスを個別に定義しておき，システムの状況や達成目標に対して，迅速に適切なサービス系列を得ることができる．

第 5 章では，ソフトウェアシステムの仕様化作業を支援する方法を述べる．仕様化作業を詳細に定義するソフトウェアプロセスモデルの技術と階層プランニング技術を利用し，仕様化作業を支援する開発環境について述べる．システム仕様化作業に関するドメイン知識をビューポイントという概念で整理し，状況に合わせて適切なビューポイントとその切り替え手順を立案するしくみとしてプランニング技術を応用する．

第 6 章では，本研究における成果と，今後の課題を述べる．

主要関連文献

第2章
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.

第3章
Shinichi Honiden, Kazuhiko Nishimura, Naoshi Uchihira, Kiyoshi Itoh，An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems，IEEE Trans. on Software Engineering, Vol.20, No.11, pp.849-867, 1994.

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, pp.139-146, 2008.

第4章
西村一彦, 中川博之, 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，人工知能学会誌, Vol. 26, No. 1, pp. 107-115, 2011.

第5章
西村一彦, 本位田真一，複合ビューポイントによる仕様化プロセスの分析，情報処理学会論文誌, Vol. 34, No. 5, pp. 1074-1086, 1993.

Kazuhiko Nishimura, Yasuyuki Tahara, and Akihiko Ohsuga，A Knowledge-based System for Software Specification，Proceedings of 4th Joint Conference on Knowledge-Based Software Engineering, pp.235-236, 2010.

第 2 章

抽象階層に基づくプランニングシステムの実現

本章では，プランニングの効率化を目的とした抽象階層プランニングの実現について述べる．特に．オペレータに関する知識からプランニングを効率化するための抽象階層を自動的に得る方法を述べる．本方法は，オペレータをその適用結果の類似性によって分類木としてまとめる．そして，オペレータの前提条件の達成の難易度を分析し，その結果を反映することによって抽象階層を得る．

2.1 研究の背景

人工知能の分野において長年にわたり研究が進められている AI プランニング技術は，プランの作成，修正，管理，実行，監視のすべての側面を完全にまたは部分的に自動化することを目的とした研究分野である．その主要なテーマは，対象となる世界におけるアクションをどのように表現し，与えられた問題に対してプランを効率よく探索することであり，これまでにさまざまな技術を産み出してきている．

また，AI プランニング技術の応用分野も各方面にわたっており，ロボットの行動計画 [19, 42, 74]，航空管制管理，生産工程の分析 [49]，ゲーム [67]，電力系統制御 [9, 73] などが代表的な事例である．さらに，ソフトウェア工学分野への適用が活発に進められている．これは，ソフトウェアの大規模化，複雑化により，人手によらない自動化支援が求められていることに起因する．たとえば，2010 年に開催された計算機システムに関する国際会議 BIONETICS 2010 では．”Emerging Synergies of Artificial Intelligence and Software Engineering ”という特別トラックが設けられ，ソフトウェアの開発及び運用の生産性向上に対して，AI プランニングなどの人工知能技術がどのように貢献するかが議論されている [38]．これまでにも，何度となく国内外の主要な会議で，これらの 2 つの分野の関係が取り上げられており，最近では，Web サービス合成問題 [10, 11, 39, 40, 55, 61, 65]，ソフトウェアのテストの自動化支援 [28, 41]，エージェントシステムに代表される自律型システムの実現手段 [4, 8, 68, 69]，開発プロセスの支援

[44, 62] などが話題になっている．

Nau らはここ数年の AI プランニングの技術動向を総括し，　AI プランニング技術が数多くの実用システムで成功を収めてきていることに触れるとともに，AI プランニング技術の課題と方向性を示している [47]．たとえば，マルチエージェント環境でのプランニングというテーマを取り上げており．従来，プランニングシステムは単体プログラムとして存在していたが，今後は，システムの一部として組み込まれ，人も含めてさまざまなエージェントとインタラクションする可能性があり，それに合わせて適切なプランを提示することの重要性が増すと指摘している．この他には，時間に関する推論，動的に変化する外部環境，ドメイン知識の獲得などを研究テーマとして取り上げている．

以上のように，AI プランニングの技術の裾野が広がる一方，プランニングに伴う探索の効率化という重要なテーマが依然として存在することも，Nau らは指摘している．STRIPS に始まりこれまでに数多くの探索効率化への取り組みが行われてきている．これらは，プランニングに用いる知識の性質に依存する方法と，知識の使い方に依存する方法の二つに分類することができる．

前者はプランニングを効率化するために質の良い知識を獲得する方法に関する研究であり，機械学習や事例ベース推論といった技術がある．また，ドメイン固有の情報や知識を付加するものである．これは，プランニングシステムのメカニズムを改良するのではなく，プランニングシステムの知識の質を向上するためのメカニズムを付加することによって，そのプランニングシステムの性能をあげようとする方法である．たとえば，MACROPS は STRIPS が成功プランを見つけるとそのプランを一つのマクロオペレータに変更し，一般化を行い，新しいオペレータ知識として再利用する．

これに対して，後者は，探索の観点からより良いメカニズムを探る研究である．いわば，プランニングの方法論という形で扱われてきた．たとえば，探索の大局的な目的を導入し，どのように探索するかを分析するという手段目的分析は代表的な考え方である．ここでは，問題の表現方法に関する研究も重要なテーマであり，たとえば，プラン間の相互関係を改善するためにプランをネットワーク構造で表現する方法がある．また，問題を階層的に捉えることにより，探索を効率化する方法も数多く提案されている．

前者に含まれる機械学習の技術の活用は，プランニングの知識が更新されるだけでそれを用いるプランニングシステムの能力に合わせたもので，それを用いるプランニングシステムは何も変化しない．つまり，学習される知識はプランニングシステムの能力に合わせたもので，いくら良い知識が得られたとしても使う側の能力を越えてしまうような知識は全く意味がない．また，多くの場合，学習システムへの入力はプランニングシステムが解いた事例が用いられる．一方，プランニングシステムの性能そのものが改善されれば，多くの事例が高速に処理可能となり，さらにプランニングシステムの性能をあげることができる．

ドメイン固有の知識を活用することは有用であることはこれまでに数多くの研究お

よび実際のシステムで評価されている．しかしながら，一方でシステムが複雑となり，拡張性や応用分野の広がりという側面では，シンプルで扱いやすいプランニングシステムのメカニズムが求められていることもまた事実である．

2.2 本章の構成

こうした背景から，本章では，探索のメカニズムに焦点を当て，具体的な方法として問題空間の抽象階層を利用した階層プランニングシステムについて述べる．本章で論じる研究のポイントは次の通りである．

研究課題

ロボットの行動計画などに用いられるプランニングの探索効率の向上を図る．

課題を解決するための手段

オペレータに関する知識からプランニングを効率化するための抽象階層を自動生成し．それをもとに階層プランニングを行うシステムを実現する．

評価結果

従来の階層プランニングシステムに比べて探索効率の良いプランニングシステムを実現することができる．特に．上位層のプランをガイドとして下位層のプラン生成が効率化することができる．

以下，2.3 節では，抽象化による階層プランニングシステムの一つである ABSTRIPS とその問題について述べ，2.4 節では上記二つの抽象階層の決定方法について詳細を述べる．さらに，2.5 節では本方法の有効性を検証するためにロボットの行動計画問題での実験について述べる．2.6 節で実験結果に対する考察を行い，本手法の課題を述べ，2.7 節でまとめとする．

2.3 抽象階層に基づく階層プランニングシステム

プランニングシステムの高速化に対し，抽象化の概念が重要なファクタであることは以前から指摘されている．抽象化とは，問題の表現の変換，すなわち，基本表現から新しい表現（抽象表現）への写像である．抽象化概念をプランニングシステムに適用する最大の効果は問題を解決するために形成される探索空間の縮小にある．より具体的に言えば，問題を抽象化することによって探索空間の縮小を効果的に行なうことができる．

抽象化による階層プランニングシステムは，ABSTRIPS で最初に実現されたものである．ABSTRIPS で最初に実現されたものである．ABSTRIPS は STRIPS の後継として開発されたもので，同時に達成されるゴールの順序付けによってプランニングの効率化を行う．すなわち，

(1) 問題の主要部と細部という概念を導入し，細部に関わる条件を無視した縮退した探索空間で問題を解決する．

(2) その結果をガイドラインとして，条件を付加した拡張した探索空間でのプランニングを行う．

といった2つのステップを繰り返す．この方法によって，STRIPSの問題であった非常に大きくなる探索量を減らすことができることが示された．

ABSTRIPSでは，この階層プランニングを行うためにオペレータの前提条件の達成の緊急値を与えることが必要となる．この緊急値は抽象化のための知識である．Sacerdotiはこの緊急値を求めるための経験則を提案している．しかしながら，あくまでも経験則であり，この経験則から導かれた緊急値がプランニングシステムの効率化に寄与しないケースがあることも実験的に確認されている．

本章では，オペレータに関する知識から自動的かつプランニングシステムを効率化するための抽象階層を得る方法を述べる．本方法は，オペレータをその適用結果の類似性によって分類木としてまとめる．そして，オペレータの前提条件の達成の難易度を分析し，その結果を反映することによって抽象階層を得る．

2.3.1 オペレータの表現

本節では，汎用的なプランニングシステムを効率化するメカニズムとして，抽象階層を利用した階層プランニングシステムのメカニズムを述べる [54].

汎用的なプランニングシステムとして開発されたSTRIPSは，ロボットの行動計画を行うために開発されたものである．STRIPSでは，プランニングの対象となる世界を述語表現とルールにより記述し，世界の変化をオペレータにより記述する．STRIPSは問題の初期状態，目標状態，オペレータが与えられると初期状態から順次オペレータを適用させて目標状態を満足するオペレータの列，すなわちプランを生成する．

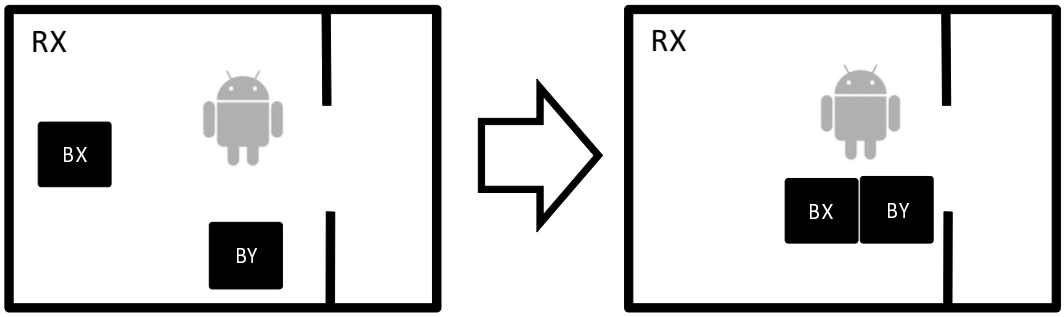
```
Opname::= オペレータ名
Precondition::= [
        オペレータ適用時の状態で成立していなければならない条件]
Delete ::= [
        オペレータ適用後の状態で成立しない条件]
Add ::= [
        オペレータ適用後の状態で新たに成立する条件]
```

図 2.1 オペレータ定義

図2.1に示すようにオペレータは「Opname」,「Precondition」, 「Delete」,「Add」から構成される．

「Opname」 にはオペレータ名を定義する.「Precondition」リストには，オペレータを適用する際に成立していなければならない条件群を定義する.「Delete」リストはオペレータ適用後に成立しない条件群を定義する.「Add」リストはオペレータ適用後の条件群を表す．なお，各条件は,「名称 (引数 1, 引数 2, ... , 引数 n)」の形式で定義される．ここで引数は変数または値である．変数は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 2.2 にオペレータの定義を例示する．このオペレータは，ロボットが離れた位置にある異なる 2 つの箱を隣合わせにするという動作を表したものである.




```
OpName ::= pushb(BX, BY)
Precondition::=[
                type(BY,box), pushable(BX), nextto(robot, BX),
                inroom(BX, RX), inroom(BY, RX), inroom(robot, RX)]
Delete ::=[
                at(robot, A,B), nextto(robot, A),
                at(BX, A, B), nextto(BX, A), nextto(A, BX)]
Add ::=[
                nextto(BX, BY), nextto(BY, BX) ]
```

図 2.2 オペレータの定義例

2.3.2 階層プランニング

人間がいくつかの部分目標からなる問題を解く際には次の手順に従う.

(1) 未解決な部分目標の中から，最も重要なものを選ぶ.

(2) 選択された部分目標を解く.

(3) すべての部分目標が解決されるまで，上記を繰り返す.

STRIPS の基本的な戦略は手段目的分析という局所的な判断のもとでプランニングを行うため，しばしば探索が非効率になる．すなわち，STRIPS ではすべての部分目標は対等に扱われ，いわゆる「重要な部分目標から先に解決する」という戦略は存在しない．このため，探索時間の増大という問題を引き起こした．

そこで，部分目標を主要な部分とその細部という概念で分類し，階層的に問題を解決するというアプローチが提案された．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オペレータ系列を定め，この骨格に従ってオペレータ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オペレータ系列が，下位レベルのオペレータ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，オペレータ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．

階層プランニングの代表的なものとして ABSTRIPS がある．Sacerdoti は，Precondition リストに含まれる条件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 [63].

階層プランニングシステムは，最も高い緊急値が割り当てられた条件のみを考慮した抽象階層において骨格となるオペレータ系列を推論する．次いで，緊急値を一つずつ下げ，その時の緊急値を下回る条件を無視した抽象階層において，上位層で得られたオペレータ系列の細部をうめていく．最終的には緊急値が 1 以上の条件，すなわちすべての条件を考慮し，オペレータ系列を求める．緊急値の計算については後述する．

図 2.3 は，緊急値を用いた階層プランニングシステムの構成を表している．階層プランニングシステムに対する入力は,「オペレータ定義」,「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オペレータ系列である．ここで，階層プランニングシステムは，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条件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．図 2.3 におけるレベル n とした抽象階層である．そして，上位階層で得られた骨格となるプランを下位階層で順次補完する形でプランニングを進め，最終的に所望のプランを出力する．図 2.3 においては，レベル n-1，レベル n-2 と順次下位階層に進められる．また，途中の階層でプランを得ることができなかった場合は，一つ上の階層に戻り，別なオペレータ系列を探索する．

各抽象階層でのオペレータ系列の推論は，STRIPS[21] 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条件が現在状態で成立しているならば，プランニングを終了する．

(2) 各オペレータの Add リストと目標状態を比較し，Add リスト中の少なくとも一つ以上の条件が目標に含まれるようなオペレータを求める．求めたオペレータが複数存在する場合は，それぞれのオペレータの Precondition リストと現在状態を

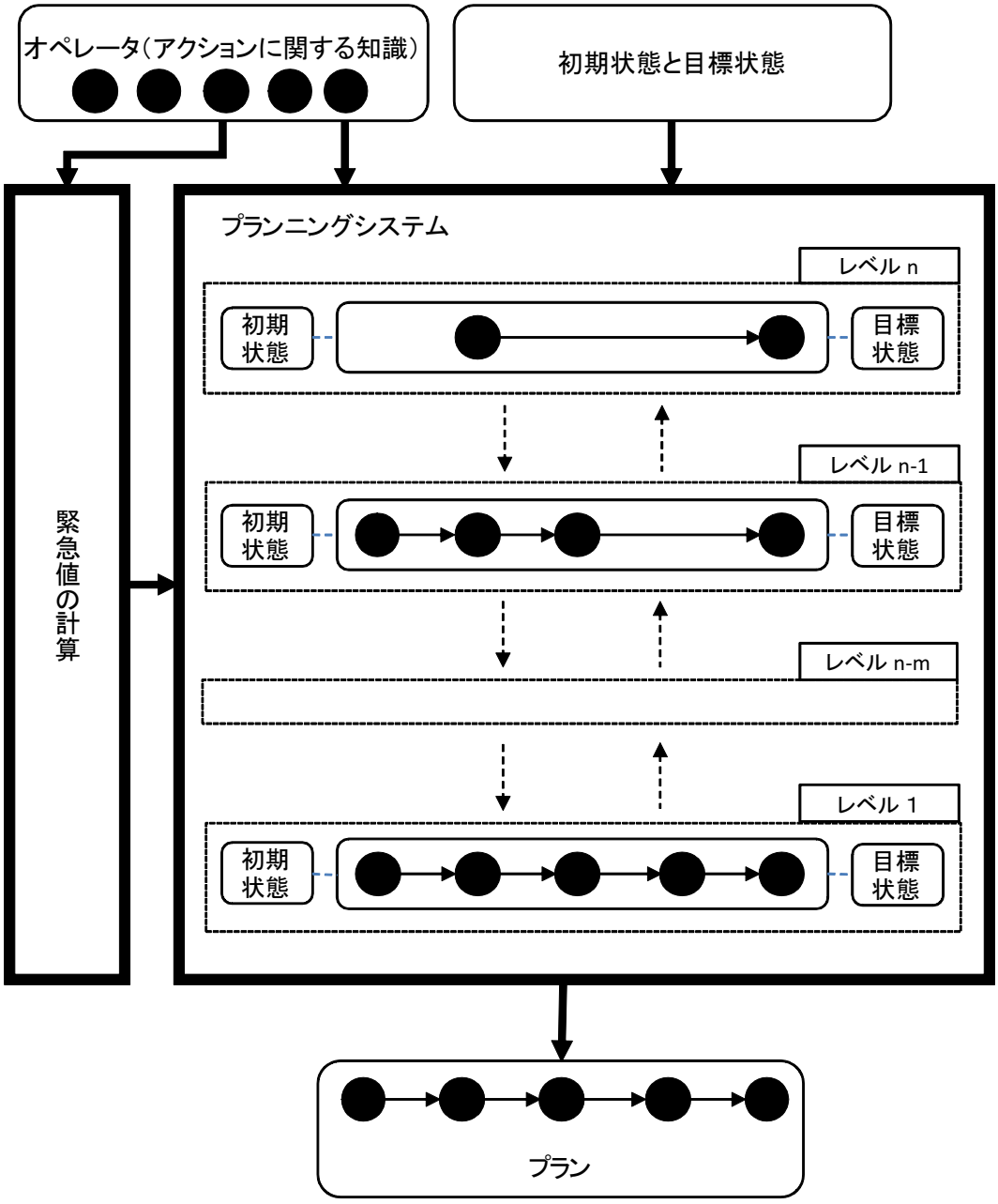


図 2.3 階層プランニングシステムの構成

マッチさせる．その結果，Precondition リスト中の条件群の中でマッチしない条件の数が最も小さいオペレータを一つ選択する．

(3) (2) で選択したオペレータの Precondition リストが現在状態で成立しているならば，そのオペレータの 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，選択されたオペレータの 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条件を新たな部分目標（サブゴール）として目標状態に追加し，(1) に戻る．

2.3.3　初期状態と目標状態の表現

オペレータ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，対象世界に存在する要素間の関係や構成要素の状態に関する情報からなる．たとえば，ロボットの行動計画においては，移動対象物に関する情報（箱，部屋，ドア）やそれぞれの位置関係，接続関係といった情報である．

```
Initial=[type(robot,robot),type(box2,box),type(box3,box),
        type(d1,door),type(d2,door),
        type(rPDP,room),type(rRAM,room),type(rCLK,room),
        pushable(box3),pushable(box2),
        connects(d1,rCLK,rRAM),connects(d2,rPDP,rCLK),
        connects(d1,rRAM,rCLK),
        connects(d2,rCLK,rPDP),
        inroomRobot(rRAM),inroom(box2,rCLK),inroom(box3,rCLK),
        status(d1,open),status(d2,open)]
Goal=[nextto(box2,box3),inroomRobot(rPDP)]
```

図 2.4 初期状態と目標状態の記述例

図 2.4 は，3 つの部屋（rPDP，rRAM，rCLK）があり，ロボットが部屋 rRAM に，そして 2 つの箱がそれぞれ異なる部屋にあるという初期状態（Initial) から，2 つの箱をまとめて，ロボットは部屋 rPDP に移動するという目標状態（Goal) を表している．

2.4　抽象階層の決定方法

前節で述べ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として Sacerdoti は次の指針を示している．

(1) オペレータの適用によって変化しない条件記述は最も高い緊急値を割り当てる．

(2) レベル K の条件 C_{K1} がレベル $L(> K)$ の条件 $C_{Li}(i = 1, ..., n)$ がすべて真である場合に達成できるならば，C_{K1} をレベル K とする．

しかし，この指針では自動的に緊急値を決定することはできない．また，得られた緊急値が探索効率を良くする指標ではないことが指摘されている [6]．よって，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，本節では，緊急値をオペレータ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述べる．前述のオペレータ定義を利用して，次の手順に従い，決定する．

(1) オペレータの分類木を作成する．

(2) オペレータの分類木から暫定緊急値を計算する．

(3) 条件の出現頻度により，最終決定する．

2.4.1 オペレータ分類木の作成

オペレータ定義から分類木を作成する．具体的には，各オペレータ定義の Add リストに含まれる条件を比較して，同一なものがあるかどうかを調べる．もし Add リストに同一条件があれば，各オペレータ定義の Precondition リストの中で同一条件があるかどうかを調べる．そして，Add リストおよび Precondition リストで同一条件から成る「抽象オペレータ」を求める．ここで条件が同一とは，条件名が同じでかつその引数の数が同じ場合をいう．ここで，引数は条件変数であっても，条件値のどちらであっても良い．抽象オペレータが複数得られた場合は，抽象オペレータ同士を比べ，これ以上抽象オペレータが得られなくなるまでこの比較を繰り返す．

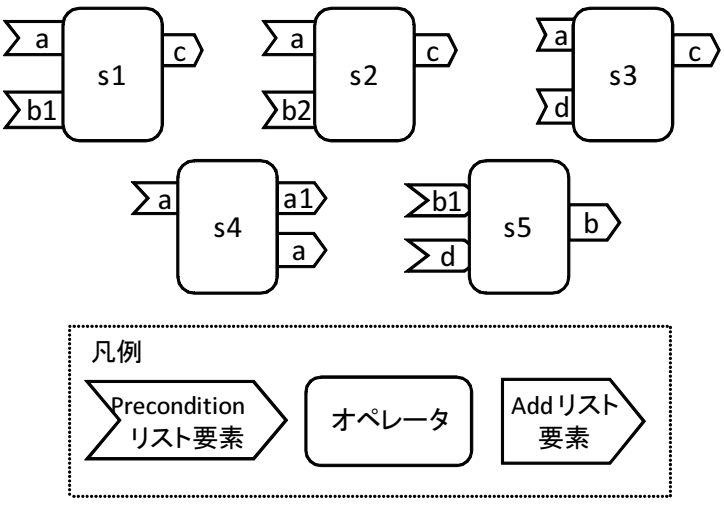


図 2.5 オペレータの分類木 (1)

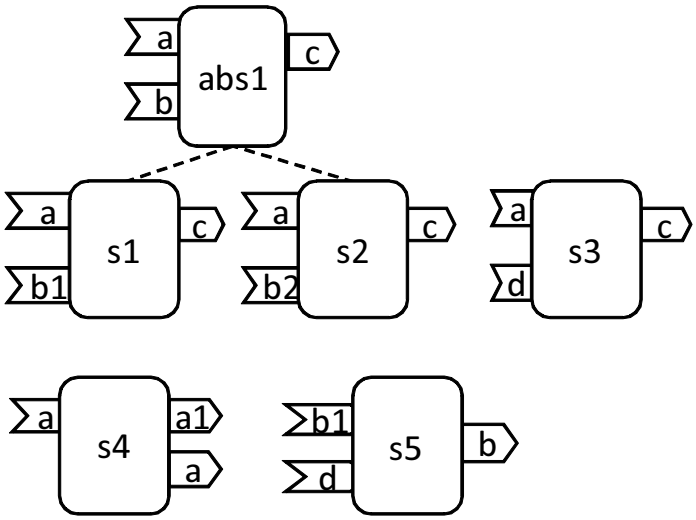


図 2.6 オペレータの分類木 (2)

図 2.5-2.8 は，5 つのオペレータ定義 (s1，s2，s3，s4，s5) に対するオペレータ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，オペレータ定義は簡略化しており，実際には図 2.2 に示すようなオペレータ定義が分類の対象となる．また，図 2.5-2.8 では，各々のオペレータの Precondition リストと Add リストに出現する条件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条件とアルファベットと数字を組み合わせた条件の 2 種類が表れているが，この違いは引数に相当する条件変数 (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条件として解釈する．

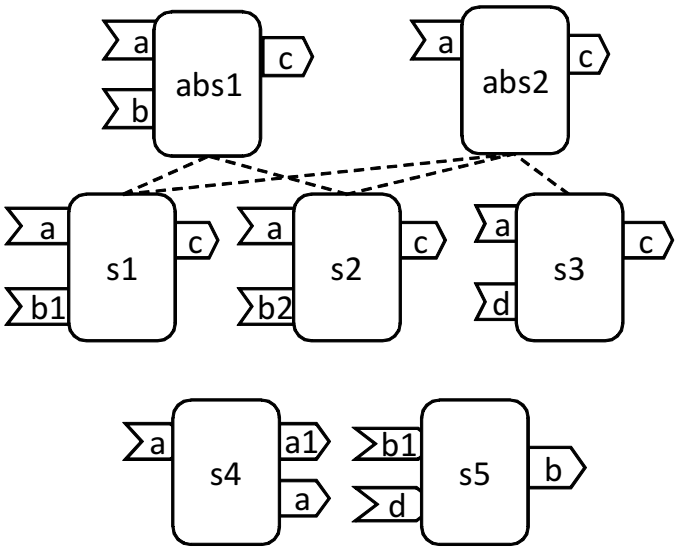


図 2.7 オペレータの分類木 (3)

図 2.5 に示した 5 つのオペレータにおいて，Add リストに c という同一条件をもつオ

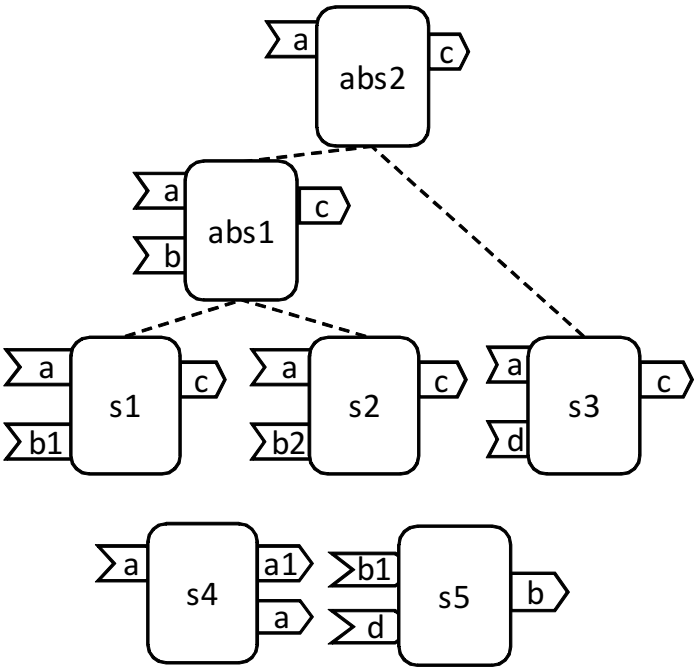


図 2.8 オペレータの分類木 (4)

ペレータは，s1，s2，s3 である．そこで，まずは，s1 と s2 それぞれの Precondition リストを比較する (図 2.6)．s1 の Precondition リストは条件 a と b1 から構成され，s2 の Precondition リストは条件 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 に条件 c を有し，Precondition リスト に条件 a と b (b は b1 と b2 を抽象化した条件とする) を有する抽象オペレータ abs1 が構成される．この様子を，図 2.6 に示す．

同様にしてオペレータ s3 を s1 および s2 と比較すると抽象オペレータ abs2 が構成される (図 2.7)．s1，s2，s3 からはこれ以上抽象オペレータは構成できないため，次に抽象オペレータ同士を比較する．すると abs1 と abs2 からは Add リストに条件 c からなり，Precondition リストは条件 a からなる抽象オペレータが構成できる．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 (図 2.8)．なお，元々の比較対象であるプリミティブなオペレータは，常に最下位にあるものとする．

最終的には図 2.8 に示す分類木が得られる．

2.4.2 緊急値の割当

オペレータ定義の Precondition リストに出現するすべての条件に対して，オペレータ分類木をもとに，暫定的な緊急値を割り当てる．Algorithm 1 の assign_critics で示すように，最上位の抽象オペレータの Precondition リストに出現する条件には，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オペレータ群の Precondition リストに出現する条件を調べる．そして，上位階層に出現した条件は除いて，この階層レベルで初めて出現する条件に $C_v - 1$ 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，Precondition リスト中のすべての条件に対して緊急値を割り当てる．

Algorithm 1 assign_critics(SH, PL, HL)

Input: SH:オペレータ分類木, PL:Precondition リスト中のすべての条件, HL:SH の高さ;

Output: HP:緊急値が付与された条件リスト;

```
L ← 1;
C_v ← HL;
HP ← [];
while L ≤ HL do
  PR : SH の上から L 番目にある各オペレータの Precondition リストに出現するすべての条件;
  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 のトップのオペレータ集合を 1 番目のレベルとする

オペレータの分類木は，オペレータの適用結果の重要度によってオペレータ群を分類したものであると同時に，オペレータの前提条件の重要度によって分類しているもとの考えられる．分類木の最上位に位置する抽象オペレータの前提条件は，その分類木のオペレータを適用する上で最も重要な前提条件と考えられる．なぜならば，この分類木に属するどのオペレータを適用する場合，必ず成立していなければならない条件だからである．したがって，分類木の上位に現れる条件ほど重要性が高い．

そこで，分類木の上位に位置する抽象オペレータの前提条件を調べる．ただし，こ

こでは前提条件の記述に現れない条件（たとえば，削除リストにのみ現れる条件）は最も低い緊急値とする．

前述の例では，5つのオペレータのPrecondition リストには4つの条件a，b1，b2，dがある．まず，最上位の抽象オペレータabs2のPrecondition リストには条件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，抽象オペレータabs1がある．抽象オペレータabs1のPrecondition リストは，条件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に緊急値として2を割り当てる．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，すべてのオペレータ定義のPrecondition リストの中に出現する条件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．出現頻度が多い条件は，現在状態でその条件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，条件b1はオペレータs1とs5のPrecondition リストに出現し，条件b2はオペレータs2のみに出現する．そこで，条件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，条件aに4，条件b1に3，条件b2に2，条件dに1が割り当てられる．

2.5 実験とその結果

ロボットの行動計画問題を例に，本方法の有効性を実験的に評価する．

本実験では，ABSTRIPS によるオペレータ表現に基づき，ロボットの行動に関する10 種類のオペレータを定義した．実験システムは，Sun 4260 上のQuintus Prolog により実装されている．

2.5.1 実験概要

比較対象として，Sacerdotiが示した方法で得られた抽象階層を用いる．表2.1に条件名と緊急値を示す．

ロボットの行動計画問題では，8種類の条件に対して緊急値が割り当てられる．

Sacerdotiの方法は，オペレータの適用によって変化しない条件，すなわち，オペレータの削除条件リストおよび追加条件リストには現れない条件が最上位に置かれる．具体的には，connects/3，type/2，pushable/1，locinroom/1の4つの条件が緊急値4となる．その他の条件については，先に述べたSacerdotiの経験則に従い決定する．たとえば，nextto/2はinroom/2やstatus/2の成立に依存するので，最下位となる．

本方法により得られた階層は，7レベルに分かれる．ほぼ1レベルに1条件が割り当てられたことになる．

評価実験では，次の4つの観点から評価を行う．

表 2.1 緊急値の割当

緊急値	本方法	ABSTRIPS
7	inroom/2	-
6	nextto/2	-
5	type/2	-
4	status/2	connects/3, type/2, pushable/1, locinroom/1
3	connects/3, pushable/1	inroom/2
2	locinroom/2	status/2
1	at/2	nextto/2, at/3

(1) 評価した枝の数：解を得るまでに評価の対象とされた枝の合計．探索空間の大きさを表す．各階層毎に記録する．

(2) 展開されたノード：解を得るまでに実際に展開されたノード数． 各階層毎に記録する．

(3) プランの長さ：プランを構成するオペレータの数．問題の複雑さを示す一つの指標であり，長いほど複雑である．

(4) 計算時間：最初の解を得るまでに要した時間．CPU タイム (単位: msec.) を測定する．

プラン生成は，ABSTRIPS によって行われる．得られるプランの長さが異なる 7 種類の問題（初期状態と目標状態）に対して，実際に得られたプランの長さ，探索時間，探索空間の大きさなどを記録する．なお，探索に要する時間が一定時間を超えた段階で探索を中止する．

2.5.2　実験結果

7 つの問題のうちの，一つの問題の測定結果を表 2.2 に示す．表 2.2 において，各レベルのかっこ内の数値は，そのレベルで実際に展開されたノードの数を表す．なお，本実験ではすべての問題に対して最適な解，すなわち，最短かつ実施可能なプランを得ることができたことを付しておく．

図 2.9 は，プランの長さと計算時間の関係を表したものである．このプランの長さは問題の複雑さを表す一つの指標として考えられる．図 2.9 は Sacerdoti の方法と本方法を比較したものである．

表 2.2 探索空間の大きさと計算時間

階層レベル	本方法	Sacerdoti の手法
1	24(5)	3(2)
2	17(5)	26(3)
3	0(0)	1(1)
4	1(1)	29(5)
5	0(0)	-
6	0(0)	-
7	0(0)	-
プランの長さ	11	11
計算時間 (msec)	1,066	1,750

抽象階層と計算時間の関係

図 2.9 より，抽象階層の決定方法によって計算時間に大きな違いが見られる．特にプランの長さが 11 を超えると顕著となっている．これは，Sacerdoti の方法が必ずしもすべての問題に対して有効でないことを示している．

抽象階層と探索空間の関係

Sacertodi の方法では，プランニングに要する探索空間の大きさ（すなわち，評価された枝の数）はおのおのの抽象階層での探索空間お総和である．たとえば，表 2.2 よりこの問題の探索空間は，Sacerdoti の方法が 59，本方法が 42 となる．この違いは探索に要する計算時間の違いとして表れている．

この原因を探るために，それぞれの方法において，各抽象階層毎の評価された枝の数と展開されたノードの数を比較する．表 2.2 から明らかなように，Sacerdoti の方法では上位階層に比べて，下位階層で評価された枝の数が多くなっている．一方，本方法は下位になるほど評価される枝の数が少ない．つまり，上位階層で作成された解が下位レベルの探索に対してうまくガイダンスとして作用していると考えられる．すなわち，問題を主要部分と細部にうまく分けられたことを示している．

2.6 考察

抽象化概念を用いたプランニングシステムでは，各々の抽象空間を基本空間（最初に与えられる問題空間）から抽象階層にしたがってドメインを定義する条件を無視する（実際は真であるとみなす）ことによってプランニングを進める．したがって，抽象階層を形成する上で最も重要な問題となることは，抽象レベルでの解が基本レベルでの解を得るためのガイドラインとして有効であるかどうかである．

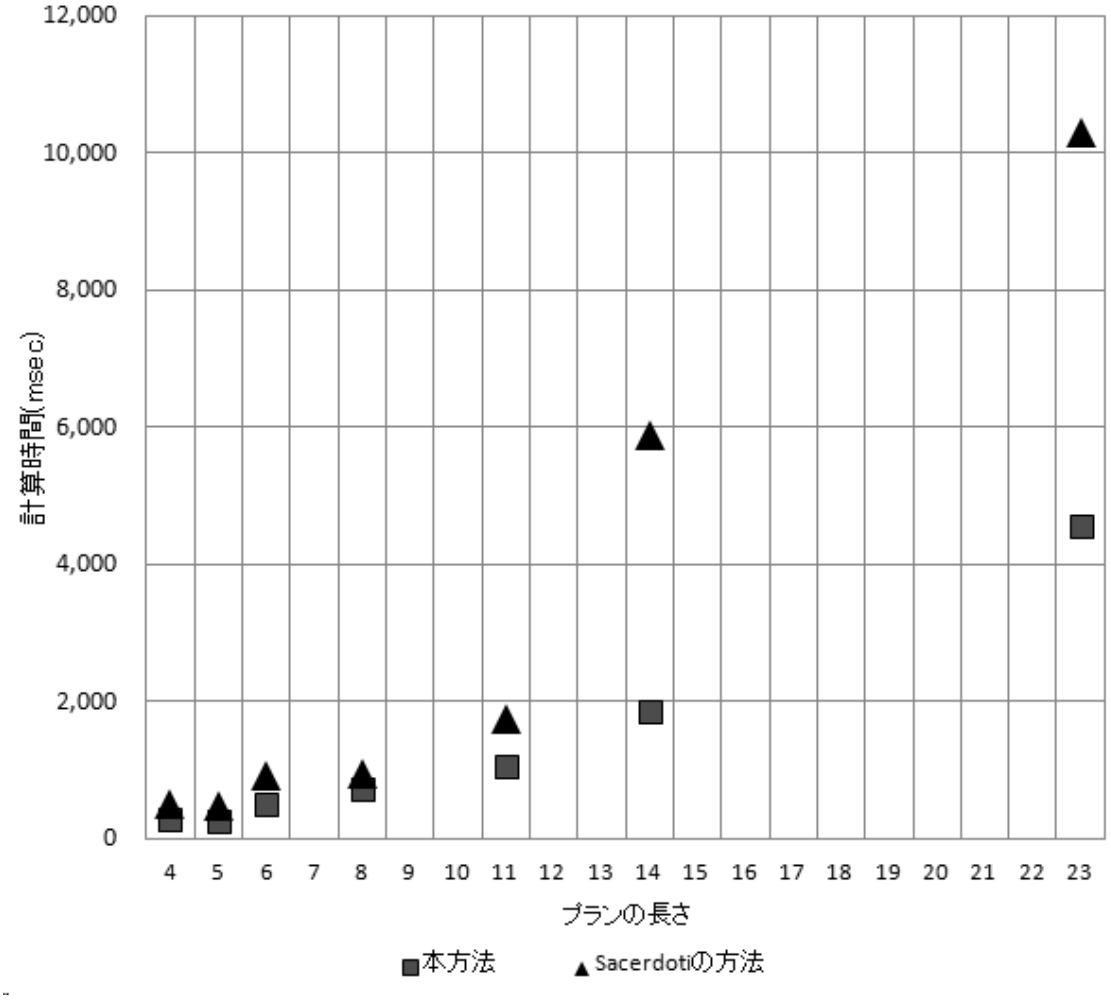


図 2.9 計算時間の比較

一言で抽象化といってもさまざまなレベルが存在し，[46] では抽象化のための変換規則を表 2.3 のように定義している．たとえば，drop_predicate 変換は問題記述から特定な述語の具体例をすべて削除してしまうというものである．また，drop_goal 変換は目標からある特定の部分目標を無視するものである．

ABSTRIPS は drop_precondition 変換であり，その変換は前提条件の緊急値に従う．

抽象階層の有効性という観点からは，前述の実験結果が示すように，本方法が Sacerdoti が与えた経験則よりも優れていることが明らかになった．

そして，今回の実験から抽象階層を決定する規則として以下の 2 つを得た．

(1) オペレータの適用結果の類似性　オペレータの適用結果の類似性は，問題の主要部分と付随する部分を分類する上で重要な情報となる．すなわち，類似する結果をもたらすオペレータの前提条件ほど重要なものということである．

表 2.3 抽象化規則

変換規則	変換内容
drop_predicate(P)	問題記述から述語 P の具体例をおとす
drop_goal(G)	ゴールからサブゴール G をおとす
drop_precondition(P,O)	オペレータ O の前提条件 P をおとす
count(P)	述語 P をそれを満足するオブジェクトの数で置き換える
compose_numbers(M,N)	2 つの番号をその和で置き換える
drop_number(N)	問題記述から番号が N の具体例をおとす
number_to_parity(N)	番号 N を 2 進数に置き換える
compose_parities(B,C)	2 つの 2 進数をその和で置き換える
drop_parity(B)	問題記述から 2 進数 B の具体例をおとす

(2) 前提条件リストの出現頻度　各オペレータの前提条件に現れる頻度がその条件の重要度を決定する上で重要である．すなわち，出現頻度の高い条件は与えられる問題の中で重要なものと考えられる．したがって，こうした条件は上位レベルに置くのが良い．

Tenenberg は，オペレータの分類木を用いてプランニングシステムが生成したプリミティブなプランを抽象化する方法を提案している [71, 72]．この方法では，プラングラフと呼ばれる構造にプランを変換する．プラングラフはプリミティブプランと各オペレータの分類木からなり，抽象階層に沿った探索をガイドするために用いられる．

Tenenberg のシステムでは，プランニングの結果を再利用するための方法を与えている．ここで重要なことはオペレータの分類木の与え方である．Tenenberg は具体的な言及をしていないが，分類木の持つ性質を次のように述べている.「分類木とは，オペレータの前提条件と結果をオペレータの特徴としてとらえ，その特徴の継承関係を階層的に分類したものである．したがって，下位レベルで作用可能なオペレータはより上位のオペレータの前提条件と結果を含む.」すなわち，分類木はオペレータの特徴の重要度によって分類しており，分類木におけるオペレータの位置によって，そのオペレータの前提条件の重要度を決定できると述べている．

この考え方は，本方法と一致するものであり，階層プランニングにおける一つの経験則として考えることができる．

以上のように，本方法により，プランニングの効率化に寄与する抽象階層を自動的に得ることができたが，いくつかの課題が残った．

(1) ガイドラインとして抽象プランが有効に働かないケースがある

実験中，上位レベルでの解が下位レベルで破棄されるというケースが発生した．これは抽象レベルのプランが必ずしも下位レベルの探索のガイドラインとして有効に働いていないことを示している．

Cheng は Subgoal Ordering 問題と呼ばれる部分目標間の制約がプランニングにおいて重要であることを論じている [12]．たとえば，次の例を考える．

$$\text{inroom}(robot, r_1)\ \&\ \text{inroom}(box_a, r_1)\ \&\ \text{nextto}(robot, box_a)\ \&$$
$$\text{inroom}(box_b, r_2)\ \&\ \text{inroom}(box_c, r_2)\ \&\ \text{nextto}(box_b, box_c)$$

ロボットが部屋 1 で箱 a に隣接し，a とは異なる箱 b が部屋 2 で箱 c に隣接していることを表している．これらの部分目標において，$\text{inroom}(robot, r_1)$ という条件は $\text{nextto}(box_b, box_c)$ という条件の成立によって達成できる．しかしながら，本方法では，inroom 条件が nextto 条件よりも重要度が高いと判断されるために，先に inroom 条件について問題を解いてしまう．結果として，一度作成されたプランが破棄されるなどして探索効率を下げてしまう可能性がある．

上位レベルの解が破棄された原因はこの Subgoal Ordering 問題が深く関わっている．

本方法で得られる抽象階層はその領域にとって前提条件がどれくらい共通して用いられているかといった『領域に対する重要性』に従って並べらたものである．Subgoal Ordering 問題を解決するためには条件の重要性ではなく，達成の困難さ，すなわち，その前提条件をオペレータを使って達成するための難しさによって抽象階層を生成する必要がある．一般に達成の難しさは解くべき個々の問題に依存するが，前提条件間の相互作用（一方の成立が他方を阻止する）が考慮されなければならない．

ところが，本方法はこうした制約を考慮せず，共通部分のみを重要視している．すなわち，ある目的を達成するために重要と思われる条件がより上位レベルであろうという仮説に基づいている．

Knoblock は部分目標間に依存関係があるケースでも，ドメイン知識から有効な抽象階層の知識を得るための得るための方法を提案した [31, 32, 33, 34]．ドメイン知識は 2 種類あり，一つはオペレータの変数のタイプ，もう一つはオペレータ知識に表現できないドメインの基準がある．たとえば，ロボットがドアのそばにいるとき，そのロボットはドアの隣接する部屋にいるといった知識である．

このドメインの公理を得ることは比較的簡単であると Knoblock は主張しているが，実はこの公理がないと十分に良い階層を得ることが困難である．問題によっては条件がすべて一つのノードにまとまってしまう場合もある．

Knoblock の方法では，オペレータに関する知識以外にドメインにおける制約条件を与える必要があり，ABSTRIPS で人間が初期抽象階層を与えることに等しい負担を強いることになる．

(2) 個々の問題により抽象階層が変わるケースがある

良い抽象階層は領域依存であるとともに個々の具体的な問題例に依存するケースもある．本方法は，抽象階層の推論は事前に一度だけ行えば良いが，上述のように解が得られない場合もある．従って，問題の性質に応じた（特に相互作用について）抽象階層を求める必要性がある．また，ある問題に対して抽象階層が一つに定まることは稀で幾つか候補が存在するはずである．しかしながら，こうしたアプローチはプランニング時にどの抽象階層を選択するかによって，プランニング能力に差が出てしまう場合にはその選択方法が重要となる．

最後に，表 2.2 に示したように，評価した枝の数のうち実際に展開されたノードの数が少ないことも明らかになった．今回実験を行った問題でみると，平均で 30%程度しか展開されていない．70%は評価したにも関わらず展開すらされないということになる．これは，そもそも STRIPS の欠点であり，オペレータの前提条件の達成度をもっと厳密に評価する方法が求められる．

2.7 まとめ

プランニングを高速化するために，階層プランニングシステム ABSTRIPS で提案された緊急値による抽象階層の生成方法について，その問題を述べ，オペレータの適用結果を考慮した抽象階層の自動決定方法について述べた．ロボット行動計画問題を例に，実験を行い，その有効性を検証した．

本方法により得られた抽象階層はプランニングを効率よく行うことができるが，解決すべき問題によっては有効に働かない可能性もある．具体的には，一度生成された抽象プランを破棄してしまう，動的な生成の必要性という課題が残されている．

第 3 章

階層プランニングによるWebサービスの自動合成

Web サービスはそれ自体が一つのソフトウェアコンポーネントであり，状況に合わせて複数の Web サービスを動的に合成することで，全体として一つの Web サービスとすることもできる．しかしながら，与えられた目的に対して，個々の Web サービスをどの順序で実行するかについての支援が必要となる．本章では，この Web サービス合成問題に対する AI プランニング技術の応用について述べる．多数の利用可能な Web サービスから適切なものを選択し，迅速かつ効率よく Web サービスを合成する方法を提案する．

3.1 研究の背景

Web サービスを使ったアプリケーション開発では，必要な機能を動的に合成することによって，全体として一つのサービスが構築される．

たとえば，図 3.1 では，オンラインショップの注文から配送までの一連のフロー（購買フローと呼ぶ）を実現する例を示している．この購買フローは，ユーザの認証が行われた後，ユーザからの注文をカートに入れて，在庫状況に応じて，新規に調達または在庫引当を行い，配送手続きを行う．個々のアクション（図 3.1 の左側の認証から配送まで）は，それぞれイントラネット上もしくはインターネット上で提供されるサービスが呼び出される（図 3.1 の右側の認証から配送処理まで）．また，特に外部のサービスに関しては，複数種類のサービスが存在する場合がある．アプリケーションの開発者は，目的にあわせて適宜利用するサービスを選択しなければならない．

この例では，予めフローという形で全体の動きが提供されているが，場合によっては，達成したい目的しか与えられておらず，どういう順序で個々のサービスを実行すれば良いかということまで含めて考えることが必要となる．

Web サービスの合成問題とは，順序まで含めたサービス群の組み合わせ方法を考える問題である．

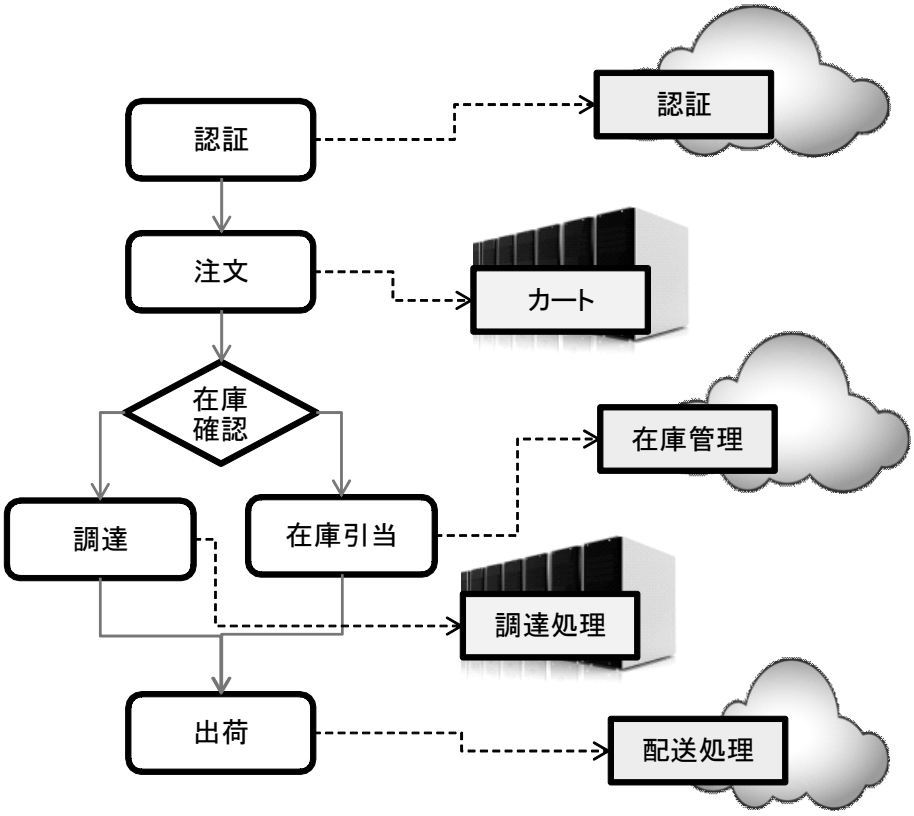


図 3.1 Web サービスとは

図 3.2 に示す例では,「利用者の好みを満たしかつ宿泊先に近いレストランを検索し,そのレストランへ行く交通手段を提供するサービス」を構築することを考える．ここに，2 つの独立した Web サービス「お好みのレストランを検索し出力するサービス（以下，レストラン検索サービス)」と「出発地から目的地へ向かうための交通手段を出力するサービス（以下，経路探索サービス)」があったとする．新サービスは，これらの２つのサービスを組み合わせることによって実現できる．まず，経路探索サービスによって，出発地から目的地までの経路図を得るという目的を達成することができる．しかしながら，経路図を得るためには，目的地の住所が必要となり，これは，初期条件から直接得ることはできない．幸いにも，レストラン検索サービスにより，お好みのレストランの住所を得ることができるので，この住所をもとにレストランまでの交通手段を探索すればよい（図 3.2）.

なお，ここで,「宿泊先」「郵便番号」「住所」などサービス間を流れるデータはインスタンス（実データ）ではなく，データ名称またはデータタイプに相当するものである.

この例はたかだか 2 つのサービスの組み合わせではあるが，目標とする「現在地からレストランへの交通手段を得る」には，レストラン検索サービスの後に経路探索サービスを行うというサービスの呼び出し間に順序関係が存在することがわかる.

こうした複数の Web サービスを合成する作業は，実際には設計者のスキルや経験に依存する作業である．その理由を，Rao らは次のように整理している [61].

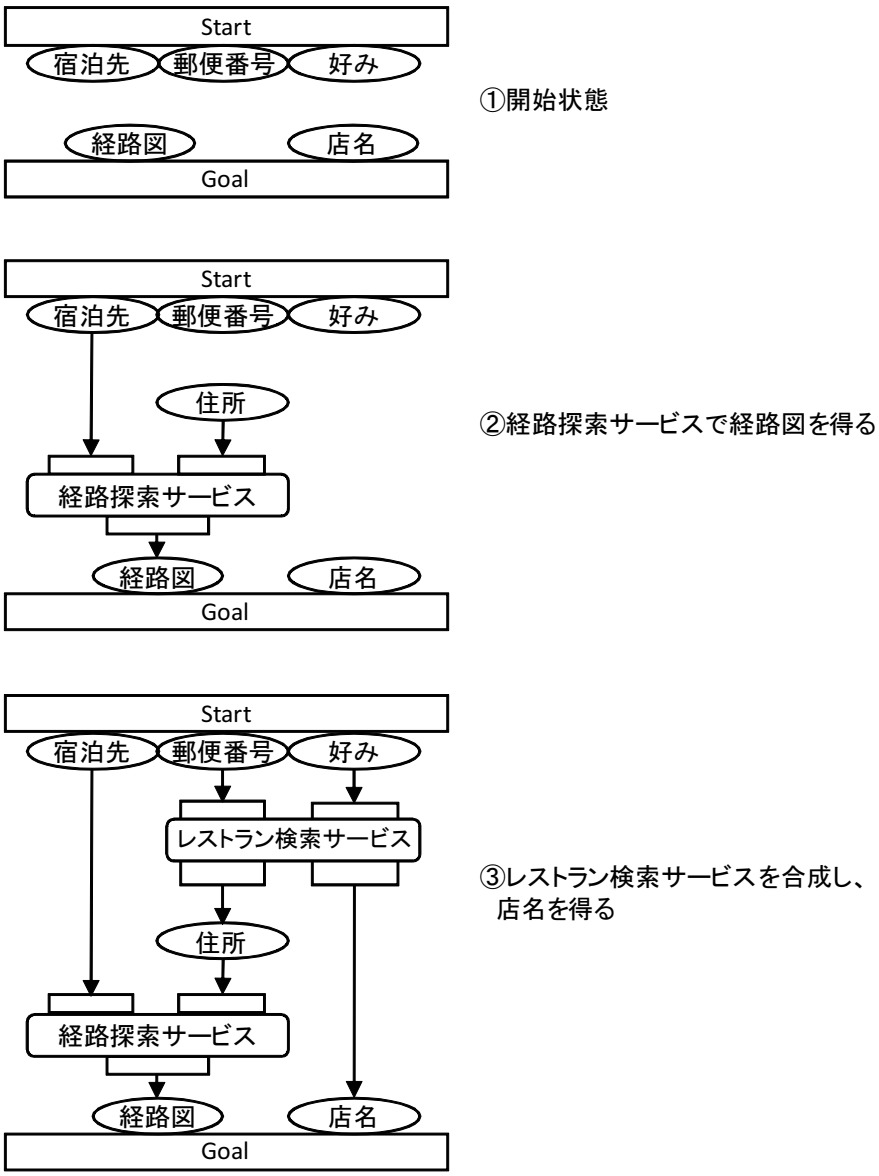


図 3.2 Web サービスの合成の例

- 様々な種類の Web サービスがインターネット上で提供されており，その数は増加の一途をたどっている．
- 個々の Web サービスは，異なるサービス提供者によって作成，管理されており，各サービス提供者が各自の都合で適宜追加や変更を加えている．実際に，類似サービスが追加されたり，また既存サービスの更新もしばしば行われる．

こうした理由から，Web サービスの合成は動的に行われることが求められている．

こうした要請に対して，複数のサービスを合成するために，BPEL に代表される標準的なプロセス定義言語が提案されている．BPEL は，複雑なサービスを定義したり，合成結果を再利用する仕組みを提供しており，BPEL に対応したプロセス定義ツールを利用することによって，設計作業の効率化を図れる．しかし，BPEL そのものはサー

ビスの動的な合成機能を提供しているわけではない．実際には，必要な Web サービスの発見や合成作業は設計者のマニュアル作業であり，作業効率が悪く，品質確保も難しい．合成対象となる Web サービスの数が少ない場合には，さほどの組み合わせ数にならないので，設計者自身で考えることも可能ではあるが，該当する Web サービスの数が多くなるにつれて，組み合わせ数も多くなり，人手で処理するのは困難となる．特に，類似の出力データを生成する Web サービスが多く存在する場合は適切な Web サービスを選択するコストが問題となる．

Web サービスは，WSDL と呼ばれるサービス定義言語によって表現される．WSDL では Web サービスの入出力データの型を定義することができ，適切なサービスを選択するための必要最小限の情報を提供する．しかし，実際の開発では，サービスの挙動（どういう状態変化を引き起こすか）も考慮して，合成が行われるが，WSDL ではこうした情報を表現する手段を提供しない．

以上の背景から，本研究では，現状，設計者がマニュアルで行っている Web サービスの発見とサービス合成作業を支援する方法を提案し，評価を行う．

なお，Web サービス合成を完全に自動化することは現実的なアプローチとはいえず，むしろ設計者の介在を前提として，設計者の作業を支援するアプローチの方がより実際的といえる [55, 59, 65]．また，一般に，合成対象となる個々の Web サービスを何ら検証もせずに利用することは稀であり，事前検証済みで実行時の動作がある程度保証されたものが利用される．以上の前提条件のもとで研究の詳細を述べる．

3.2　本章の構成

事前に動作保証がある Web サービスが多数存在するという前提において，設計者が所望の目的を達成するためにサービス合成を必要とする状況を想定し，その際の設計者の負担を軽減する Web サービスの動的な合成手法を提案する．具体的には，階層プランニング技術を用い，サービス合成を効率化することに主眼を置く．

研究課題

Web サービスの自動合成問題に対してプランニング技術を適用するにあたって，以下の研究課題に取り組む．

- Web サービス合成を自動化し，さらに効率の良い合成を行うための手法
- 既存の Web サービスの定義からオペレータ定義への変換方法

課題を解決するための手段

Web サービスの定義言語である Web Services Description Language から，オペレータ定義への変換を行う．変換された各オペレータ定義に対して階層プランニング技術を適用する．

評価結果 Web サービスの標準的な定義から比較的容易にプランニングのための情報を得ることができる．実験により，Web サービスの自動合成の効率の良さを検証した．

以下，3.3 節では，階層プランニング技術を応用した Web サービス合成の手法を提案する．階層プランニングを応用するにあたって既存の Web サービス定義をどのように利用するか，システムの構成を詳しく説明する．3.4 節では試作したシステムの評価を行う．3.5 節で，本研究と関連する研究と比較し，3.6 節で今後の課題も含め，まとめとする．

3.3 階層プランニングの応用

複数の Web サービスを組み合わせることによって，所望の目的を達成する問題は，近年 AI プランニング問題として議論が活発に行われている [10, 11, 18, 35, 61, 65]．元々，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いかに効率よく求めるかを追究する研究分野であり，そこでは問題の表現，推論，探索といった基礎的な内容を扱う [48]．Peer[58, 59] は，古典的な AI プランニングと Web サービスの合成問題の類似点，相違点を整理しているが，古典的な AI プランニングは，プランニングに必要な情報や知識はすべて事前に定義されており，しかもプランニング時にはその定義は原則変化しないという前提のもとで議論されていた．しかしながら，一般に Web サービスは WSDL によってインタフェースは規定されているものの，サービスの挙動は実行時に決まるものであり，その他の不確実要素を含むことが前提となっている．したがって，実際には古典的なプランニング問題よりもはるかに困難な問題であるという指摘もある．

AI プランニングは，与えられた目標を達成する一連のアクション系列を求める技術である．一方，Web サービス合成とは前節で述べたようにサービスリクエスタ（設計者）が要求する目標に対して，Web サービスの系列を求めることである．いずれの技術においても重要な課題は，いかにして組み合わせの爆発を抑え，効率の良い解の探索を行うかにある．

実際に，AI プランニングの技術をサービス合成に適用するためには，合成対象とする Web サービスの定義と効率の良い推論方式が技術的な課題となる．

一般的なソフトウェアシステムの開発において，予め用意されているコンポーネントを組み合わせて再利用することは頻繁に行われている．しかしながら，どのコンポーネントをどのように組み合わせるかに関しては開発者のスキルが必要となるため，自動合成などさまざまな支援が試みられてきた．たとえば，[27] では，リアルタイムシステム設計時のプロトタイピングを支援する目的で，プランニングベースのコンポーネント合成の方法を提案した．それぞれのコンポーネントは利用する際の前提条件、結果、入出力条件などを定義し，ゴール仕様を与えることによって，必要なコンポーネ

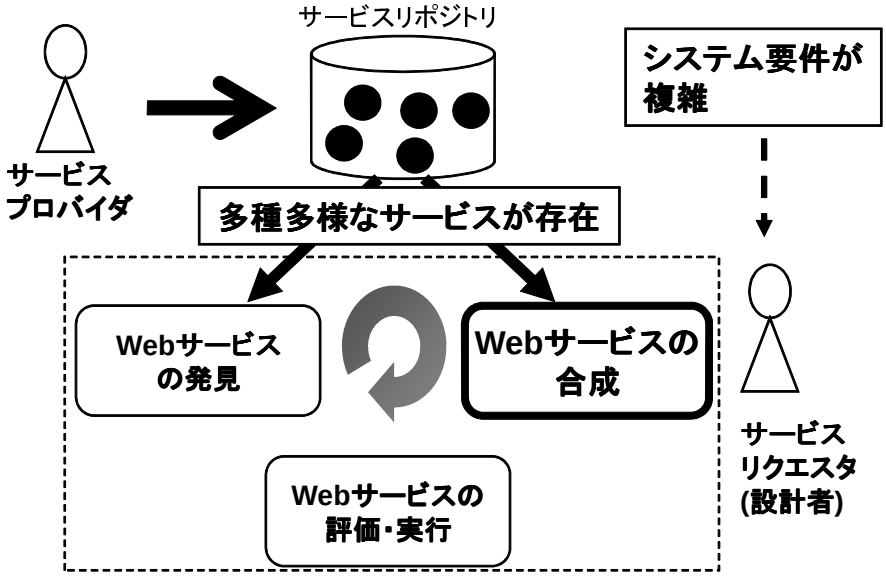


図 3.3 Web サービス合成の課題

ントを自動的に組み合わせることができる．Web サービスもソフトウェアコンポーネントであることから，このアプローチが適用できると考えた．

以下の各節で，詳細を説明する．

3.3.1 Web サービスの定義

代表的な Web サービスを定義する言語として，Web Service Description Language（WSDL）[2] がある．WSDL は，XML ベースの言語仕様として規定されており，Web サービスがどのような機能をもつのか，それを利用するためにはどのような条件を満足すれば良いのかなど記述する方法である．

以下は，WSDL の定義を構成する主な要素である．

- 抽象的定義部分

  wsdl:types 要素 メッセージのフォーマットを定義する際に使用する型を抽象的に定義する．メッセージ定義にユーザ定義の型を使用する場合は，型を定義しておく必要がある．

  wsdl:message 要素 Web サービスで使用するメッセージのフォーマットを抽象的に定義する．

  wsdl:operation 要素 入力と出力とフォルトメッセージ出力を行う処理の 1 単位である操作（operation）を抽象的に定義する．入出力メッセージやエラー情報を通知するために使用するフォルトメッセージのフォーマットとして，wsdl:message 要素で定義されたフォーマットを割り当てる．

wsdl:portType 要素　関連する操作をひとまとめにした抽象的なポートである，ポートタイプ（portType）を定義する．

- 低レベル定義部分

wsdl:binding 要素 wsdl:portType 要素で定義されたポートタイプ内の個々の抽象的な操作に具体的な通信プロトコルをバインドする．ポートタイプの定義に通信プロトコルをバインドした定義のことをバインディング（binding）と呼ぶ．具体的な通信プロトコルへのバインドは，拡張性要素を使用して記述する

wsdl:port 要素 wsdl:binding 要素で定義されたバインディングに通信エンドポイントのネットワークアドレスをバインドして具体的なポートを定義する．ネットワークアドレスのバインドは拡張性要素を使用して記述する．

wsdl:service 要素 wsdl:port 要素で定義したポートのうち，関連するポートをひとまとめにしたサービスを定義する．

上記の仕様に基づき定義されたサービスを WSDL 文書と呼ぶ．

図 3.4 は，前述のレストラン検索サービスを WSDL 文書として定義したものの一部を表している．図中の点線枠内は，抽象的定義部分に相当する．

3.3.2 WSDL からオペレータ表現への変換

階層プランニングシステムが動作するには，WSDL による定義だけでは不十分である．たとえば，入力データとしては現れないがそのサービスを適用のための前提条件やサービスの実行結果も，階層プランニングシステムにとって重要な情報となる．そこで，本手法では，図 3.5 に示す形式で，Web サービス仕様を定義する．

この Web サービス仕様は，precondition リスト, add リスト, delete リスト からなる．precondition リストには，Web サービスを実行する上で必要な入力データおよびそのサービスを適用する前提条件を定義する．add リストと delete リストはそれぞれ Web サービスの実行結果として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，Web サービスの実行の過程で消費されるデータ（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の仕様を定義する．なお，入力および出力データと各条件は，属性名＋属性変数（もしくは値）として定義し，以降では属性と呼ぶことにする．また，属性変数はアルファベットの大文字で始まる文字列，属性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．

WSDL 文書において，wsdl:input 要素は precondition リストの要素となりうる．また，wsdl:output 要素は，add リストの要素となりうる．この時，wsdl:message 要素も考慮される．また，WSDL 文書で定義されていないシステムの状態に関して追記することになる．

```
<wsdl:definitions...

<wsdl:message name="findRestaurantService">
  <wsdl:part name="Zip" type="xsd:int"/>
  <wsdl:part name="Pref" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="findRestaurantResponse">
  <wsdl:part name="Name" type="xsd:string"/>
  <wsdl:part name="Address" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="findRestaurant">

 <wsdl:operation name="findRestaurant"
                 parameterOrder="Zip Pref">
  <wsdl:input name="findRestaurantService"
                 message="impl:findRestaurantService"/>
  <wsdl:output name="findRestaurantResponse"
                 message="impl:findRestaurantResponse"/>
 </wsdl:operation>

</wsdl:portType>

<wsdl:binding name="findRestaurantSoapBinding" type="impl:Estimate">
・・省略・・
</wsdl:binding>

<wsdl:service name="findRestaurantService">
・・省略・・
</wsdl:service>

</wsdl:definitions>
```

図 3.4 WSDL 文書の例

たとえば，図 3.4 に示した「レストラン検索サービス」の定義は次のようになる．

precondition リストには，　入力データである郵便番号（Zip）と利用者の好み（Pref）に加えて，Web サービスがその地域で利用できること（is_available）と，ユーザが与える好みが事前にシステムで扱える情報であること (in_category)，そして現在のシステムの内部状態が「検索指定モード」であることを表す属性（in_process(search_view)）が定義される．ここで，hold() は Web サービスの入出力データを表す属性である．

また，このサービスの実行によって，お店の名前（Name）と住所（Address）が出力データとして得られ，そしてシステムの内部状態は「結果表示モード」であることを表す属性 (in_process(result_view)) に切り替わる．なお，入力データとして与えられる「好み（Pref)」はサービス適用時にシステム内部で消費され，サービス適用前に成立していたシステムの内部状態（in_process(search_view)）も切り替わることから，Delete リストには，サービス適用後に成立しないこれらの 2 つの属性が定義される．

```
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]
```

図 3.5 オペレータ形式での Web サービス仕様の定義

```
opname ::= レストラン検索
precondition::=[
        hold(Zip), hold(Pref), is_available(Zip),
        in_category(Pref), in_process(search_view)]
delete ::=[
        hold(Pref), in_process(search_view)]
add ::=[
        hold(Address), hold(Name), in_process(result_view) ]
```

図 3.6 レストラン検索サービスの定義例

階層プランニングシステムは初期状態として入力データの仕様およびサービスの実行で必要な情報が，そして目標状態として出力データの仕様が与えられる．そして，その目標状態を達成する Web サービス群とその順序関係を推論する．最初の Web サービスが初期状態に適用され，新しい状態を生成し，続く Web サービス群がその新しい状態に対して順に適用されていくことで，最終的に目標状態に達することになる．

また，別な例として，複数のサービス提供サイトをまたがったオンラインショッピングサイトの構築サービスについて説明する．これは，オンラインカタログの表示から商品購入まで一連のサービスを行う．商品カタログの表示，ショッピングカートへの追加および削除，注文手続き，配送手続きなどは，それぞれ Web サービスとして提供されているものとする．

図 3.7 は，商品リストに存在する商品をショッピングカートに入れるというサービスの記述例である．ここで，VX，OBJ1，OBJ2 といったアルファベットの大文字で表されたものは属性変数であり，プランニングの過程では実際の値に置き換えられる．

この Web サービスは，あるユーザがブラウザから指定した商品アイテム（ItemX）を入力データとする．そして，その商品がカタログに存在し，かつブラウザの画面状態がカートに追加可能なモードである場合に実行される（属性変数 VX は利用者識別子にマッチする）．サービスの実行結果は，入力データ（商品アイテム）および商品アイテムに付随するデータがシステム内部で消費される．さらに，そのユーザのショッピ

```
opname ::= addToCart
precondition::=[ in_catalog(ItemX),
                 hold(user,ItemX), on_screen(user,VX),
                 on_screen(ItemX,VX), cart_status(Cart,VX) ]
delete ::=[hold(user,OBJ1), hold(ItemX,OBJ2),
                 in_mycart(ItemX,List), hold(OBJ3,ItemX)]
add ::=[ in_mycart(ItemX,Cart), hold(user,ItemX)]
```

図 3.7 ショッピングカートサービスの定義例

ングカートには商品アイテムが追加され，再び出力データとして商品アイテムを返す．

初期条件としては，Web サービス間の遷移関係（商品一覧のサービスの後に実施可能な Web サービスとしてどのようなものがあるか，サービスの利用可否の状態，商品カタログに掲載されている商品例など）を記述する．また，目標としては，たとえば，特定商品がカート内に存在するとか，配送状態にするといった内容を定義する．

3.3.3　サービス合成のための仕様

Web サービスを合成するために与えられる仕様は，プランニングに必要となる初期条件と目標として定義される．それぞれシステム外部から与えられる入力データ仕様と最終的にシステムから得られる出力データ仕様の記述から構成される．各々のデータ仕様は属性の集合から構成される．また，初期条件には入出力データ以外にも，データタイプやインスタンスに関連した定義が含まれる．

たとえば，先のオンラインショッピングの例では，初期条件として，購入するユーザ名，商品名，カタログ情報などが定義の対象となる．また，目標として，購入したい商品の仕様，価格，在庫の有無などが定義される．

3.3.4　システム概要

図 3.8 に本システムの概念図を示す．

サービス提供者はあらかじめサービスを定義し，サービスリポジトリに格納する．また，必要に応じて，緊急値の計算部がサービス定義より緊急値を計算する．サービス合成部は，サービス設計者から与えられる外部仕様（初期条件と目標）に対して，初期条件から目標を達成するための Web サービス系列を合成する．ここで，階層プランニングシステムは，緊急値に従って決定される抽象階層（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）の上位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プラン（骨格となるサービス系列）を下位階層で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（合成結果）．

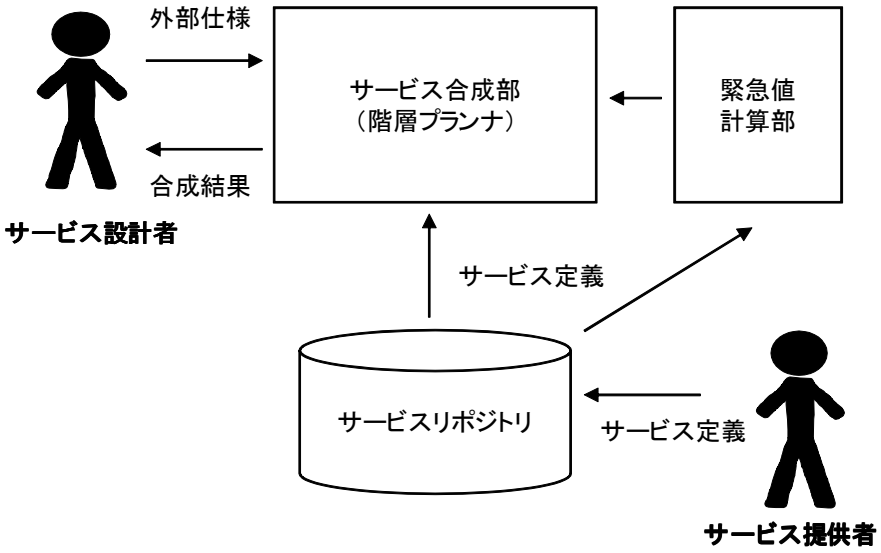


図 3.8 システム概要

サービス合成部は，第 2 章で述べた階層プランニングシステムと同じ方式で動作する．すなわち，まず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，上位階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．なお，各抽象階層でのサービス系列の推論は，STRIPS[21] で採用された以下に述べる手段目的分析の方法に従う．

1. 目標と現在状態を比較する．目標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場合，目標は達成されたものとする．なお，プランニングの開始時は現在状態は初期条件となる．

2. 各 Web サービスの add リストと目標をマッチさせて，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 Web サービスを求める．求めた Web 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 Web サービスを一つ選択する．．

3. 2. で選択した Web サービスが現在状態において適用可能である（すなわち，その precondition リストが現在状態で成立している）ならば，そのサービスを適用し，現在状態を更新する．しかし，選択されたサービスが適用可能でない場合は，一致しない属性を新たなサブゴールとして目標に追加し，1) に戻る以上の結果，最終的に初期条件から目標を達成するためのサービス系列が求められる．

3.4　実験とその結果

本手法の有効性を示すために，Sacerdoti の方法との比較実験を行った． Web サービス仕様（10 種類）を用いて，4 つの具体的なケース（それぞれの合成結果は異なる）に対して，実際に解が得られるまでの実行時間および探索空間のサイズを測定した．

3.4.1　緊急値の割り当て

表 3.1 に本手法と Sacerdoti の方法によって得られた緊急値を示す．数字が大きくほどより上位の抽象階層に位置づけられることを意味する．

表 3.1 緊急値の割り当て

属性名	説明	本手法	ABSTRIPS
on_screen	クライアント画面状態	7	3
hold	入出力データ	6	1
datatype	データタイプ	5	4
service_status	サービスの状態	4	2
business_flow	画面の遷移	3	4
in_catalog	商品とカタログの関係	3	4
cart_status	カートの利用状態	2	4
in_mycart	個人用カートの内容	1	1

3.4.2　実験結果

4 つのケース（初期条件と目標）に対して，表 3.1 の緊急値を用いてサービス合成を行った．

実際にどんなプランができるのか．１つのケースについて説明する．

その際，探索空間上で評価された枝の合計と計算時間を測定した（各ケース３回ずつ測定し平均値を算出）．なお，本プランニングシステムの実験環境は，ハードウェアが Sun4 260，実装言語は Quintus Prolog である．

表 3.2 および表 3.3 は，表 3.1 に示した緊急値に従って階層プランニングを行った際の探索空間のサイズをケース毎 (No.1 から 4) にまとめたものである．「サービス合成の結果」という行に示した数字は，最終的に得られたサービス系列の長さを示す．なお，表 3.2 および表 3.3 に示すように，本手法と Sacerdoti の手法で得られたサービス系列はいずれのケースも長さおよび内容は同一であった．

また，「抽象階層」という行の１から始まる数字は，表 3.1 の緊急値に基づいて構成された抽象階層のレベルを示している．抽象階層が１とは，最も高い緊急値を有する属性のみを考慮して形成される探索空間となる．

表 3.2 探索空間の大きさ（本手法）

ケース No.	1	2	3	4
サービス合成の結果	8	11	14	23
抽象階層				
1	29	24	50	122
2	15	17	31	49
3	0	0	0	0
4	0	0	0	0
5	0	1	1	0
6	0	0	0	0
7	0	0	0	0
計	44	42	82	171

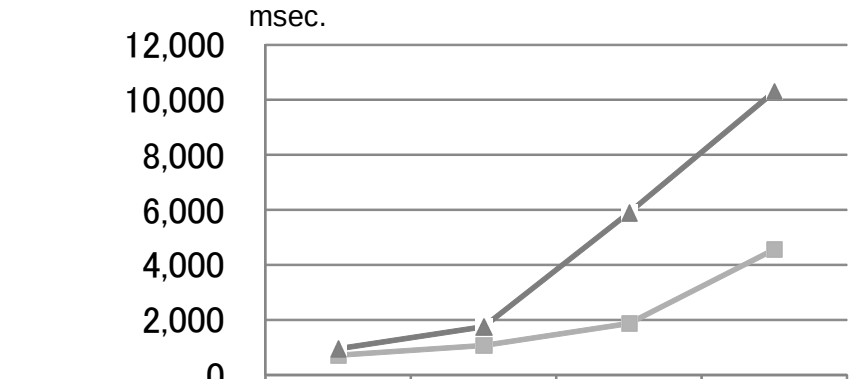
表 3.3 探索空間の大きさ（Sacerdoti の手法）

ケース No.	1	2	3	4
サービス合成の結果	8	11	14	23
抽象階層				
1	4	3	29	39
2	14	26	79	40
3	0	1	1	0
4	19	29	78	75
計	37	58	187	154

図 3.9 は，プランニングに要した計算時間（CPU time）を比較したものである．Web サービス合成に要した計算時間はいずれのケースでも Sacerdoti の手法に比べて短かった．

この時間差は探索空間の大きさとも関連するわけだが，表 2 と表 3 に示した探索空間の大きさを比較した場合，ケース 3 以外は合計値は本手法と Sacerdoti の方法ではそれほど大きな違いはない．ケース 4 ではむしろ本手法の方が大きい．しかしながら，下位の抽象階層での探索空間に違いがある．Sacerdoti の方法は，下位の抽象階層では本手法よりもサイズが大きい．すなわち，上位階層で得られた抽象的なサービス系列が下位階層ではガイドラインとして有効に働かなかったといえる．一方，本手法では上位階層での探索でほとんどが完結しており，効率的な探索が行われていることが示された．

なお，今回の実験で用いた個々の Web サービスは実際のオンラインショッピングサイトの構築を想定して作成したものである．４つのケースで得られた結果は，合成の



	1(8)	2(11)	3(14)	4(23)
本手法	716	1,066	1,867	4,564
Sacerdotiの方法	950	1,750	5,900	10,300

図 3.9 計算時間 (CPU Time, 単位:msec)

規模としては実用レベルと考えている．ただし，例外処理や性能，さらには信頼性といった本格的なシステム運用は想定したものではないため，合成結果をそのまま利用するという意味では実用的であるとは言えない．これは今後の課題としたい．

3.5　関連研究

ネットワーク上のさまざまな場所で公開されている Web サービスを合成して，新たなサービスを作成しようとする動きは，最近のサービス指向コンピューティングやマッシュアップといった動きに代表されるように広く浸透しつつある [3, 18]．こうしたサービス指向コンピューティングの実現のためには，BPEL4WS などのプロセス記述言語を用いてどのサービスをどの順序で呼び出すかを開発者が直接的に定義するアプローチと，サービス記述をもとにある程度自動的に推論し，それを開発者が修正するという協調的アプローチの 2 つが存在する．前者はワークフローをベースとしたサービス合成のアプローチであり，設計者は言語規則にのっとってサービスの組合せ方を定義すると，その言語規則にしたがって，必要なサービスを実行時に動的に決定する．また，ワークフローをベースとしていることから，条件分岐やループなどの Web サービスの実行順序を細かく指定することができる．たとえば，EFlow では，サービスをどの順序で実行するかをグラフ形式で定義し，このグラフを解釈するエンジンが動的にサービスをバインディングすることができる．しかし，このアプローチではサービスの実行順序そのものは人間が決めなければならない．

開発者との協調的なアプローチの例としては，新たなサービス記述仕様（OWL-S）や同様の記述がよく利用されている．OWL-S は，セマンティック Web 関連規格のひとつである OWL 言語 によって定義された Web サービス用のメタデータである．WSDL

では定義することができないサービスの IOPEs(Input, Output, Precondition, Effect) や非機能要件を記述する枠組みを提供しており，サービスの発見，合成，検証に関連する方法を提供している．しかしながら，OWL-S は多数の XML 標準を利用しており，かなり複雑な構成となっている．

Seog-Chan らは，複数の AI プランニング手法を組み合わせたサービス合成の方法を提案している [65]．しかしながら，適用可能性を示したのみで，実際に検証や評価までには至っていない．この他にも，SWORD[60], 小出ら [35] が AI プランニング技術を利用した方法を提案しているが，具体的な評価には至っていない．

また，古典的なプランニングと Web サービス合成にはいくつかの相違点があることが指摘されている．Peer[58, 59] は，こうした Web サービス合成の問題を AI プランニングの観点から整理している．前述のように，古典的プランニングでは，対象世界の記述は完全であることを前提としている．しかし，Web サービスの実行時にはじめて明らかになる情報も含めて計画時とは異なる情報は多々ある（たとえば，オンラインショッピングの場合，在庫量や価格は常に動いている）．また，計画時には存在していたサービスが何かしらの原因で実行時に機能していない場合にどう対応するかという状況も考えられる．

計画時と実行時のギャップという課題に対しては，Liu らは，Semantic Graph Transformation という手法を利用したオントロジーベースのプランニングシステムを提案している [39]．このシステムによって，Web サービス合成時とその合成結果を実行する時のギャップを最小化し，性能も含めた評価が行われている．

ただし，実用面という観点でみた場合，実行時に存在ないしは実行条件が変わるサービスを利用することは稀であり，個々のサービスは安定運用されており，しかも継続的に提供されていることが多い．このため，古典的なプランニングであってもその適用には意義があると思われる．

3.6 まとめ

階層プランニング技術を応用した Web サービス合成の方法について述べた．本手法は，設計者が与える外部仕様を満たす Web サービス系列を効率良く合成することを目指したものである．階層化に必要な緊急値を Web サービス仕様から自動生成し，複雑な Web サービス合成問題を効率よく解くことができ，実験によってその有効性を確認した．

本手法は，実行結果あるいは出力データのタイプが類似しているサービスが多い分野には有効な方法と思われる．また，数個のサービスを組み合わせる問題よりも，多数のサービスを合成する問題でその効果が発揮される．

現在のプランニングシステムでは，Web サービスを合成するために，インスタンスレベルでの初期条件および目標を提供する必要があるが，実際にどのような値を持つかは，実行時でないと不明なケースが多い．そこで，状態変数や入出力データの意味

を考慮したサービス合成を行うことで，サービス実行時を考慮した柔軟な結果を得ることも可能である．また，本手法では，必ずしも最適な解を得られる保証はない．場合によっては，無駄なサービスを含む系列を合成する場合もある．これらに関しては今後の課題としたい．

第 4 章

階層プランニングによる自律システムの構築

本章では，階層プランニングの技術を活用した自律的なソフトウェアシステムの実現に向けたアーキテクチャの構築について論じる．システム障害などの状況変化に対して，迅速かつ適切な対処をシステム自身が行うことを目指し，Kramer らが提案する 3 層アーキテクチャに基づき，特に重要と思われるゴール管理層の実現方法を述べる．

4.1 研究の背景

情報システムの大規模化，複雑化が進む中，情報システムには高い信頼性が要求されると同時に，それらを運用する運用管理者にも様々なシステム障害に対応できる高い専門性やシステム構成を柔軟に変更する高度な技術が要求されている．

システムの構成要素であるハードウェア，ネットワーク，オペレーティングシステム，ミドルウェアなど，それぞれは先進的な技術の集合体であることから，これらを扱うには専門的な技術が必要とされる．このため，システム構築や運用管理作業が限られた担当者に集中してしまい，それが作業上のボトルネックになっている。特に，情報システムの運用管理作業は手順化されたものもあるが，事前に想定できない状況もあり，作業がそもそも定義できない場合や，また同じ目的で複数の手段が存在する場合がある．こうしたケースでは，運用管理者の判断ミスに起因する二次障害を誘発することも珍しくない．

さらに，アクセス数の急激な増大や事前に顕在化していないバグなど，事前に予測不可能なケースに遭遇した場合，現状では運用管理者のスキルに任せてしまうことがほとんどである．こうした背景から，システムが自律的にこうした状況に適応できるようにすべきという要望が高まっている [30, 36, 50, 51, 56, 68]．

図 4.1 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．

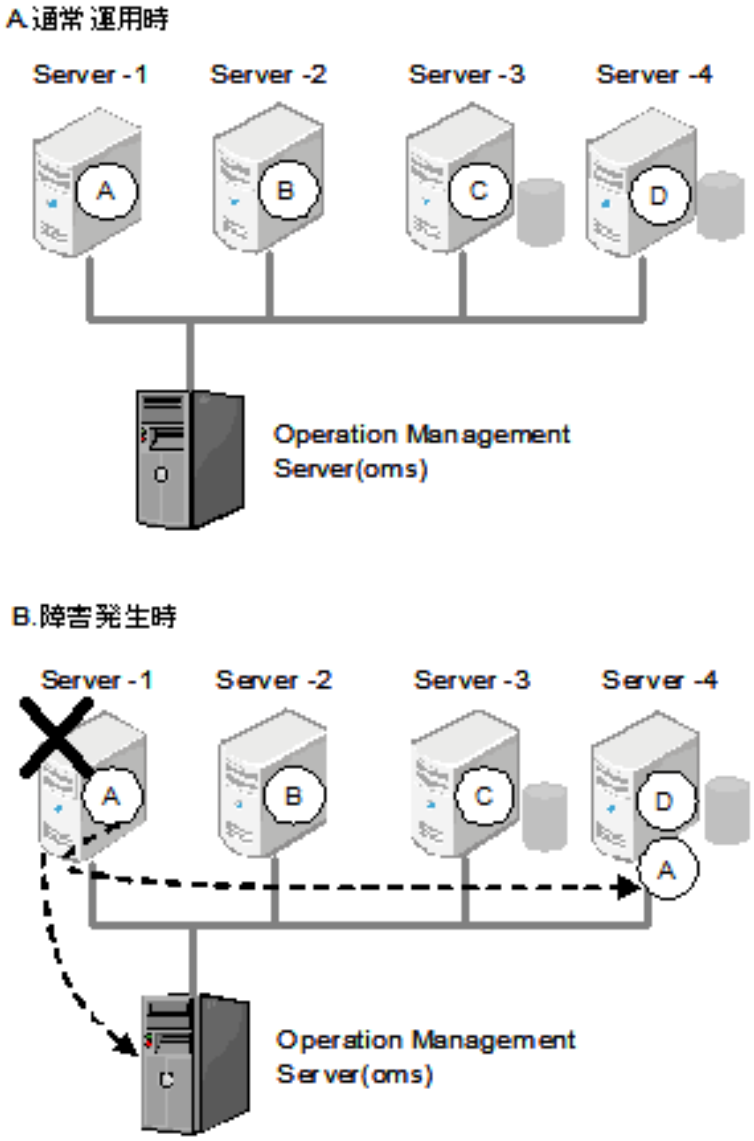


図 4.1 自律コンピューティングの例

また，障害対応以外でも，運用中のシステム上に新たにサーバを増強したり，一部のソフトウェアをバージョンアップするといったシステム構成の変更はたびたび行われる．

この例は小さなものであるが，大規模で複雑なシステムでは，こうした変化に対して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．

オートノミックコンピューティングは，2001 年の National Academy of Engineers 会議の基調講演で IBM の Paul Horn による「Autonomic Computing Manifesto」という講演で一躍脚光を浴びた考え方である．システム運用を 監視，発見，計画，実行の 4 つのサブプロセスからなる Closed Loop システムとしてとらえ，これらのサブプロセスを順次行うことによってシステムを自律的に管理するアプローチである．オートノミックコンピューティングでは，この制御を Monitor-Analyze-Plan-Execute という要素から構成される MAPE-K ループと呼んでいる．

しかし，システムが問題を把握し，分析し，問題を解決する能力を持つまでには至っていない。自律性を実現にするためには，システム運用で発生する種々な問題を蓄積し，専門家であるシステム運用管理者がどのような問題解決を行っているかを明らかにし，ルール化する必要がある．すなわち，解決すべき問題の定義と解決のためのアクションを明らかにしなければならない．

Kramer らはこうした自律性を備えるシステムを具体的な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 [36]．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 4.2)．

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 4.1 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アクション系列 (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．

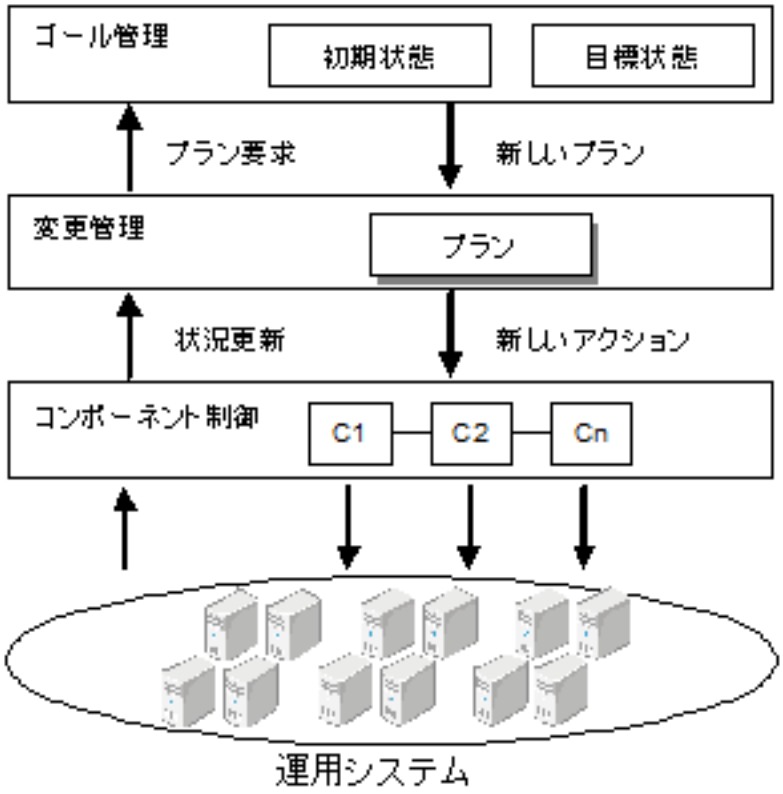


図 4.2 3 層アーキテクチャモデル

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 [70]．ゴール管理層では下位 (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．

4.2　本章の構成

以上の背景から，本章では，図 4.2 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，以下の課題を解決する方法を提案する．本章で論じる研究のポイントは次の通りである．

研究課題

大規模で複雑なシステムにおける運用管理業務軽減を目指し，状況変化に対して適切に対応する自律システム実現に向けて，特にシステム運用の戦略立案を行う

ゴール管理層を実現する手段を確立する．ゴール管理層では，迅速な戦略立案と不確実な状況にも対応できる能力が求められる．

課題を解決するための手段

システムの運用手順に関する知識をアクションとして定義することによって，階層プランニング技術を用いた効率の良い戦略立案の方法を提供する

評価結果

状況の変化に対する運用手順の戦略立案を迅速に行えることを実証した. さらに, プランニングに必要な情報の優先度を予め計算することができるので．無駄な情報収集活動を抑えることができた.

以下，4.3 節では，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．4.4 節では，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によるゴール管理層の実現方法を紹介する．4.5 節では，本手法を用いた実験とその結果に関して考察し，4.6 節にてまとめとする．

4.3 関連研究

情報システムの運用において，システムに障害が発生してから一定時間内には何らかの手段を講じる即応性は重要な要件である．状況によっては多様な選択肢があるが，ここでは，完全で最適なプランを作成する深い推論よりも，限られた時間の中で，状況をよりよい方向へと改善するための手順を得ることが重要である．しかし，即応性が求められるからと言って，近視眼的に観測された状態に対応するのではなく，ある程度の熟考を踏まえた上での戦略的な対応が必要である．

自律システムの代表例として「オートノミックコンピューティング」がある [50]．オートノミック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．

オートノミック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，さらにその上にオートノミック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さまざまな情報を収集し，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の一つとして，ポリシー管理技術がある．ポリシー管理とは，条件-動作型の規則（ポリシー）を予め定義しておき，システムの状況によってポリシーを選択し，適用することでシステムを管理する方法である．ポリシー管理は，PONDER [15] に代表される様々なポリシー記述言語やポリシーの配布方式を中心に，1990 年代から活発な研究がなされている．特に，ネットワーク管理の分野で適用が進み，ポリシーサーバなどの製品開発が行われてきた．例えば、「Web サーバのレスポンス時間が 2 秒を超え、かつ他に利用可能なコンピュータがある時は，新たに Web サーバを起動する」といったポリシーを定義することができる．しかしながら，ポリシーの数が増えていった場合に，適用可能なポリシーの競合をどのように解消するかなどの新たな問題も発生している．

一方，近年，AI プランニング技術をオートノミック・マネージャの要素技術として利用する試みがあり，活発な議論が展開されている．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 [21]，宇宙船の制御や自律ロボット [48]，電力系統システムの障害復旧の立案 [9, 73]，Web サービスの自動合成 [11, 35, 59, 65] 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 [11, 65].

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 [40, 68]．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 [59]．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さまざまな機器のみならず，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 4.5 節で説明する．

4.4 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．

そこで，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．

4.4.1 運用時のアクションの表現

運用対象システムに対する各種のアクション（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，図 4.3 次のように「opname」,「precondition」，「delete」,「add」から構成される．

```
opname::= アクション名
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]
```

図 4.3 アクションの定義

これは，STRIPS などの AI プランニングシステムで採用された定義方法である [21].「opname」 にはアクション名を定義する.「precondition」リストには，アクションの実行に必要な入力データやアクションを実行するための前提条件を定義する.「delete」リストはアクション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) およびアクション実行後の状態で成立しない条件を定義する.「add」リストはアクションの実行結果として新たに得られる出力データおよびアクション実行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．

```
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)].
```

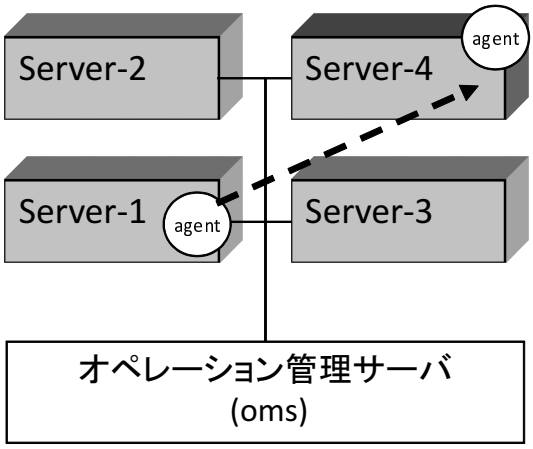


図 4.4 アクション定義の例（監視エージェントのサーバ間移動）

図 4.4 にアクション定義を例示する.「監視エージェントのサーバ間移動 (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，アクション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．このアクションが実行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定義 (migrate_sw)」である（図 4.5）．対象システムに障害が発生した際に，バックアップ系に切り替えるために定義されたアクションである．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，アクション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 4.1 は，図 4.1 の問題においてアクション定義の各リストに表れる属性の例である．

4.4.2 ゴール管理層の構成

図 4.6 は，階層プランニングシステムを中核としたゴール管理層の構成を概念的に表したものである．階層プランニングシステムに対する入力は,「アクション定義」,「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アクション系列である．ここで，階層プランニングシステムは，後述の緊急値が決定する最上位の

```
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(agent,SRVY),
            inserver(SWX,SRVY)] ::
 add # [inserver(SWX,SRVX),
            inserver(agent,SRVX)].
```

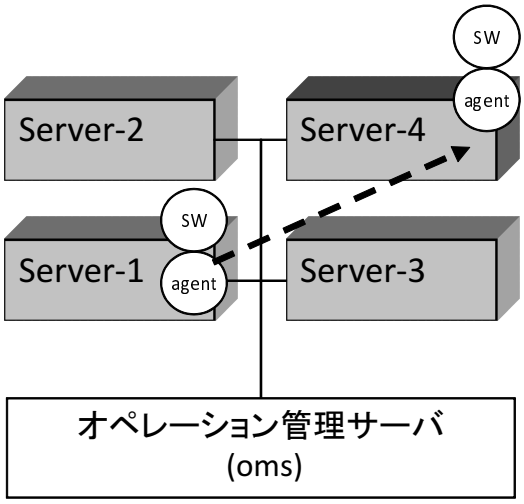


図 4.5 アクション定義の例（ソフトウェアコンポーネントのサーバ間マイグレーション）

表 4.1 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.

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アクション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，最終的に所望のアクション系列を出力する．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，別なアクション系列を探索する.

抽象階層を決定する緊急値の計算や各抽象階層でのアクション系列の推論方法は，第 2 章で述べた方法に従う．すなわち，アクション定義をもとにして，分類木を作成し，その分類木の上位に現れる属性から．高い緊急値を割り当てるという方法である.

4.4.3 初期状態と目標状態の表現

アクション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 4.1 に示す属性の集合から構成される.

たとえば，図 4.7 は，図 4.1 に示した 4 台構成のシステムで障害発生時 (B) のサーバ

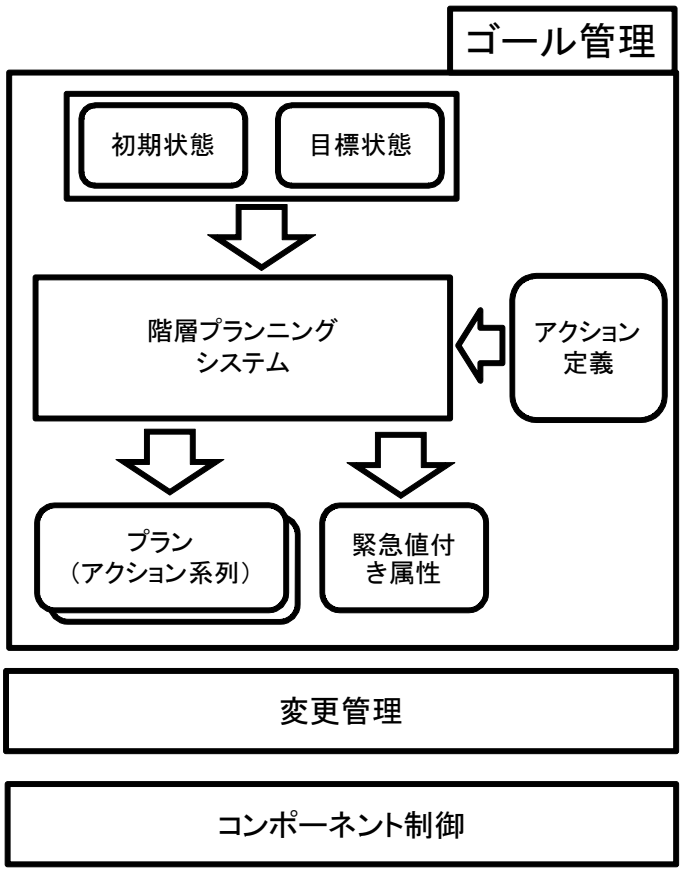


図 4.6 ゴール管理層の構成

が停止した状況 (アクション A が停止) を初期状態とし，アクション 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．

4.5 実験とその結果

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 4.1 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．

4.5.1 緊急値の割当

表 4.2 に，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，最短のアクション系列（最適解）でなく，実行可能なアクション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ニングシステムに与え，階層プランニングシステムが最初にアクション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ニングシステムが最初に生成したものは最適解であった．

4.5.2 実験結果

2 つのシナリオに関して，プランニングに要した時間および探索空間のサイズを測定した．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．

初期状態:
type(agent,agent), type(oms1,oms),
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)],
目標状態
inserver(agent,server-4),
inserver(serviceA,server-4),
process_stats(serviceA,run),

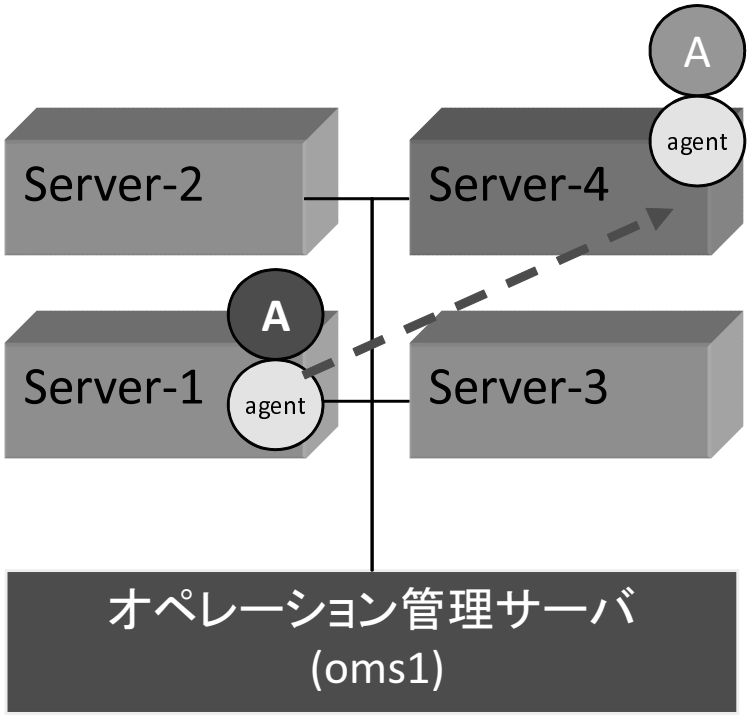
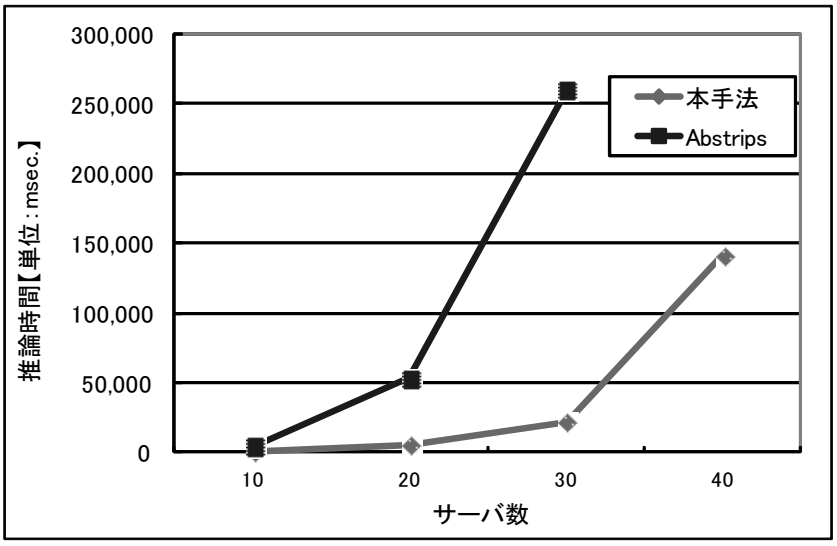


図 4.7 初期状態と目標状態の記述例

表 4.2 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



Toshiba Dynabook（Intel Core Duo プロセッサーT2300/1.66GHz ）
OS:Windows XP, Program: SWI-Prolog 5.6

図 4.8 アクション系列を得る時間 (本手法と ABSTRIPS)

表 4.3 アクション系列を得る時間 (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 4.4 (シナリオ 1) における階層別の探索空間のサイズとアクション系列の長さ (本手法)

階層	探索空間のサイズ	アクション系列の長さ
1	21,409	59
2	3,698	123
3	0	123
4	1	124
5	0	124
6	0	124
7	0	124

表 4.5 (シナリオ 1) における階層別の探索空間のサイズとアクション系列の長さ (ABSTRIPS)

階層	探索空間のサイズ	アクション系列の長さ
1	15	5
2	998	59
3	1	60
4	7,712	124

表 4.6 (シナリオ 2) における階層別の探索空間のサイズ (S) とアクション系列の長さ (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

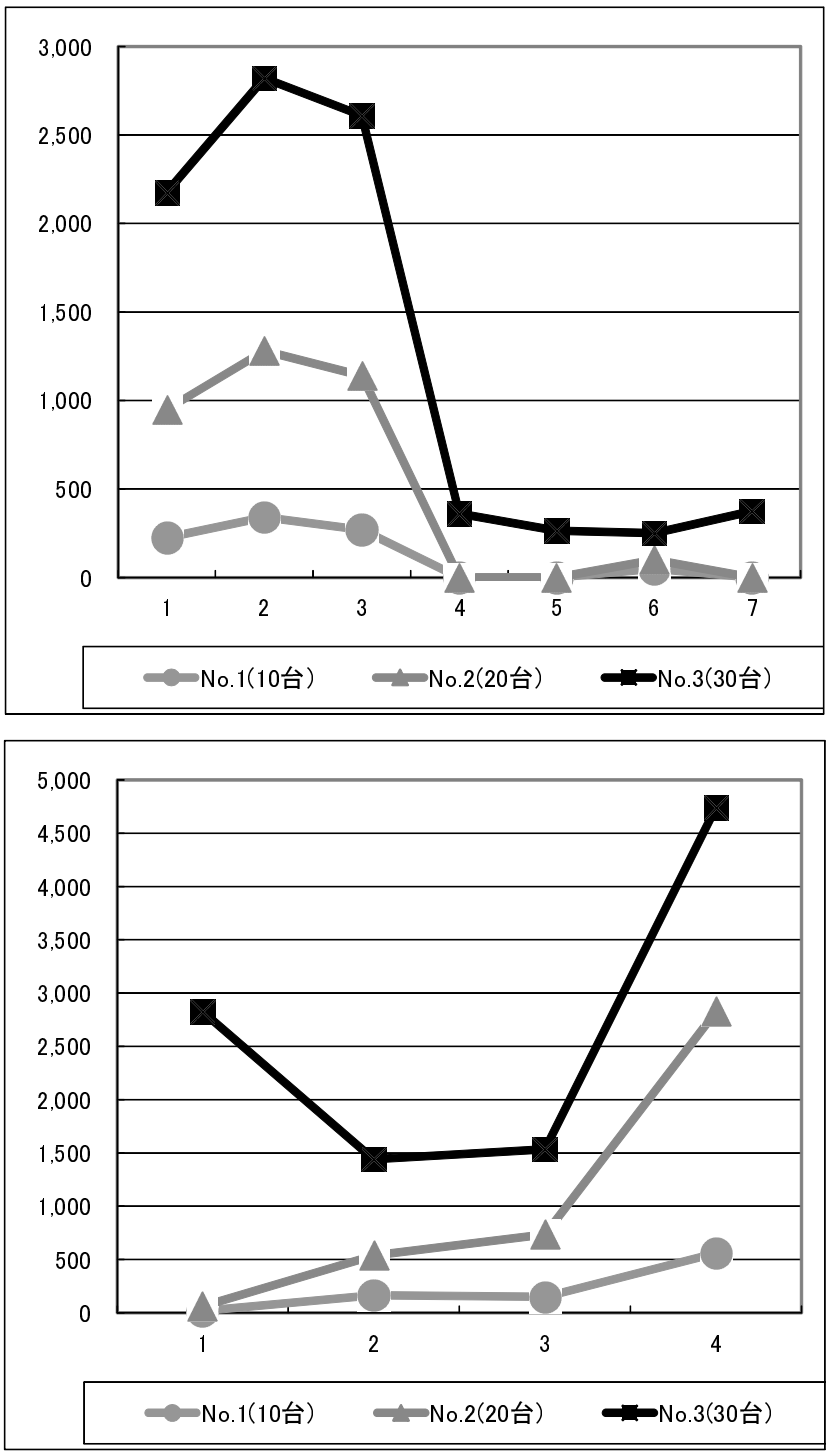


図 4.9 (シナリオ 2) における階層別の探索空間のサイズ (S)(上段：本手法．下段：AB-STRIPS)

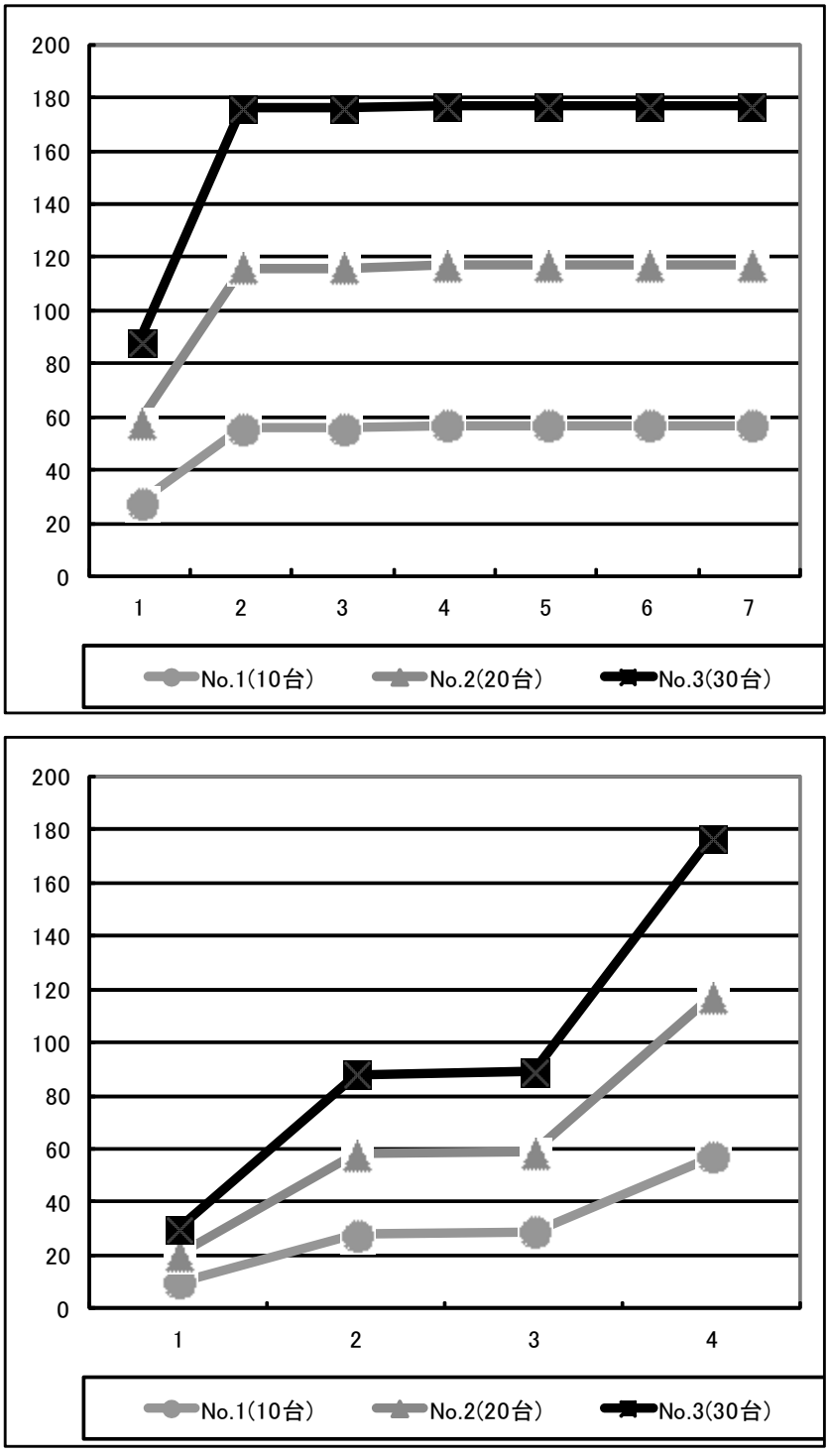


図 4.10 (シナリオ 2) における階層別のアクション系列の長さ (L)(上段：本手法．下段：ABSTRIPS)

表 4.7 (シナリオ 2) における階層別の探索空間のサイズ (S) とアクション系列の長さ (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

図 4.8 は，2 つのシナリオにおけるアクション系列の合成時間 (CPU time) を表している．(シナリオ 2)(50 台) については，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 に比べて短時間でアクション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．

図 4.9，図 4.10 は，(シナリオ 1) における探索空間のサイズ (S) とアクション系列の長さ (L) を階層別に測定した結果である．同様に，表 4.6，表 4.7 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 で得られたアクション系列の長さおよび内容は同一であり，さらには，階層プランニングシステムが最初に生成したアクション系列は最適なものであった．しかしながら，これは階層プランニングシステム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．最適なアクション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．

図 4.9 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 (図 4.9 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アクション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．

次に，各層で得られたアクション系列を分析する．表 4.7 より，ABSTRIPS では，最下位層に至るまでは最終的なアクション系列を得ることができなかった．これは，アクション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アクション系列が合成されている (表 4.6)．すなわち，緊急値 3 以下の属性を無視しても，所望のアクション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

で言うところのアクション) 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アクション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アクション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の抽象階層で得たアクション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．

4.6　まとめ

障害発生など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システム運用の自動化の方法を述べ，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アクション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アクション合成の可能性をあわせて見出すことができた．以上の点は，本研究の効果と言える．

一方，運用作業に関連するアクションをどのように定義するかはまだ十分な検討を行っておらず，今回の実験例でもアドホックな作りとならざるを得なかった．今後は，アクション定義のガイドラインを定めることも必要である．

今回は，アクション系列を迅速に得ることに主眼においたが，実際に得られるアクション系列の品質評価と高品質なものをいかにして得るかが課題である．たとえば，現在は，アクション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さまざまな条件を考慮していく必要がある．また，時間に関する情報をアクション定義の中に取り込む必要もあろう．

さらに，適応性の観点からすると，システムに障害をもたらすような事象が発生する前に予防的措置として、事前に対策を講じるという考え方も重要である．そのためにも，変更管理層やコンポーネント制御層との連携が必要である．

最後に，昨今の大規模分散システムを鑑みた場合，分散コンピューティングシステムにおいて，全体最適化を図るためのプランニング技術の開発も重要と考えている．本手法では，ゴール管理層は全体システムにおいて一つ存在するという，いわゆる集中管理という前提であるが，分散システムではそれがシステムのいたるところに分散していることもある．それぞれのゴール管理部がどのように協調して，システム全体を管理するかという点は興味深いテーマであると考えている．

第 5 章

仕様獲得プロセスの知的支援

ソフトウェアシステム仕様化作業を支援するために，その諸活動を分析するソフトウェアプロセスモデルに関する研究が行われている [5, 13, 14, 43, 44, 64]．ここでいうプロセスモデルは仕様化作業における活動，実行順序，資源などを明確に規定するものである．このプロセスモデルにより，高度に知識指向的な作業である仕様化作業を支援することができる．

本章では，システム仕様化作業を効率化するために，プロセスモデルの技術と階層プランニング技術を連携させ，支援システムの試作および評価について述べる．

5.1 研究の背景

最初にソフトウェアの仕様化作業において従来の技術が抱える問題を整理する．

ソフトウェアシステム開発の上流工程では，OOA[66]，STATEMATE[26]，UML[25] などの各種技法が使われる．これらの技法は，要求分析，概念設計など，開発フェーズで行うべき作業，作成すべき成果物，成果物が満たさなければならない制約条件を定義している．また，技法に沿って成果物を作成したり，管理するために CASE ツールと呼ばれるものがある．一般には，こうした技法やツールを利用すれば高品質なシステムを効率よく作成できると考えられているが，次の問題を抱えている．

第一の問題は，システム開発者が技法を効果的に活用するためには，高度なスキルを必要とすることである．多くの技法は，理想的な作業手順を規定しているだけで，現実的な規模と複雑さを持ったシステム開発に適用する際のガイドライン，注意事項，ドメインに関わる情報や，情報の利用方法について述べられていることは稀である．すなわち，システム開発者が技法を理解して，現実問題に効果的に適用するためには，このドメインに関する情報を整理し，蓄積することが要求される．

第二の問題は，現状のツールは，開発者が作業の過程で作成する成果物の管理を中心とし，作業のガイド (作業手順の指示) や情報の再利用に関する支援が弱いことである．ツールに慣れていない，あるいはツールのバックグラウンドにある技法を十分に

理解していない開発者に対して，作業上の注意点を的確に指示することや，過去の情報を有効に利用するためのガイドを強化するべきである．

以上の 2 つの問題を解決することによって，ツールや技法に不慣れなシステム開発者でも，仕様化作業を容易に行なえると考えている．

実際の仕様設計者の活動を支援するには，まず仕様化の工程でどのような作業が行われているかを明確に定義することが必要である．前述のように既存の仕様化技法は標準的な作業を規定するだけで，実際の設計作業の要素（作業の前提条件と結果，成果物に関する情報，作業を支援するツールなど）と要素間の関係を明確にし，その上で自動化も含めてどのような支援が可能なのかが検討することが求められている．

これらの問題に対する一つのアプローチとして，ソフトウェアプロセスモデルの研究がある．この研究の目的は，ソフトウェアの開発プロセス自身を詳細に定義し，その再利用を行うことである．多くの方法論や技法，ツールが利用されているが，それぞれは問題のタイプや大きさ，利用者の経験やスキルの程度によって向いているもの，そうでないものがある．こうした判断は，かなりの経験と熟練を必要とする．なぜならば，初心者にとって拠り所となる既存の開発方法論が成功した事例を中心とした一般的な説明に終っているからである．たとえば，システムの仕様化のレベルでは，あいまいなユーザ要求や開発者の経験不足のために，その作業の過程で多くの後戻りが発生する．しかしながら，多くの既存の方法論は，こうした問題に対して十分な解を与えてはくれない．従来，ソフトウェアの開発において再利用の対象となったものは，プロダクトそのものであったが，より品質の高いプロダクトを作成するためには，その開発プロセスも含めた再利用が必要である．システムの改定や追加要求の実現といった場合，既存のシステムを理解するために開発者は数多くのドキュメント類を読むことになる．これらの成果物間の参照関係，作成順序といった情報は，システムを理解する上で重要な情報である．

5.2　本章の構成

本章では，ソフトウェアシステムの仕様化に関するドメイン情報を整理するために，ビューポイントという概念を導入する [53]．また，ドメイン情報を効果的に利用するための作業手順立案のために，プランニング技術を応用する [52]．さらに，ソフトウェアの仕様書作成を支援するシステムを試作し，評価結果について述べる．

本章で論じる研究のポイントは次の通りである．

研究課題

仕様化作業において，プロセスに関する知識を蓄積し，作業状況に合わせて提示する方法を確立する．また，文書作成に関する作業の負荷軽減を目指す．

課題を解決するための手段

ソフトウェアプロセスモデリングの技術を用いて仕様化作業プロセスをタスクレベルで形式的に定義する．階層プランニング技術を用いて，与えられた目標に対するタスクおよびその実行順序を立案し，必要な情報の提供，ツールの起動などを制御する．さらに，図形から半自動的に説明文を作成することによって，文書作成時間の短縮を図る．

評価結果

仕様書作成支援ツールを試作し，異なるスキルを有する開発者に試用実験を行った．ビューポイントを切り替える順序（プラン）にしたがって，開発者に対して適切な情報やツールを提供することが可能となった．加えて，図形から半自動的に説明文を作成することによって，仕様の検証手段としても利用可能となることが明らかになった．

試作ツールは，開発者がソフトウェアシステムの設計を複数の視点 (ビューポイント) から行なうことを可能とする．対象を複数の視点から見ることによって，仕様化作業にありがちな，曖昧性，不完全性，冗長性を減らすことできる [22]．ここでは，ソフトウェアシステムを表現する標準的な図形モデル (たとえば，データフロー図，状態遷移図) をビューポイントとする．

さらに，試作ツールは，ビューポイントを切り替える標準的な設計作業プロセスを知識として持っている．ここで,「標準的な」という意味は，図形モデルの背景にある方法論より得られた知識であることを表している．この知識を利用することによって，開発者は自身の作業目的に対して適切な作業手順と，各作業を行なうためのツールを得ることができる．

5.3 節では，仕様化プロセスとは何かについて論じ，5.4 節ではプロセスを定式化する枠組みを説明する．5.5 節では，仕様化作業を支援するツールの構成や機能を詳しく述べる．5.6 節では，今回のツール適用実験の詳細を述べ，実験結果と得られた知見に述べる．5.7 節で，他の研究との比較を行い，5.8 節にてまとめとする．

5.3 仕様化作業プロセスのモデル化

技法とその支援ツールの問題について述べたが，これらの問題は仕様化における作業プロセスに深く関連している．仕様化作業をどのように定義するかはいろいろあるが，ここでは，仕様獲得 (要求を顕在化させる) とドキュメンテーション (要求を形式化する) という 2 つのプロセスから構成されるものと考える．

(1) ユーザ要求を獲得するプロセス（仕様獲得）

仕様獲得とはユーザの要求を早期に顕在化し，システムイメージを作成するプロセスである．このプロセスは，ユーザ要求を理解し，抽象化したり，詳細化をする作業からなる．このプロセスを規定する多くの開発技法が提案されている．

しかしながら，これらの技法は要求をどのように捉えるかという点について，一般的な指針を与えているが，実際には作業者の経験的知識に委ねられている．また，技法を活用する上でかなりの専門的な知識を必要とする．そこで，こうした技法をより使いやすくするために，仕様化作業における経験的知識を計算機上に表現し，活用することが望まれている [57]．有能な開発者は対象システムのモデル（ドメインモデル）に基づいた事例をユーザに提示することによってユーザ要求を獲得し，ドメインの基準によってユーザ要求を検証する．そして，これらの作業を効率良く進めるためには，状況に応じてさまざまな問題解決プロセスがあるという報告がなされている [20]．

(2) 要求を文書化するプロセス（ドキュメンテーション)

仕様獲得の結果を，文書としてまとめるプロセスである．多くの仕様書は，システムの特徴や開発目的に応じて文書の構成が決まっていることが多い．再利用可能な部分を知識として表現し，それを組み込んだ文書作成支援を行なう提案もなされている．たとえば，Garg らはドキュメントの論理構成に基づいたインテリジェントエディタを開発している [23]．文書の論理的な構成に従うことで開発者は容易に文書を作成することができる．しかしながら，文書の構成に関する情報のみであり，そこにはどの部分をどのような順序で記述すれば良いかという知識は持っていない．この記述順序に関する知識は上述の仕様獲得プロセスと密接に関連している．

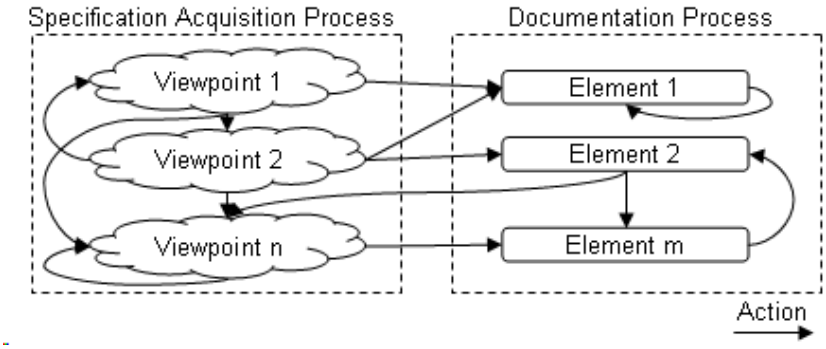


図 5.1 システムの仕様化プロセス

これらの 2 つの作業は必ずしも独立したプロセスではなく，相互に深い関連を持っている（図 5.1）．この相互関係の複雑さが，経験の浅い開発者にとって，仕様化作業の理解を難しくさせている原因にもなっている．

また，仕様化プロセスはある種の意思決定のプロセスでもあり，経験的知識は最終的な成果物の完成度にも大きく影響する [17]．

そこで，こうした仕様化作業の経験が浅い開発者を支援することを目指し，特に，仕様獲得における経験者の持つノウハウの提示，文書作成時間の短縮，文書の品質の向上という課題を解決する方法を提案する．

- 仕様獲得の基本的な作業行為は何か：プロセスのモデル化
- 設計作業の状況に合わせた作業手順をどのように推論し，提示するか：経験的知識の蓄積と利用
- 文書作成作業を如何にして減らすか：自動化

これらの課題を解決するために，本章では仕様獲得の作業プロセスをビューポイントとプランという概念により定義する．仕様獲得や文書作成作業の状況は状態ベースで表され，作業目的を与えることにより作業手順を推論する．そして，文書作成作業を減らすために，仕様獲得で得られた情報からを説明する文生成について述べる．

5.4 プロセスモデル

仕様獲得の作業プロセスをモデル化し，ドメイン知識の整理および表現を行なうための基本的な考え方について述べる．最初に，プロセスを特徴づける概念としてビューポイントについて述べる．次に，ドメイン知識を計算機上に表現するための技術としてプランニング技術について述べる．

5.4.1 ビューポイント

多くのシステム開発では，複数の観点から対象システムを分析し，定義する．ここで，それぞれの観点から得られるシステムの側面（モデル）を明らかにし，統合していくかが大きな問題となる．Finkelstein はこの問題を Multiple perspectives problem と呼んでおり，その解決策として，個々の開発者の目的，手順，表現方法などの作業者の知識をビューポイントとして定義し，ビューポイント間の変換方法を提案している [22]．システム仕様とはビューポイントの集合であり，そこには，単にシステムのみならず，開発の過程までも見ることができる．すなわち，分析や設計という作業は，ツール上でビューポイントを操作することであり，あらかじめ，ビューポイントを方法論に従って設定しておけば，適切なガイダンス機能も提供することができるという思想である．

では，このビューポイントとして何が存在するのであろうか．

たとえば，OOA では対象の物理的な構成要素を表すオブジェクトモデル，オブジェクトの機能的な面を表すプロセスモデル，オブジェクトのライフサイクルを表す状態モデルを定義している．これらはビューポイントの重要な要素となりうる．

ある問題に直面したとき，その問題を理解するために，我々は 4W1H という概念を利用している．この 5 つの概念は理解の目的を示すとともに，問題理解のための知識のタイプを規定している．たとえば，システムの構成要素 (Who あるいは Where) を明らかにするために，そのシステムの機能 (What) や動き方 (How や When) を利用する．ただし，その利用の仕方は問題や作業の状況によって異なる．

本章では，これらの 5 つ概念をベースにして 3 つのビューポイントを設定する．

(1) 構成ビューポイント

システムの構成要素と要素間の関係といった物理的な構造に注目するビューポイントである．ソフトウェアシステムの場合，システムの構成要素としてはデータおよびプロセスと考えられるが，ここでは，データの構成についてのみ考える．すなわち，システムを実体関連モデルという面から捉える．

(2) 機能ビューポイント

システムの機能をモデル化するためのビューポイントである．ここでは，データ変換を行なうプロセスとして機能を考える．すなわち，システムをデータフローという面から捉える．

(3) 振舞いビューポイント

システムの時間的な面に対するビューポイントである．ここでは，状態とイベントを考える．すなわち，システムの振舞いを状態遷移として捉える．

本研究でのビューポイントは Finkelstein の定義とは若干異なり，Statemate[26] における 3 つのチャートと同様である．

5.4.2　プランニング

システム仕様化における知識の整理が可能になると，次はその知識を効果的に適用する手順を明らかにしなければならない．仕様化プロセスは前述の幾つかのビューポイントの流れや依存関係を制御するものである．対象システムの特徴や作業目的の違いによって，適切なプロセスが存在する．この仕様化プロセスを導くことは，人工知能技術の一つであるプランニング技術と密接な関係がある [7, 17, 29, 44]．

AI プランニングは，目標を達成する行動計画の作成ばかりでなく，さまざまな利用価値がある．たとえば，学習者の問題解決のトレース（プラン）をベースにした数式処理の教育システムを構築し，プランとして表現した学習者のモデルを利用することによって，学習者のメンタルな操作を認識したり，学習者の誤りを指摘したり，その誤りを修正する有効な手順を提示する t といった利用である [24]．

システム仕様化支援においても，仕様化プロセスによって，システム開発者が仕様化という行為をどのように行っているのかを認識したり，仕様化の結果が制約条件を満足しないなどの問題が発生したときに，何が問題で，その問題を解決する最適な手順が何かを得ることができる．

5.5 支援システムの概要

前節において，仕様化プロセスを支援するために，ビューポイントに基づく知識の整理とプランニング技術の応用について述べた．本節では，試作した支援システムを詳しく説明する．

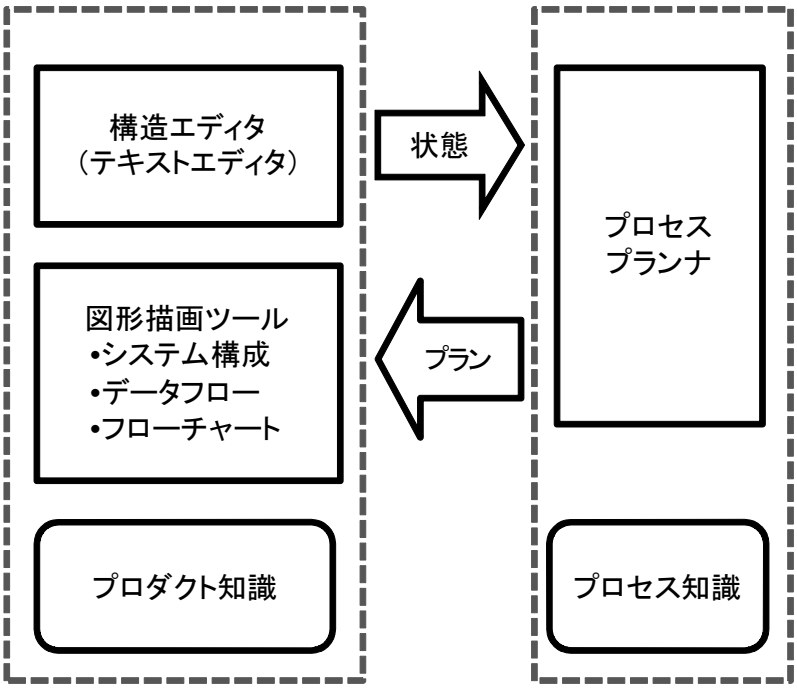


図 5.2 システム仕様化支援システムの構成要素

図 5.2 に示すように，支援システムは，プロセスプランナ，エディタ（構造エディタと図形描画エディタ），知識ベース（プロダクト知識とプロセス知識）から成る．プロセスプランナは，仕様化作業に関するアクション定義したプロセス知識を利用して，状況に応じた作業プロセスを推論する．開発者は，推論された作業プロセスを実行することによって，どんな作業をすべきなのか，どのドメイン情報を利用すべきなのかといった指示を受ける．また，作業プロセスはドキュメントの記述プロセスもコントロールする．すなわち，作業プロセスの実行によって，構造エディタはその編集領域を制限される．

5.5.1 プロセスプランナ

プロセスプランナは仕様化の進行状況に応じて，仕様化プロセスを推論する．プロセスの基本構成要素となるアクティビティはオペレータ定義として定義される．

図 5.3 は，ソフトウェアシステムの特許明細書を作成するプロセスにおけるアクティビティに関する知識の一例である．ソフトウェアシステムの特許明細書は，一般のソ

フトウェアシステムの仕様書の構成，内容ともに類似していることから，応用事例として採用した．

オペレータの表現形式は，基本的には，第 2 章で述べた階層プランニングシステムにおけるオペレータ定義に基づいているが，add リスト中の条件の中で，特に重要なものと思われるものを main_effect 部とし，また，各属性の引数間の型情報を sort 部で定義する．また，オペレータを適用する際の制約条件は constraint 部にて定義する．

さて，このアクテビィティは，発明の適用効果を実現するための技術および技術の作用を検討することを表している．発明の効果を検討しており，技術および技術作用が未検討であるという前提条件を満足すると，その結果は，発明効果の検討を終了して，技術およびその作用の検討を実施する状態にする．

```
opname # システム化を検討する ::
     precondition # [
             課題を解決する手法 (未定),
             技術の作用 (未定)， 技術の適用効果 (検討中) ] ::
     delete # [
             課題を解決する手法 (_)，技術の作用 (_)， 技術の適用効果 (_) ] ::
     add # [
             課題を解決する手法 (検討中),
             技術の作用 (検討中)，技術の適用効果 (終了) ] ::
     main_effect # [
             課題を解決する手法 (検討中)，技術の作用 (検討中)，技術の適用 (終了) ] ::
     sort # [
             type(未定,editor, 起動条件),type(検討中,editor, 作業目的),
             type(終了,editor, 作業目的) ] ::
     constraint # true .
```

図 5.3 オペレータ知識: システム化を検討する

プロセスプランナは，図 5.4 に示すようなこのオペレータ知識から作成される抽象階層を利用して，問題空間を分割し，上位空間から段階的にプランニングを行なう．その処理を簡単に述べると，各々のオペレータ知識の結果を比較し，同一あるいは類似の場合に限り，それらの共通部分からなる抽象オペレータを作成する．これをすべてのオペレータ知識に適用することによって，オペレータの分類木を得る．たとえば，システム化を検討するオペレータと， 問題解決手法を検討する オペレータから作成された抽象オペレータは図 5.4 に示した抽象オペレータ 7 となる．

次に問題空間を構成するリテラルのランクづけを行なう．これは分類木の上位にあるほど高いランクを設定する．プロセスプランナはリテラルのランクに従って，オペレータ知識を再構成する．すなわち，オリジナルなオペレータ知識の各条件部を，同じもしくはそれよりも高いランクのリテラルから構成されるようにする．プロセスプランナは問題が与えられると，その問題も抽象化し，上位空間より段階的にプランニングを行なう．

なお，プロセスプランナは第 2 章で説明した階層プランニングシステムと同じものを利用している．

```
opname # 問題の解決手法を確定する ::
          precondition # [
               システムの名称 (検討中), 適用の分野 (検討中),
               従来技術の調査 (検討中), 課題 (検討中), 課題を解決する手法 (未定),
               技術の作用 (未定), 技術の適用効果 (未定)] ::
          delete # [
               システムの名称 (A), 適用の分野 (B), 従来技術の調査 (C), 課題 (D),
               課題を解決する手法 (E), 技術の作用 (F), 技術の適用効果 (G)] ::
          add # [
               システムの名称 (終了), 適用の分野 (終了), 従来技術の調査 (終了),
               課題 (終了), 課題を解決する手法 (検討中), 技術の作用 (検討中),
               技術の適用効果 (検討中)] ::
          main_effect # [
               システムの名称 (終了), 適用の分野 (終了), 従来技術の調査 (終了),
               課題 (終了), 課題を解決する手法 (検討中), 技術の作用 (検討中),
               技術の適用効果 (検討中)] ::
          sort # [
               type(未定,editor, 起動条件),type(終了,editor, 作業目的),
               type(検討中,editor, 作業目的)] ::
          constraint # true .

opname # 抽象オペレータ 7(abvar29,abvar30,abvar31) ::
          precondition # [
               技術の作用 (未定), 課題を解決する手法 (未定)]  ::
          delete # [
               技術の適用効果 (abvar29),
               技術の作用 (abvar30), 課題を解決する手法 (abvar31)]  ::
          add # [
               技術の適用効果 (終了), 技術の作用 (検討中),
               課題を解決する手法 (検討中)]  ::
          main_effect # [
               技術の適用効果 (abvar32),
               技術の作用 (abvar33), 課題を解決する手法 (abvar33)]  ::
          sort # [type(abvar29,variable,system),type(abvar30,variable,system),
                  type(abvar31,variable,system),type(未定,editor, 起動条件),
                  type(終了,editor, 作業目的),type(検討中,editor, 作業目的)]  ::
          constraint # [].
```

図 5.4 抽象オペレータの作成

技術の作用/1 や課題を解決する手段/1 は他のリテラルよりもランクが高くなる．すなわち，プロセスプランナは技術の作用/1 や課題を解決する手段/1 を優先する．

5.5.2　エディタ

エディタは，構造エディタと図形描画ツールから構成される．構造エディタは主にテキストを編集する際に利用される．一方，図形描画エディタは，ソフトウェアシステムのビューポイントを描くために利用される．

多くの仕様書は記述前にその構成や一部内容が決まっていることが多い．従って，エディタがこの構造を内部に持つことで，編集作業の省力化が図れる．

構造エディタは，確定節文法により表現した文書構造規則 (プロダクト知識) を内部にもつエディタである．文書の構成単位を非終端・終端記号として表現し，推敲手続きを prolog 節として記述する．

また，構造エディタはプランナからの指示によって，編集可能／不可能領域を設定することができる．たとえば，図 5.5 において，「実施例 (Description of the Preferred Embodiment)」の部分を編集不可能にしておくこともできる．編集不可能領域は参照のみが可能であり，その領域内に文字の入力はできない．この処理は，構造エディタ上のメニューによって起動される．

図 5.5 は構造エディタを利用して，ソフトウェアの特許明細書を作成している画面例である．中ほどに表れているウィンドウはその項目に対するヘルプ画面である．また，枠で囲まれた部分は編集可能な部分を表している．推敲結果は枠内の反転文字列で示される．図においては否定表現を含む文が示されている．

また，構造エディタの補助機能として，図形描画から説明文を生成する機能を実現した．

システムの仕様化プロセスは「要求を形式化する仕様獲得プロセス」と「文書化を行なうドキュメンテーション」からなる．これらの作業プロセスでは，一方では図を，他方では自然言語を主に用いる．通常は，これらの作業は別々に行なわれるので，それぞれの記述内容の間の一貫性を管理しなければならない．もし，この部分が自動化できれば，非常に大きな効率化を期待できる．説明文生成機能はこの要求を満たす目的で作成された．

図面内容を説明する文章は，ドキュメントに直接利用することができることが望ましい．したがって，単純に図面上に記載されている個々の図形を説明するだけではなく，分かりやすさと一意性を考慮した質の高い文章が求められる．分かりやすさとは，難解な語彙を用いずに，どんな読み手でも理解が容易であることを意味する．また，一意性とは，文の意味が一つになることである．

また，説明文の作成は，図面の内容が設計目的と合っているかを確認するといった別な目的のためにも行なわれる [57]．ここでは，高速な文生成が望まれる．

そこで，テンプレートを用いた説明文の生成機能を実現する．図 5.6 に示すテンプレートは，システムプロセスに関するものである．[　] で囲まれた部分は，図面に現れる基本構成要素の名称に具体化される．また，基本構成要素間の関係に応じて，動詞が選択される．たとえば，プロセスの説明で，入力先がデータベースの場合は，動詞「読み込む」が選択される．

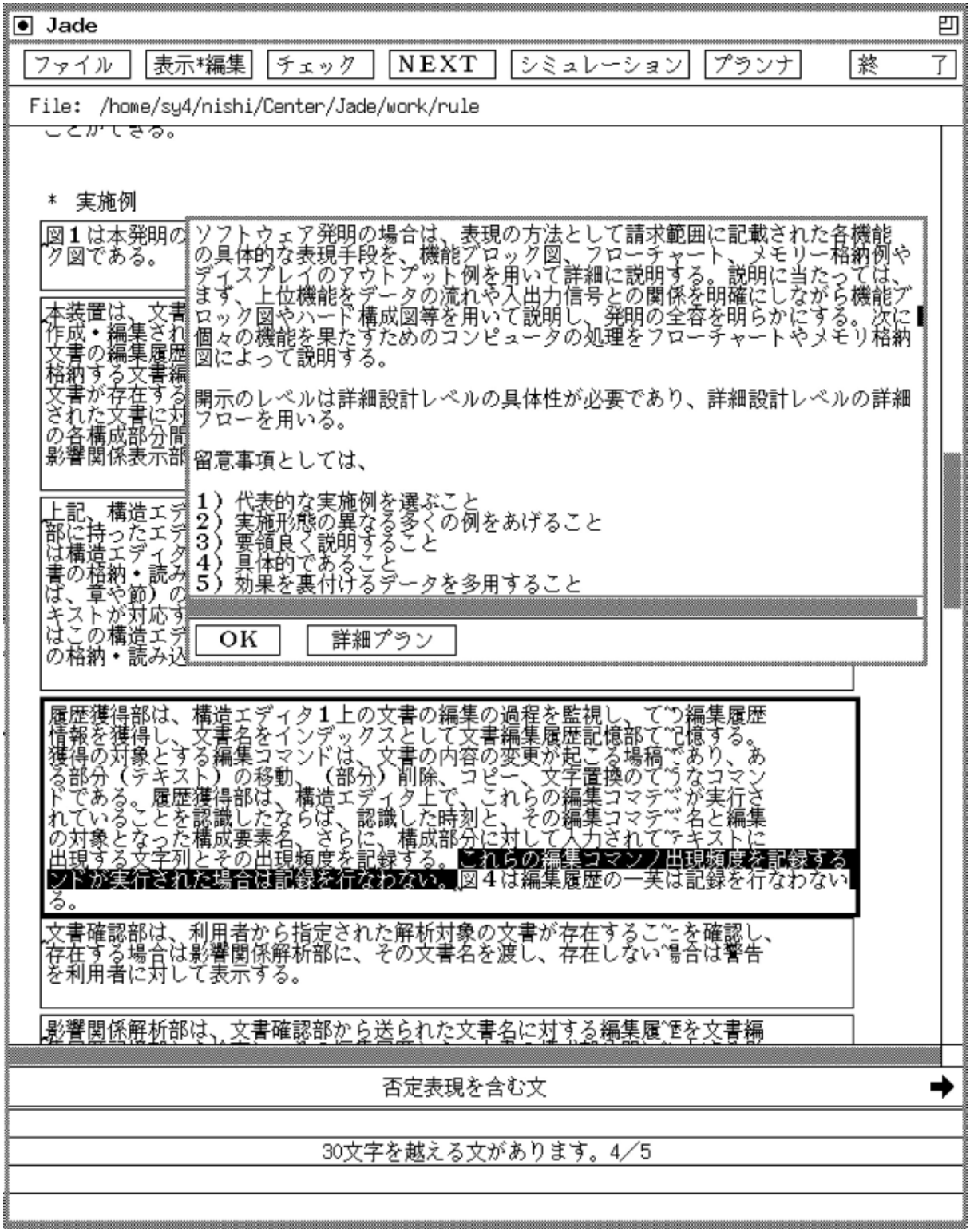

Jade
ファイル
表示*編集
チェック
NEXT
シミュレーション
プランナ
終　了
File: /home/sy4/nishi/Center/Jade/work/rule
＊ 実施例
ソフトウェア発明の場合は、表現の方法として請求範囲に記載された各機能の具体的な表現手段を、機能ブロック図、フローチャート、メモリー格納例やディスプレイのアウトプット例を用いて詳細に説明する。説明に当たっては、まず、上位機能をデータの流れや入出力信号との関係を明確にしながら機能ブロック図やハード構成図等を用いて説明し、発明の全容を明らかにする。次に個々の機能を果たすためのコンピュータの処理をフローチャートやメモリ格納図によって説明する。
開示のレベルは詳細設計レベルの具体性が必要であり、詳細設計レベルの詳細フローを用いる。
留意事項としては、
1）代表的な実施例を選ぶこと
2）実施形態の異なる多くの例をあげること
3）要領良く説明すること
4）具体的であること
5）効果を裏付けるデータを多用すること
OK
詳細プラン
履歴獲得部は、構造エディタ上の文書の編集の過程を監視し、その編集履歴情報を獲得し、文書名をインデックスとして文書編集履歴記憶部で記憶する。獲得の対象とする編集コマンドは、文書の内容の変更が起こる場合であり、ある部分（テキスト）の移動、（部分）削除、コピー、文字置換のようなコマンドである。履歴獲得部は、構造エディタ上で、これらの編集コマンドが実行されていることを認識したならば、認識した時刻と、その編集コマンド名と編集の対象となった構成要素名、さらに、構成部分に対して入力されてテキストに出現する文字列とその出現頻度を記録する。これらの編集コマンド出現頻度を記録するコマンドが実行された場合は記録を行なわない。図4は編集履歴の一実は記録を行なわないる。
文書確認部は、利用者から指定された解析対象の文書が存在することを確認し、存在する場合は影響関係解析部に、その文書名を渡し、存在しない場合は警告を利用者に対して表示する。
否定表現を含む文
30文字を越える文があります。4／5

図 5.5 構造エディタ

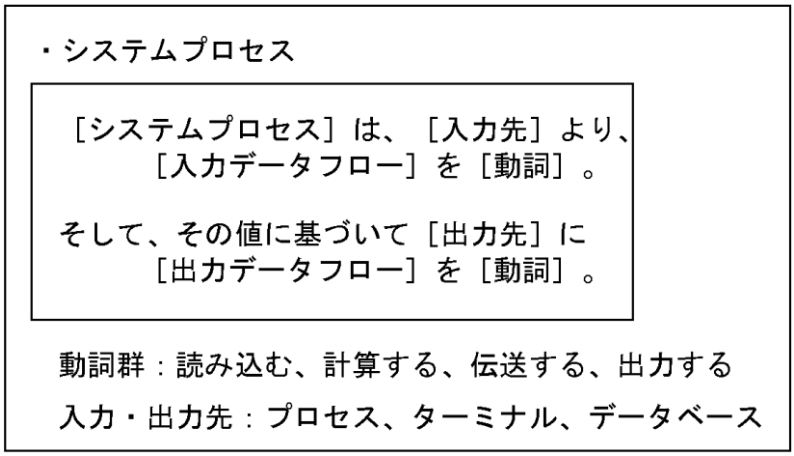


図 5.6 説明文のテンプレート

また，わかりやすい文書を作成するために，文の並びに関する経験則を用いて，説明の順序を決定する．たとえば，図面の左上から右下にかけて説明順を決定する，データの流れに基づいて説明順を決定するなどである．

5.5.3　知識ベース

本システムでは，プロセスとプロダクトに関する 2 種類の知識がある．

仕様化作業は， 要求定義や基本設計といったフェーズが存在し，その間に，情報の依存関係や時間的な関係がある．ここでは，図 5.7 の上部に示すようなプロセスを設定する．これを，マクロプロセスと呼ぶことにする．マクロプロセスは初期フェーズと終了フェーズを除いて，5 つのフェーズからなり，12 のアクティビティから構成される．アクティビティ間の状態 (作業状態) は，構造エディタ上の各項目の記述の程度を示している．ここで作業状態を未定，検討中，終了の 3 段階とする．

一方，マクロプロセスにおける各作業状態は，さらに詳細な作業から構成することができる．これをミクロプロセスと呼ぶことにする (図 5.7 下部)．本章では 詳細設計のミクロプロセスをビューポイントの推移として定義する．ビューポイント間の推移の順序に関する情報を一般的に述べたものはないことから，本システムでは，仕様書に書かれた内容の記述順序から類推することとした．説明の記述順序と設計順序が同じであるという保証はないが，仕様書に書かれた内容は再現性があり，課題や要求に対応する解決方法という流れで書かれることが多いため，記述順序に従って設計が行なわれたと考えても良いとした．ここでは，いくつかのシステム仕様書を調査し，その記載内容を分析することによってどのように記述されているかをビューポイントの移動という観点で整理した．その結果は図 5.7 の下部に示すプロセスとなった．図中 s は構成，f は機能，b は振舞いを示す．矢印はビューポイントの移動アクティビティを表す．たとえば，s_f は構成から機能へビューポイントを変更するアクティビティであ

る．ミクロプロセスは機能の注目から始まり，構成，振舞いへとビューポイントが移動する．

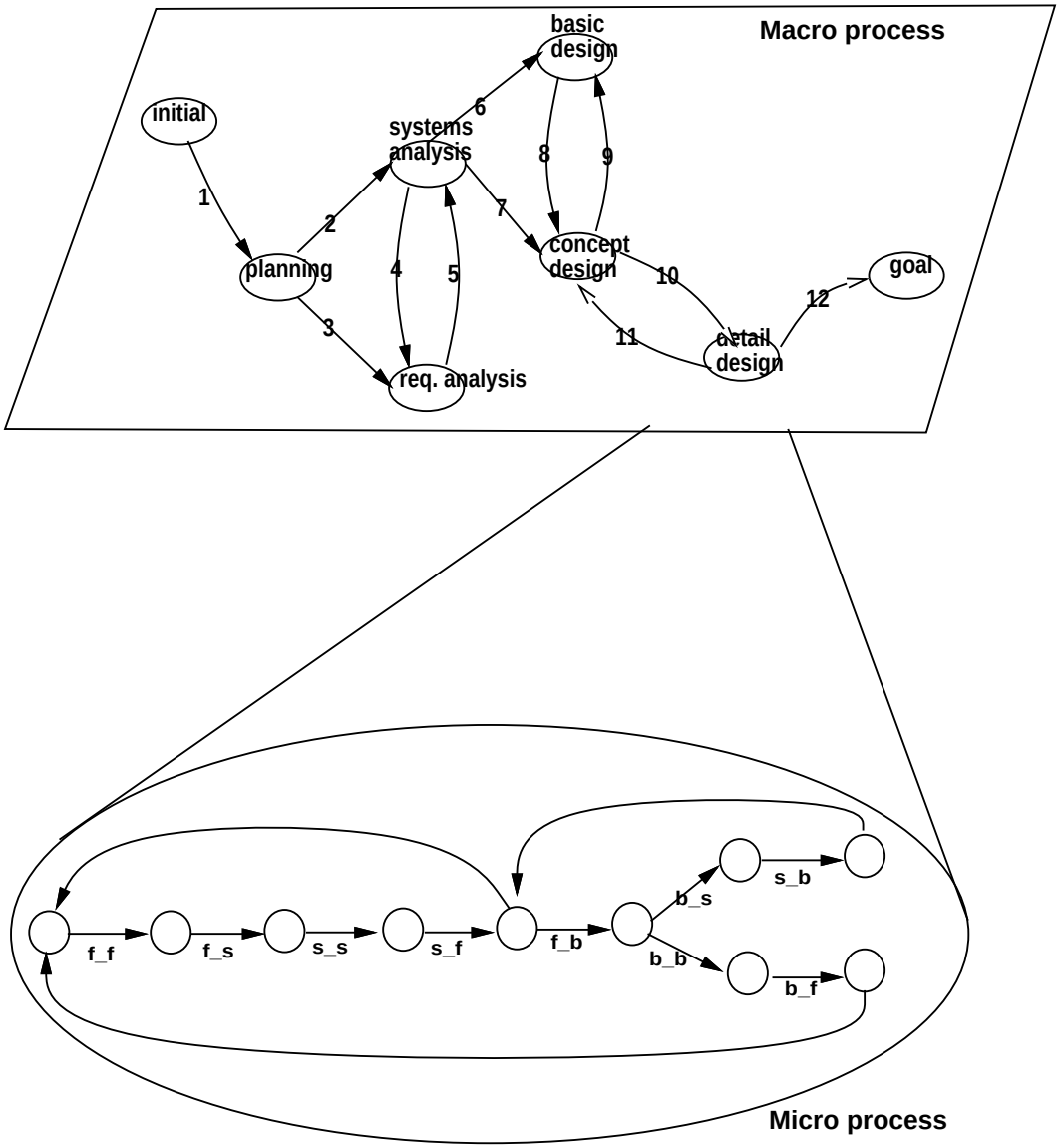


図 5.7 仕様化のマクロプロセスとミクロプロセス

プロセスの生成は初期条件と目標条件を与えることによって行なわれる．たとえば，初期条件 (現状) が「課題の検討中」で,「作業を終了したい」場合のプランは，[**5-7-10-12**]*というプロセスが生成される．図 5.8 は，この作業プロセスの作成過程を表している．階層プランニングシステムは，プロセスの知識の分析から得られた抽象階層に従って，問題を構造化し，上位階層から段階的に問題解決を行なう．本例では，4 レベルの抽象階層となった．

プロダクトに関する知識は，ドキュメントの論理的な構成を定義したものである．システムの要求仕様書のリファレンスには，たとえばIEEE Guide to Software Requirements

*番号は図 5.7 のマクロプロセスに対応する

```
Define Static Axiom !!
--抽象階層-- 4
抽象プラン　 [詳細な実施例を記述する , すべての作業を終える]
探索空間　　 2
展開した空間 2
プランの長さ 2

--抽象階層-- 3
抽象プラン　 [問題解決の手順を具体化する , 詳細な実施例を記述する , すべての作業
を終える]
探索空間　　 3
展開した空間 1
プランの長さ 3

--抽象階層-- 2
抽象プラン　 [問題の解決手法を確定する , 問題解決の手順を具体化する , 詳細な実施
例を記述する , すべての作業を終える]
探索空間　　 4
展開した空間 1
プランの長さ 4

--抽象階層-- 1
抽象プラン　 [問題の解決手法を確定する , 問題解決の手順を具体化する , 詳細な実施
例を記述する , すべての作業を終える]
探索空間　　 0
展開した空間 0
プランの長さ 4
```

(注)「抽象階層」は抽象階層のレベルを表す.「抽象プラン」はその階層で作成されたプランを表す.

「探索空間」はプランニングに要した探索ノードの数を表す.「展開した空間」は実際に，プランナが

選択したノード数を表す.「プランの長さ」は「抽象プラン」を構成するオペレータの数を表す.

図 5.8 プロセスプランナの出力

Specification[1] がある．しかし，多くのプロジェクトでは，開発対象のシステムやプロジェクトの特性で，こうした標準を参考にしつつも，独自の構成をとることが多い．また，仕様書を作成する際には，目次レベルでの構成を決定した後，個々の章や節の内容を記述するケースが多い．

本システムでは，文書の論理構成をプロダクト知識として予め定義しておき，この知識を上記の構造エディタで利用する．

5.6 実験とその結果

ソフトウェア仕様書作成ツールを実作業に適用し，評価を行う．この評価の目的は，次の 2 点を明らかにすることである．

(1) 作業プロセスを遂行する知識

仕様化作業プロセスは，開発者の持つビューポイントが状況に応じて移り変わる過程とする．ここで，ビューポイントとは，設計対象を表現するためのモデルを表す．

実験では，実際の開発者がどんなビューポイントを持ち，どういう状況で切り替えているかを分析する．また，方法論として提供されていない，開発者の経験的な知識も明らかする．

(2) 図面作成による文書作成の自動化

実験では，このテンプレート方式による文生成の結果を評価するとともに，図面作成作業に対する文生成作業の影響を分析する．

試作したツールを使って，数名の開発者に，各自の得意とする (ソフトウェアシステムに限定した) 分野のソフトウェア仕様書を作成してもらい，その作成過程を分析する．

5.6.1 実験システムの構成

図 5.9 は，実験システムの構成を示している．

ここで，Jade はドキュメント作成専用の構造エディタを，K-SCORE-E は図形エディタを表す．開発者は，Jade 上でソフトウェア仕様書の本文を記載する．そして，図面は K-SCORE-E を用いて作成する．

K-SCORE-E では，ソフトウェアシステムを表現する図形表現の基本構成要素をビューポイントとした．一般に，図形モデルをベースとしたシステム仕様化方法論では，開発者の作業手順はこの図形モデルに基づいて定義されている [16]．たとえば，データフロー図では，システムプロセス，データベース，ターミナル，データフローという基本構成要素があり，開発者は，仕様化対象をこれらの構成要素で定義しなければな

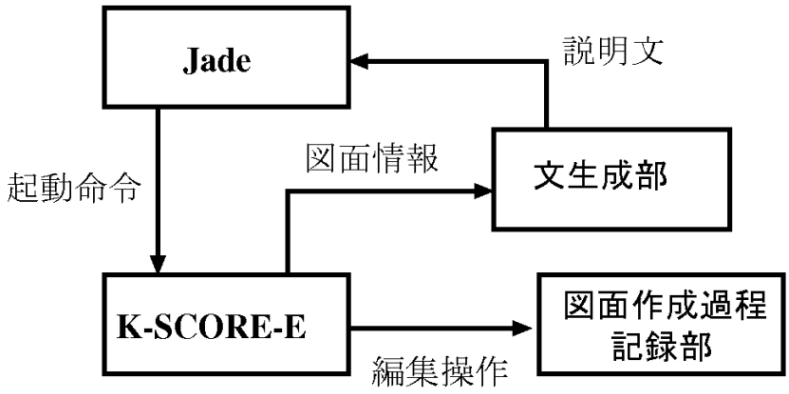


図 5.9 実験システムの構成

らない．つまり，開発者はこれらの構成要素に注目して作業を行なっていることになる．K-SCORE-E では，これらのビューポイントを部品 (図 5.10 下 K-SCORE-E エディタ内の右上) として定義した．開発者は自身の意図を，与えられる部品群から選択し，エディタ上で図面作成を行なう．

ビューポイントの移り変わりは，上述の各基本構成要素の編集操作とする．すなわち，基本構成要素の追加，削除，名前の変更である (図 5.10 下 K-SCORE-E エディタ内の右下)．

文生成部は，開発者の指示によって，図面の内容からテキスト文書を作成し，その文書を Jade へ送る．図面作成過程記録部は，開発者の図面作成の過程を記録する部分である．

図 5.10 は Jade と K-SCORE-E の画面例である．

5.6.2　被験者の特徴

本実験は，表 5.1 に示す 3 名の開発者を対象として行った．表に示すように，各開発者はシステム設計や仕様書の記載経験が異なる．なお，今回は自身の専門分野における発明の考案を行なってもらった．

表 5.1 被験者一覧

開発者名	発明の名称	設計経験	仕様書の記載経験
A	ニューラルネットワーク	7 年以上	10 回以上
B	データベース	2 年以上	4 回
C	予測システム	4 年以上	4 回

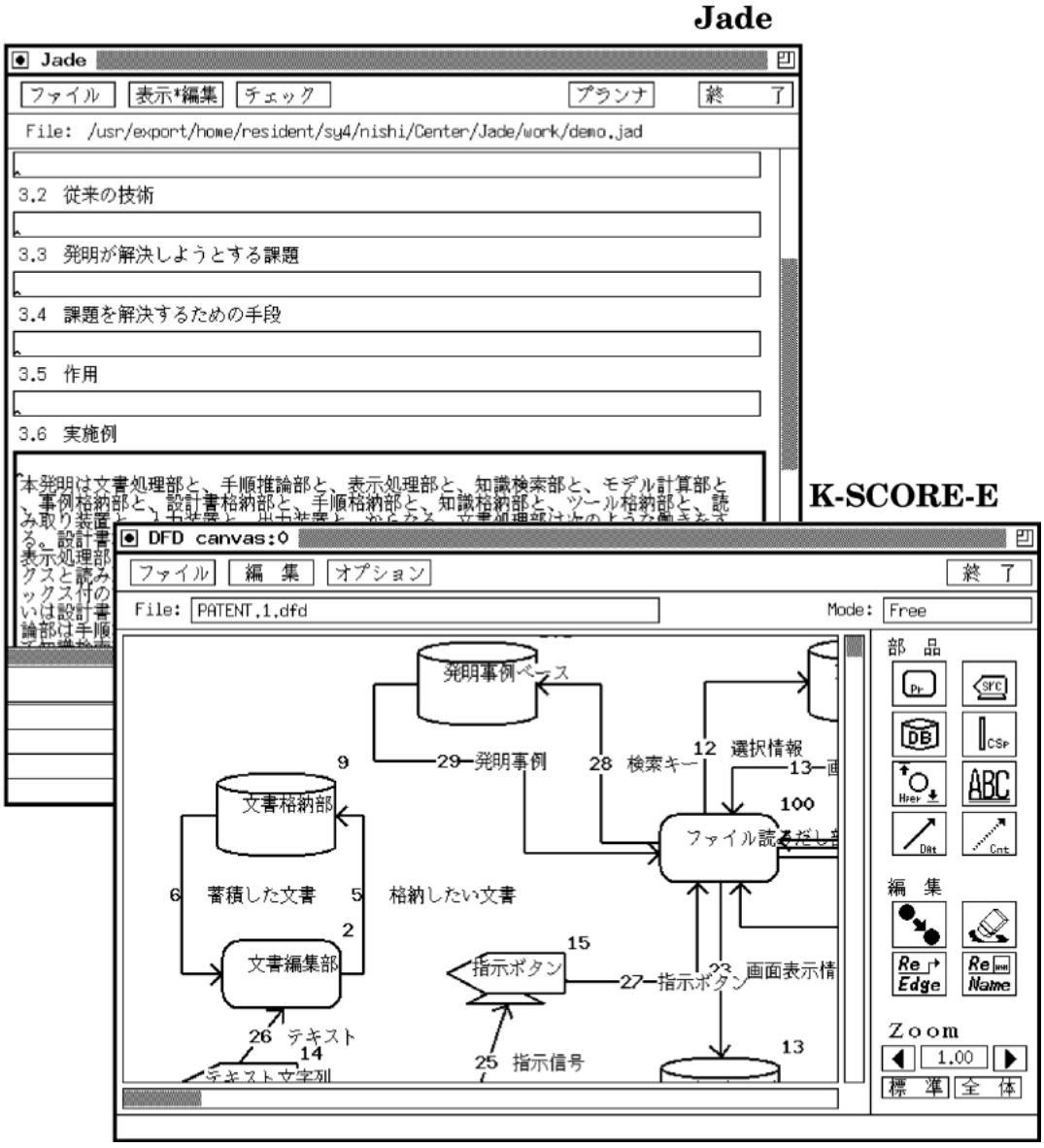


図 5.10 作業中の画面例

5.6.3　作業過程の記録

実験では，図面作成と文書化作業が中心であり，開発者には実施例の記載のみを依頼した．各開発者は，実施例で用いる図面を図形エディタ (K-SCORE-E) を用いて作成する．なお，今回はデータフロー図とフローチャートに限定した．

開発者が図形エディタ上で行なった編集操作は，上述の図面作成過程記録部が保存する．すなわち，各ノード，エッジの追加と削除，名前の変更コマンドの時系列を記録する．

こうして得られた作業過程をもとに，開発者にインタビューを行なうことによって分析を行なう．次章で，この分析結果の詳細について述べる．

5.6.4　実験結果の分析

各開発者の作業過程の記録とインタビューから明らかになった事を表 5.2 にまとめる．

表 5.2 分析結果

開発者 分析項目	A	B	C
(1)	DFD→FC→DFD	DFD→FC	DFD→FC
(2)	システム構成要素 (図形の意味)，機能的な関連	図面の基本構成要素	データの流れ (DFD)，機能的な関連
(3)	システムの目的，処理方式，図面レイアウト	システムの目的，図面レイアウト	システムの目的，処理方式，図面レイアウト
(4)	DFD と FC の一貫性確認，ビューポイントの切り替え	使用せず	ドキュメント作成

表 5.2 における分析項目は次の通りである．

(1) 図面の作成順序

(2) 開発者が意図したビューポイント

(3) ビューポイントの移動要因

(4) 図面説明文の作成意図

これらの中で，(1) から (3) は作業プロセスの遂行知識に関するものであり，(4) はドキュメンテーションに関するものである．以下では，これらの分析結果について詳細に述べる．

(1) 図面の作成順序

表 5.2 中の，DFD はデータフロー図，FC はフローチャートを表す．矢印は，図面の作成順序を表す．なお，図面の修正などによる後戻りは示していない．

それでは，各開発者毎にその特徴を述べる．

開発者 A は，最初にデータフロー図の作成を試みたが，すぐにフローチャートの作成に作業を変更した．そして，フローチャートが完成すると，その結果を基にして，データフロー図の作成を行なった．一方，開発者 A を除く他の 2 名はいずれもデータフロー図の作成を完了してから，フローチャートの作成に着手した．

この作成過程の違いは，設計の対象システムの特徴に深く関連する．開発者 A は，数式処理システムの設計を目的としており，事前に扱う数式が明らかになっていた．そのために，鍵となる作業は処理方式の決定であった．この理由から，開発者 A はまず最初にフローチャートで処理方式を明確にし，次にシステム機能の決定に作業を移した．一方，他の開発者はシステム機能を明確にすることを先に行なった．彼らの設計対象はデータ処理を中心とするシステムであり，すでにトランザクションが明確になっていたため，データフロー図を先に作成する結果となった．

(2) 開発者が意図したビューポイント

図面作成の過程で，開発者が何をビューポイントとして保持していたかを分析した．前述のように，ツールが提供するビューポイントは，図面の基本構成要素に基づいている．このために，各開発者は，設計 (発明) 対象を表現する時，おのずとこれらの基本構成要素に注目することを強いられる．

では，各開発者の作業過程を分析した結果を説明する．

開発者 B は，ツールが提供するビューポイントに忠実な作業を行なった．たとえば，開発者 B はデータフロー図において，まず最初に幾つかのデータベースを描いた．次に，各々のデータベースに関連する機能 (e.g.， データの検索) をシステムプロセスとして描いた．そして，あるデータベースで別な機能が必要になると，その他のすべてのデータベースにその機能を描くという作業を繰り返していた．すなわち，開発者 B はシステムプロセスだけを意識する，データベースだけを意識するというように，ツールの保持するビューポイントに基づいて作業を行なった．

一方，他の 2 名の開発者は，開発者 B とは異なるビューポイントを意識していた．設計経験の豊富な開発者 A と C は，対象システムの「機能的な関連」を意識した作業を行なっている．この「機能的関連」とは，システム構成要素の機能的なまとまり (たとえば，一つのサブシステムとしてまとめることができる複数のシステムプロセス) を表す．すなわち，対象システムを機能の違いによって幾つかのサブシステムに分割し，各々のサブシステムを設計するというトップダウンスタイルで作業を行なっていた．

開発者 A は，システムの構成要素の名前の意味にも注目していた．これは，上述の機能的関連よりもさらにドメインに依存するビューポイントである．たとえば，データ

フロー図に限ると，ノード (システムプロセス，データベース，ターミナル) では，主にその意味を詳細にし (e.g.，…処理手段→…推論手段)，エッジ (データフロー) では，その意味を抽象的なものにしている (e.g.，…制御信号→信号).

開発者 C は，データフロー図の作成時に，システム内のデータの流れを強く意識していた．設計経験の豊富な開発者 C は，データの流れにそって，大まかなシステムプロセスを描き，各々のプロセスの周辺を「機能的な関連」を意識して図面を描いていた．

以上のように，各々の開発者の保持するビューポイントは唯一ではない．図 5.11 は，これらのビューポイントを整理したものである．

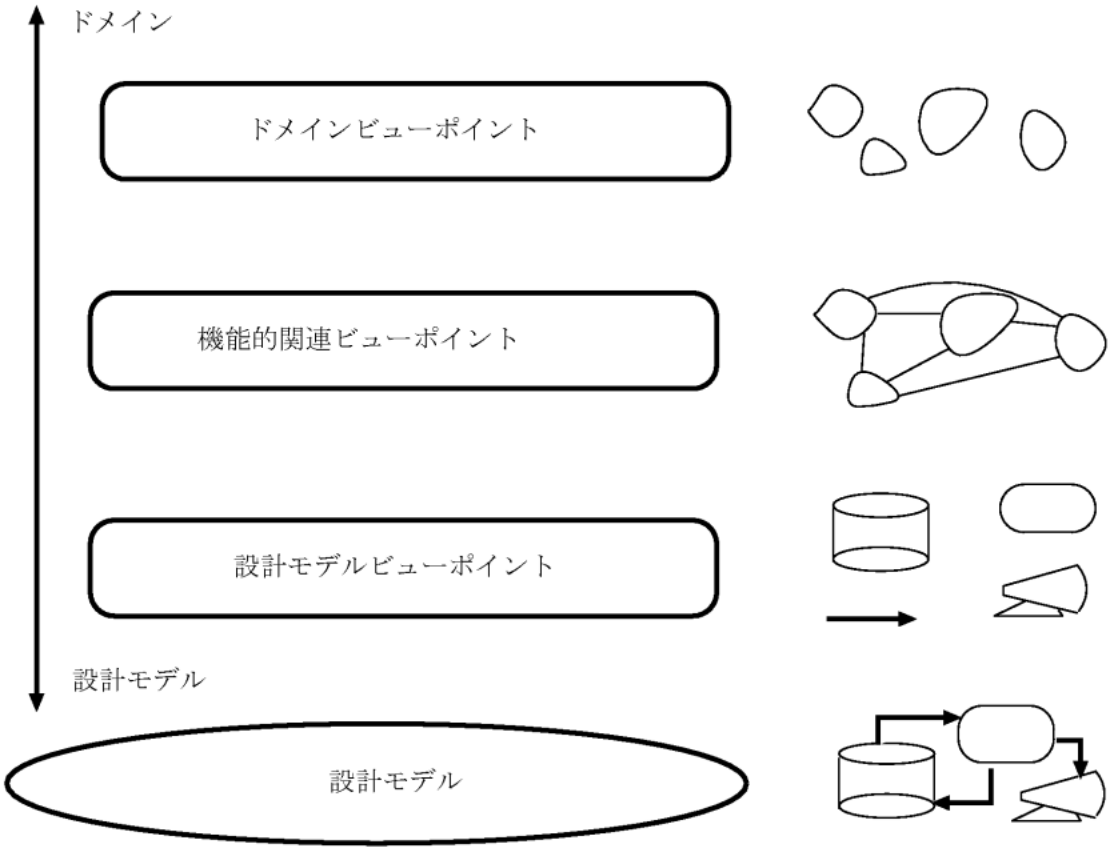


図 5.11 ビューポイントの構造

ドメインビューポイントは，設計対象のシステムを取り巻く世界を表す．開発者 A はシステムの構成要素の名称を，このビューポイントで決めている．その結果は，曖昧さのない厳密な定義となっている．しかしながら，専門用語が多用されていることも事実である．

機能的関連ビューポイントは，開発者 A や C が持っていたような対象システムの機能的なまとまりを表す．これは，システムの開発目的やシステムに求められる機能を意識するレベルである．ドメインビューポイントよりも，抽象的な用語が用いられるが，対象システムの機能を把握することは容易である．たとえば，幾つかのシステムプロセスがまとまって，あるサブシステムを構成するような場合,「そのサブシステム

名+具体的な機能名」という名称が付けられている (e.g., 結合荷重保持手段と結合荷重修正手段).

設計モデルビューポイントは図面の基本構成要素を表す．開発者 B は主にこのビューポイントを持っていた．また，開発者 C の「データフロー」もこのビューポイントである．このビューポイントでは，一般の計算機システムで用いられる語彙が多く用いられる (e.g., データ検索，格納，計算など)．従って，このビューポイントのみでの作業結果は，図面上に類似した要素が多く現れることになる．

次に，各開発者が保持するビューポイントと作業内容の関係について述べる．

ドメインビューポイントと機能的関連ビューポイントに注目していた開発者 A は，作業の前半で，思考錯誤をかなり行なった傾向がある．特にノードやエッジの名称には気を配っていた．そして，作業の後半では修正作業は減っている．

設計モデルを主に注目していた開発者 B は，当初描いた図面に冗長な機能が数多く存在することになった．そのために，作業の後半では，ほとんどすべての部分が修正の対象となり，再設計に近い作業が行なわれた．

機能的関連ビューポイントと設計モデルビューポイントの両方を意識していた開発者 C は，ノードやエッジの追加や削除といった修正作業はほとんど見られなかった．しかし，作業後半での名前の変更が多かった．

以上のように，保持するビューポイントの違いによって，作業内容，特に修正作業が異なることがわかった．

(3) ビューポイントの移動要因

ビューポイントの移動要因とは，図面作成過程で図面の修正 (図形の追加，削除，名称変更) を行なった時の状況を意味する．

全員に共通する要因に,「システムの目的」がある.「システムの目的」とは，図面作成で発明の目的を意識したことによって，ビューポイントが移り変わったことを表す．たとえば，開発者 C は，あるシステムプロセスを「データベース管理手段」としていたが，発明の目的に無関係であることから「データ入力手段」と名称を変更している．

開発者 A と C は「処理方式」を意識することが，他のビューポイントへ切り替わる要因であるとしている．これは，データフロー図を作成している時に，システムの内部処理の方式を考慮していたことを表す．すなわち，システム内部の処理を意識することによって，ビューポイントが移り変わる．たとえば，開発者 A の場合，フローチャートの変更によって，その処理に関連するシステムプロセスやデータフローの名称変更を行なっている．また，開発者 C の場合，システム内のデータの流れを強く意識していたが，これは処理をもともと意識していたからである．

「図面のレイアウト」を意識することも，ビューポイントの移り変わりに影響している．開発者 A と B は，当初図面のレイアウトに注意を払っていない．しかし，時間が経過するにつれて，線の交差や基本構成要素の重複が多くなり，図面の理解が容易ではなくなった．そのために，図形の再配置を行なった．この図形の再配置では，単

に移動だけではなく，名称の変更や基本構成要素の追加・削除といった修正作業も行なわれた．また，修正の過程では，各開発者ともに，(2) で述べたビューポイントを意識した作業を行なっている．たとえば，開発者 A は，各構成要素の機能的な関連の強い部分をまとめて再配置している．しかし，開発者 B は，図面の見やすさを求めたために開発者 A の図面よりもわかりにくい．

(4) 図面説明文の作成意図

図面説明文の作成意図とは，ツールの図面説明文作成機能を開発者がどのような意図で利用したかを表す．開発者 A は図面間のチェックと作業を切り替えるために，開発者 C はドキュメント作成のために図面説明文作成機能を用いた．

テンプレートによる図面説明文は，文としては非常に単純なものであり，改良すべき点が多い．しかし，仕様書の中では，そのまま利用できる部分もある．たとえば，ソフトウェア仕様書の実施例の項目では，まず最初に，図面 (多くは機能ブロック図) に描かれた構成要素を個々に説明する．この部分は，処理フローを意識したり，図面を補足しながら文書化することもあるが，図面に書かれた事を言い換えているだけである．従って，図面がきちんと書けていれば，機械的に作成された文でも十分に利用できると思われる．実際，開発者 B は，支援ツールが作成した文章をそのままで利用した．

図面の内容を自然言語で説明することは，ドキュメントの自動作成だけでなく，図面内容 (意味構造) の妥当性を確認することにも関係する．たとえば，データフロー図を機械的に言い替え，それを開発者が読むことによって，開発者 (あるいは顧客) の意図が定義されているかどうかを確認することができる．今回の試用では，開発者 A が，この目的のために説明文を作成した．

開発者 A は，さらに，次に注目すべきビューポイントを決める手段として，図面説明文を利用している．つまり，説明文のわかりやすさで，ビューポイントを決定している．たとえば，データフロー図に描かれた複数の入出力データをもつシステムプロセスの説明は，入出力データ間の依存関係が不明なために，非常に難解な説明文を作成する (図 5.12 上)．この説明文によって，開発者 A はそのシステムプロセスにビューポイントを移し，詳細化を行なった (図 5.12 下)．

5.7　関連研究

ソフトウェア開発を支援するために，プロセス中心のソフトウェア開発環境（PSEE）の研究が行われている．ソフトウェア開発におけるさまざまな活動をプロセスモデルとして表現し，開発者の役割や作業内容，関連するツール類を規定するというものである．

類似の研究には，メソッドエンジニアリングがある．メソッドエンジニアリングでは，開発者やそのプロジェクトに適した開発方法や環境の構築を迅速に行うことを主

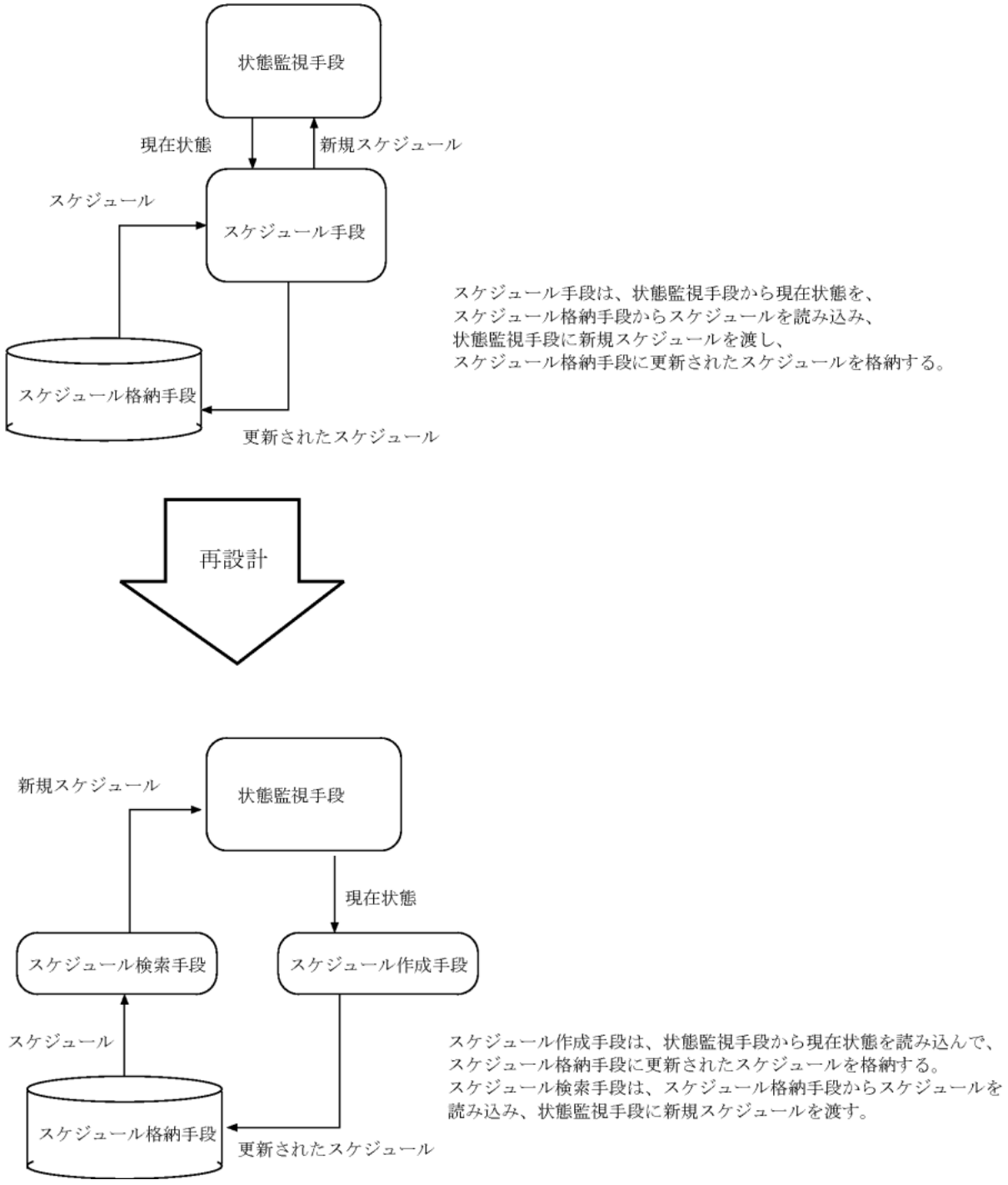


図 5.12 難解な説明文からの再設計

な目的としており，プロセス中心のソフトウェア開発環境をさらに発展させ，実用性を高めることを目指している．メソッドエンジニアリングでは，開発手法はコンポーネント化され，再利用可能な形で形式化が行われる．そして，必要に応じてコンポーネント群から必要なものを組合わせて新しい手法を作ることができる．いくつかの研究グループが，AI 技術の有効性を検証しており，プロセスモデリングの技術と統合する試みを行なっている [44, 62, 64]．AI プランニング技術を適用したものとしては，開発当初にプロセスを規定するフェーズでの利用 [17, 29, 37] や，仕様化作業におけるワークフロー支援のために利用している例がある [45].

本章で述べたシステムもまたこれらの類似研究と目的は同じであり，開発者の置かれている状況やプロジェクトの目的に対して，どのような作業をどのような順序で行えばよいかをアドバイスすることができる．さらに，文書作成プロセスの部分では，決まりきった作業を半自動化することも可能となっている点で，単に開発プロセスや環境を構築することから一歩進めたサポートを行うことが可能である．

しかしながら，本章で述べたシステムは，作業手順を立案し，設計者に提示まではするが，設計者の活動そのものを制御するまでには至っていない．作業間の時間的順序関係は，常に変化する可能性がある．例えば，作業 A が終了した後作業 B を実行すると計画していたものが，ある時点で，作業 A と B を並行して進めることになったり，複数人で分担して作業を進めたりと，計画の変更は実際の開発作業では頻繁に起こりうるものであるが，本研究においては今後の課題としたい．

5.8　まとめ

本章では，システム仕様化作業における課題をその作業プロセスという観点から述べ，その解決方法と支援システムを提案した．ビューポイントに基づくドメイン知識の整理方法は，ドメイン知識の利用と作業プロセスとの関連を明確にし，ドメイン知識の計算機上での表現および推論が可能であるという見通しを与える．具体的には，プランニング技術の応用した作業プロセスの自動生成が可能となった．

ソフトウェア仕様化の作業を実践的なレベルで解明する研究はようやく始まったばかりであり，よってプランニング技術も適用可能である一つの道筋を示したに過ぎない．実用面から見た場合，今後の課題を列挙する．

- 事例の語彙表現レベルの設定

  多くの事例は，そのドメインでのみ通用する語彙を用いている．そのために，ドメインが異なると同じものでも異なる語彙を用いて記述される．その結果，他のドメインで得られた知識の再利用が難しくなる．一つの解決方法としては，ドメインに依存しない汎用的な語彙表現を利用することが考えられる．

- プラン遂行知識の獲得

現状では，ビューポイントの切替え手順をプランとして推論することはできるが，そのプランの遂行は開発者任せであり，いつビューポイントを切替えるかに (アクティビティをいつ行なうか) 関しては未検討である．これは，作業者の意思決定に関する問題でもある．初心者は，指示された作業をどこまで行なえば良いか，いつ次の作業を行なうべきかを客観的に判断することはできないことから，このビューポイントの切替えに対してより強力な支援が必要である．一つの解決方法としては，ベテラン作業者の仕様書の記述過程を分析し，ビューポイントの切替えに関連した知識の獲得を行なういう方法が考えられる．

- 新たなビューポイントの定義と利用

  今回の試用では，ビューポイントの新しいタイプ (たとえば，機能的な関連ビューポイント) を明らかにすることができた．このようなビューポイントを直接支援するものは既存の CASE ツールも含めて見当たらない．今後は，この機能的な関連ビューポイントをベースとした編集機能を支援ツールに組み込みたいと考えている．

第 6 章

結論

本論文では，システムに対する要求変更やシステムの運用時に発生するさまざまな状況変化に対して，適応力を備えたソフトウェアシステムの実現へ向けて，階層プランニング技術の応用について述べた．各章で論じた内容をまとめる．

第 1 章では，適応力を備えたソフトウェアシステムが求められる背景を説明した．システムの規模拡大に伴い，複雑な開発や運用が強いられている現状を打破するには，システムに起こる変化に対するアクションをソフトウェアシステム自らが考える機能が必要であることを主張した．また，その実現手段の一つとして，AI プランニング技術の適用について論じた．

第 2 章では，プランニングを高速化するために，階層プランニングシステム ABSTRIPS で提案された緊急値による抽象階層の生成方法について，その課題を述べ，オペレータの適用結果を考慮した抽象階層の自動決定方法について述べた．本方法は，オペレータをその適用結果の類似性によって分類木としてまとめる．そして，オペレータの前提条件の達成の難易度を分析し，その結果を反映することによって抽象階層を得る．また，ロボット行動計画問題を例に，実験を行い，その有効性を検証した．第 2 章の要点は次の通りである．

本研究の課題

ロボットの行動計画などに用いられるプランニングにおいて, 探索効率の向上を図る．

課題を解決するための手段

オペレータに関する知識からプランニングを効率化するための抽象階層を自動生成し．それをもとに階層プランニングを行うシステムを実現した．

評価結果

従来の階層プランニングシステムに比べて探索効率の良いプランニングシステムを実現することができた．特に．上位層のプランをガイドとして下位層のプラン生成が効率化することができた．

ただし，2.6 節で述べたように，部分目標間に達成の順序関係が存在する場合，本手法が必ずしも有効に働かないケースもある．この種の問題に対しては別なアプローチが必要である．また，各層での問題解決は基本的に STRIPS ベースであるが，状態空間ベースの他の高速なプランニング技術を利用することは可能である．こうした手法を取り込みのハイブリッドなプランニングシステムを構築することは一つの可能性として考えられる．

次に，適応力のあるソフトウェアシステムの実現において，提案する階層プランニング手法がどのように適用できるかを，3 つのソフトウェアシステムを例に述べた．

第 3 章では，Web サービス合成問題に対するプランニング技術の応用について述べた．与えられた目標に対して，多数の利用可能な Web サービスから適切なサービスを選択し，迅速かつ効率よくサービスを合成する方法を提案し，いくつかの実験によりその有効性を示した．既存の Web サービス定義（WSDL 文書）に対して，若干の情報を追加することによって，比較的容易にプランニング技術を適用することができた．

第 3 章でも述べたように，本手法は，実行結果あるいは出力データのタイプが類似しているサービスが多い分野には有効な方法と思われる．また，数個のサービスを組み合わせる問題よりも，多数のサービスを合成する問題でその効果が発揮される．第 3 章の要点は次の通りである．

本研究の課題

Web サービスの自動合成問題に対してプランニング技術を適用するにあたって，以下の研究課題に取り組む．

- Web サービス合成を自動化し，さらに効率の良い合成を行うための手法
- 既存の Web サービスの定義からオペレータ定義への変換方法

課題を解決するための手段

Web サービスの定義言語である Web Services Description Language から，オペレータ定義への変換を行う．変換された各オペレータ定義に対して階層プランニング技術を適用する．

評価結果

階層プランニング技術を応用し，階層化に必要な緊急値を Web サービス仕様から自動生成し，複雑な Web サービス合成問題を効率よく解くことができ，実験によってその有効性を確認した．本手法は，実行結果あるいは出力データのタイプが類似しているサービスが多い分野には有効な方法と思われる．また，数個のサービスを組み合わせる問題よりも，多数のサービスを合成する問題でその効果が発揮される．

ただし，現在のプランニングシステムでは，Web サービスを合成するために，インスタンスレベルでの初期条件および目標を提供する必要がある．しかしなが

ら，実際にどのような値を持つかは，サービスの実行時でないと不明なケースが多い．そこで，状態変数や入出力データの意味を考慮したサービス合成を行うことで，サービス実行時を考慮した柔軟な結果を得ることが可能性として考えられる．また，問題によっては，必ずしも最適な解が得られるという保証がない．無駄なサービスを含む系列を合成する場合もある．これらに関しては今後の課題としたい．

第 4 章では，状況の変化を自ら検知し，自律的に振舞うソフトウェアシステムの実現に向けたアーキテクチャの構築について論じた．障害発生など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た．システム運用に関わる各種サービスを個別に定義しておき，システムの状況や達成目標に対して，適切なサービス系列を迅速に得ることができた．

本手法では，階層プランニングにおける緊急値を，前述のサービス定義から自動的に発見する．また，この緊急値にしたがって行われるプランニングは，サービス合成に必要な探索空間を狭め，効率の良いプランニングを行うことができることを実験により確認した．第 4 章の要点は次の通りである．

本研究の課題

大規模で複雑なシステムの運用管理業務軽減を目指し，状況変化に対して適切に対応する自律システム実現に向けて，特にシステム運用の戦略立案を行うゴール管理層を実現する．ゴール管理層では，迅速な戦略立案と不確実な状況にも対応できる能力が求められる．

課題を解決するための手段

システムの運用手順に関する知識をアクションとして定義することによって，階層プランニング技術を用いた効率の良い戦略立案の方法を提供する．

評価結果

状況の変化に対する運用手順の戦略立案を迅速に行えることを実証した．本手法では，階層化に必要な緊急値を運用に関するアクション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，プランニングに必要な情報の優先度を予め計算することができるので．無駄な情報収集活動を抑えることができた．これは，情報が不完全な場合でも，意味のあるアクション列を導ける可能性を示している．以上の点は，本研究の効果と言える．

一方，運用作業に関連するアクションをどのように定義するかはまだ十分な検討を行っておらず，今回の実験例でもアドホックな作りとならざるを得なかった．今後は，アクション定義のガイドラインを定めることも必要である．

さらに，プランニングにより，サービス合成の方法としてその効果を評価することはできたが，合成されたサービスの実行は，今後の課題である．今後は，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さまざま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．

また，今回は，アクション系列を迅速に得ることに主眼においたが，実際に得られるアクション系列の品質評価と高品質なものをいかにして得るかが課題である．たとえば，現在は，アクション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さまざまな条件を考慮していく必要がある．また，時間に関する情報をアクション定義の中に取り込む必要もあろう．

第 5 章では，ソフトウェアシステムの仕様化作業を支援するために，その諸活動を分析するソフトウェアプロセスモデルの技術と階層プランニング技術を連携させ，開発支援環境の構築とその検証について述べた．システム仕様化作業に関するドメイン知識をビューポイントという概念で整理し，状況に合わせて適切なビューポイントとその切り替え手順を立案するしくみを実現するためにプランニング技術を応用した．第 5 章の要点は次の通りである．

本研究の課題

仕様化作業において，作業プロセスに関する知識を蓄積し，作業状況に合わせて提示する方法を確立する．また，文書作成に関する作業の負荷軽減を目指す．

課題を解決するための手段

ソフトウェアプロセスモデリングの技術を用いて仕様化作業プロセスをタスクレベルで形式的に定義する．階層プランニング技術を用いて，与えられた目標に対するタスクおよびその実施順序を立案し，必要な情報の提供，ツールの起動などを制御する．さらに，図形から半自動的に説明文を作成することによって，文書作成時間の短縮を図る．

評価結果

仕様書作成支援ツールを試作し，異なるスキルを有する開発者に試用実験を行った．ビューポイントを切り替える順序（プラン）を自動的に合成することができ，それにしたがって，開発者に対して適切な情報やツールを提供することが可能と

なった．加えて，図形から半自動的に説明文を作成することによって，仕様の検証手段としても利用可能となることが明らかになった．

ただし，現状では，ビューポイントの切り替え手順をプランとして推論することはできるが，そのプランの遂行は開発者に委ねており，積極的な振舞いはしていない．システム開発の初心者は，指示された作業をどこまで行なえば良いか，いつ次の作業を行なうべきかを客観的に判断することはできないことから，このビューポイントの切替えに対してより強力な支援が必要である．一つの解決方法としては，ベテラン作業者の仕様書の記述過程を分析し，ビューポイントの切替えに関連した知識の獲得を行なういう方法が考えられる．

以上の３つの応用を通じて，状況変化に対して，適応力のあるシステムを構築する上で，階層プランニングの技術が有効であることを示すことができた．

しかしながら，本研究の成果は，第 1 章で述べた「自己適応型システム」の実現へ向けての第一歩であり，今後も継続して取り組むべき研究課題は多数ある．それらの中で，特に重要と思われる課題を以下にまとめる．

(1) ターゲットシステムの状況変化の観測

第 1 章で述べた自己適応型システムの実現へ向けて，プランニングシステムとともに重要な構成要素が，ターゲットシステムの状況を観測する部分である．ターゲットシステムのどの部分を観測対象とするか，対象からどのような手段で値を観測するか，さらには観測値から有用な情報をどのように発見するのかなど，正確かつ効率の良い観測方法が求められる。一般に情報システムでは，サーバやネットワーク機器などのハードウェアレベル，オペレーティングシステムレベル，アプリケーションレベルと複数レベルでの監視が必要であるが，これらを統一的に扱うための技術は存在せず，現実にはそれぞれの対象毎に個別の監視技術が利用されている．また，一般的には情報の種類や量が大量に存在することからターゲットシステムに対して有用な情報を効率良く発見する手段の確立が課題である．

(2) 実行時の変化を考慮したプランニング

現実の世界では，状況や目標は時々刻々と変化しており，そうした中でのプランニングは，プランナが立案したアクション列のうち全部または一部が実際には実行できないこともありうる．なぜならば，プランニングの際には想定していなかった変更が実行時に発生するからである．現在，こうしたケースでは，その都度，再プランニングを行わなければならない．しかしながら，プランニングにかかるコストを考えた場合，プランの実行に失敗するたびに再プランニングを行うことは避けたい．より効率の良い，プラン生成から実行までをカバーする方法が求められる．

(3) 時間に関する知識の表現と利用

アクションの順序だけでなく，時刻や時間に関する知識を扱うことも重要である．たとえば，ある一定時間しか動作しないであるとか，指定された時刻に起動し，何秒以内にタスクを完了しなければならないといった時間に関する制約情報を扱うケースが多々ある．本論文で述べた階層プランニングシステムでは，こうした時間に関する知識を直接的に扱うことができない．

(4) 予防的措置のしくみの構築

ソフトウェアシステムが自ら問題を把握し，自ら考え，問題を解決する能力を持つことが自己適応型ソフトウェアの理想と考えているが，この自己適応力には何かの事象が起きてからそれに対応するということだけでなく，将来起こるであろう課題を予測して未然に防ぐという意味合いもある．現在のアプローチは，前者の実現を目指しているものであるが，その先の課題として後者に関しても考慮すべきと考える．具体的には，アクション定義とアクション列の実行制御の改良が必要である．

(5) 複数プランニング手法との連携

自己適応型ソフトウェアの中核となるプランナに関しては，一つのプランニング方式に限定せずに，複数の方式を備え，それらを適宜切り替えて使う、あるいは複数の方式を組み合わせた使い方も必要と考えている．複数方式を備えることで，可用性の向上という効果も期待できる上，一層の高い品質のプランを高効率で得ることができると見込んでいる．

今後も，本研究の成果をより多くのソフトウェアシステムに適用し，改善を重ねていきたいと考えている．

謝辞

本論文をまとめるにあたり，終始ご指導と励ましをいただいた電気通信大学大学院情報システム学研究科大須賀昭彦教授に深く感謝の意を捧げます．親切なご指導，御鞭撻を賜りました岡本敏雄教授，渡辺俊典教授，末廣尚士教授，田原康之准教授に感謝の意を表します．また，本研究を進める上で多くの有益な助言をいただいた中川博之助教に感謝の意を表します。株式会社東芝にご在職中の頃より長年に渡ってご指導を賜りました現国立情報学研究所本位田真一教授に心より感謝の意を表します．

さらに，本研究の遂行にあたり，社会人学生として著者の活動を理解し，励まし，協力していただいた株式会社ボイスリサーチ代表取締役深田優氏をはじめとし，職場の同僚に感謝します．

最後に，本研究の遂行と論文執筆に理解を示し，様々な面で協力し，励ましてくれた，妻美佐に心から感謝する．

参考文献

[1] IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications, 1998.

[2] Web Services Description Language (WSDL) Version 2.0, 2007.

[3] V. Agarwal, G. Chafle, S. Mittal, and B. Srivastava. Understanding Approaches for Web Service Composition and Execution. In *Compute '08: Proceedings of the 1st Bangalore annual Compute conference*, pp. 1–8, 2008.

[4] M. Alia, G. Horn, F. Eliassen, M. U. Khan, R. Fricke, and R. Reichle. A Component-Based Planning Framework for Adaptive Systems. In *OTM Conferences (2)'06*, pp. 1686–1704, 2006.

[5] V. Ambriola, R. Conradi, and A. Fuggetta. Assessing Process-centered Software Engineering Environments. *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 3, pp. 283–328, 1997.

[6] J. S. Anderson. Plan Abstraction based on Operator Generalization. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pp. 100–104, 1988.

[7] J. S. Anderson and S. Fickas. A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. In *Proceedings of the 5th international workshop on Software specification and design*, pp. 177–184, 1989.

[8] N. Arshad and D. Heimbigner. A Comparison of Planning Based Models for Component Reconfiguration. *Technical Report CU-CS-995-05*, 2005.

[9] P. Bertoli, R. Cimatti, J. Slaney, and S. Thiebaux. Solving Power Supply Restoration Problems with Planning via Symbolic Model Checking. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pp. 576–580. Wiley, 2002.

[10] M. J. Carman and C. A. Knoblock. Inducing Source Descriptions for Automated Web Service Composition. In *Proceedings of the AAAI 2005 Workshop on Exploring Planning and Scheduling for Web Services, Grid, and Autonomic Computing, Technical Report WS-05-03*. AAAI Press, 2005.

[11] M. J. Carman, L. Serafini, and P. Traverso. Web Service Composition as Planning. In *Proceedings of ICAPS 2003 Workshop on Planning for Web Services*, pp. 28–35, 2003.

[12] J. Cheng and K. B. Irani. Ordering Problem Subgoals. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, pp. 931–936, 1989.

[13] R. Conradi, M. Hagaseth, J. Larsen, M. N. Nguyên, B. P. Munch, P. H. Westby, W. Zhu, M. L. Jaccheri, and C. Liu. EPOS: Object-oriented Cooperative Process Modelling. *Software Process Modelling and Technology*, pp. 33–70, 1994.

[14] B. Curtis, M. I. Kellner, and J. Over. Process Modeling. *Communications of the ACM*, Vol. 35, No. 9, pp. 75–90, 1992.

[15] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. *Lecture Notes in Computer Science*, pp. 18–38, 2001.

[16] T. DeMarco. Structured Analysis and System Specification. *Classics in Software Engineering*, pp. 409–424, 1979.

[17] J.A. Diaz-Pace and M. R. Campo. Experiences with Planning Techniques for Assisting Software Design Activities. *Applied Intelligence*, Vol. 29, No. 1, pp. 56–78, 2008.

[18] S. Dustdar and W. Schreiner. A Survey on Web Services Composition. *International Journal of Web and Grid Services*, Vol. 1, No. 1, pp. 1–30, 2005.

[19] R. Estlin, T.and Castano, B. Anderson, D. Gaines, F. Fisher, and M. Judd. Learning and Planning for Mars Rover Science. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.

[20] S. Fickas. Automating the Analysis Process: An Example. In *Proceedings of 4th International Workshop on Software Specification and Design*, pp. 58–67. CA: IEEE Computer Soc. Press, 1987.

[21] R. Fikes and N. J. Nilsson. STRIPS: A New Approach to Theorem Proving Problem Solving. *Artificial Intelligence*, pp. 189–208, 1971.

[22] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 2, No. 1, pp. 31–57, 1992.

[23] P. K. Garg and W. Scacchi. ISHYS: Designing an Intelligent Software Hypertext System. *IEEE Expert: Intelligent Systems and Their Applications*, Vol. 4, No. 3, pp. 52–63, 1989.

[24] M. R Genesereth. The Role of Plans in Intelligent Teaching Systems. *Intelligent Tutoring Systems*, pp. 137–156, 1982.

[25] Object Management Group. Unified Modeling Language.

[26] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, pp. 403–414, 1990.

[27] S. Honiden, K. Nishimura, N. Uchihira, and K. Itoh. An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems. *IEEE Transaction on Software Engineering Vol. 20, No. 11*, pp. 849–867, 1994.

[28] A. E. Howe, A. V. Mayrhauser, R. T. Mraz, and D. Setliff. Test Case Generation as an AI Planning Problem. *Automated Software Engineering*, Vol. 4, pp. 77–106, 1997.

[29] K. E. Huff and V. R. Lesser. A Plan-based Intelligent Assistant that Supports the Software Development. In *SDE 3: Proceedings of the third ACM SIGSOFT/SIGPLAN software engineering symposium on Practical Software Development Environments*, pp. 97–106, 1988.

[30] 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. 1–8, 2008.

[31] C. A. Knoblock. Learning Hierarchies of Abstraction Spaces. In *Proceedings of Sixth International Workshop on Machine Learning*, pp. 241–245, 1989.

[32] C. A. Knoblock. Learing Abstraction Hierarchies for Problem Solving. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 923–928, 1990.

[33] C. A. Knoblock. An Analysis of ABSTRIPS. In *Proceedings of 1st International Conferece Artificial Intelligence Planning Systems*, pp. 126–135. 1992.

[34] C. A. Knoblock. Automatically Generating Abstractions for Planning. *Artificial Intelligence*, Vol. 68, pp. 243–302, 1994.

[35] 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, pp. 1D1–1. 人工知能学会, 2007.

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

[37] C. Liu, R. Conradi, M.N. Nguyen, and A.I. Wang. Planning Support to Software Process Evolution. In *Proceeding of the 10th International Conferece on Software Engineering and Knowledge Engineering*, pp. 17–20, 1998.

[38] S. A. Liu and M. Mernik. Special Track on Artificial Intelligence and Software Engineering. In *5th International ICST Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2010.

[39] Z. Liu, A. Ranganathan, and A. Riabov. Modeling Web Services using Semantic Graph Transformations to aid Automatic Composition. *Proceedings of IEEE International Conference on Web Services*, pp. 78–85, 2007.

[40] A. Mediratta and B. Srivastava. Applying Planning in Composition of Web Services with a User-Driven Contingent Planner. *IBM Research Report*, pp. 1–9, 2006.

[41] A. M. Memon, Martha E. Pollack, and Mary Lou Soffa. Hierarchical GUI Test Case Generation Using Automated Planning. *IEEE Transactions on Software Engineering*, Vol. 27, pp. 144–155, 2001.

[42] 三浦純. ロボットにおけるプランニング (特集：プランニング技術の進展と新たな応用展開). 人工知能学会誌, Vol. 16, No. 5, pp. 617–622, 2001.

[43] M. D. R. Moreno, D. Borrajo, and D. Meziat. Process Modeling and AI Planning Techniques:A New Approach. In *Proceedings of Second International Workshop o Information Integration and Web-based Applications and Services*, 2000.

[44] M. D. R. Moreno, D. Borrajo, and D. Meziat. Planning-based generation of process models. In *Proceedings of the ECP-01/PLANET Workshop on Automated Planning and Scheduling Technologies in New Methods of Electronic, Mobile and Collaborative Work*, pp. 22–341, 2001.

[45] M. D. R. Moreno and P. Kearney. Integrating AI Planning Techniques with Workflow Management System. *Knowledge-Based Systems*, Vol. 15, No. Issue 5-6, pp. 285–291, 2002.

[46] J. Mostow and A. E. Prieditis. Discovering Admissible Heuristics by Abstracting and Optimizing : A Transformational Approach. In *Proceedings of the 11th International Joint Conferences on Artificial Intelligence*, pp. 701–707, 1989.

[47] D. S. Nau. Current Trends in Automated Planning. *AI Magazine*, Vol. 28, No. 4, pp. 43–58, 2007.

[48] D. S. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., 2004.

[49] D. S. Nau, S. K. Gupta, and W. C. Regli. AI Planning versus Manufacturing-operation Planning: A Case Study. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, pp. 1670–1676, 1995.

[50] 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.

[51] 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, pp. 542–531, 2009.

[52] K. Nishimura, Y. Tahara, and A. Ohsuga. A Knowledge-Based System for Software Specification. In *Proceedings of 4th Joint Conference on Knowledge-Based Software Engineering*, pp. 235–236, 2010.

[53] 西村一彦, 本位田真一. 複合ビューポイントによる仕様化プロセスの分析. 情報処理学会論文誌, Vol. 34, No. 5, pp. 1074–1086, 1993.

[54] 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.

[55] 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, pp. 139–146. 情報処理学会, 2008.

[56] 西村一彦, 中川博之, 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. 人工知能学会誌, Vol. 26, No. 1, pp. 107–115, 2011.

[57] C. Niskier, T. S. E. Maibaum, and D. Schwabe. A Pluralistic Knowledge-Based Approach to Software Specification. In *ESEC '89: Proceedings of the 2nd European Software Engineering Conference*, pp. 411–423, 1989.

[58] J. Peer. SESMA - Semantic Service Markup:Domains and Use Cases. *Technical Report University of St. Gallen, Switzerland*, 2004.

[59] J. Peer. Web Service Composition as AI Planning- A Survey *. *Technical Report University of St. Gallen, Switzerland*, 2005.

[60] S. R. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Web Service Composition. In *Proceedings of the 11th World Wide Web Conference*, 2002.

[61] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, pp. 43–54, 2004.

[62] C. Rolland. Method Engineering: State-of-the-Art Survey and Research Proposal. In *Proceeding of the 2009 conference on New Trends in Software Methodologies, Tools and Techniques*, pp. 3–21, 2009.

[63] E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *Artificial Intelligence*, Vol. 5, pp. 115–135, 1974.

[64] M. Saeki, K. Iguchi, K. Wen-yin, and M. Shinohara. A Meta-Model for Representing Software Specification and Design Methods. In *Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process*, pp. 149–166, 1993.

[65] O. Seog-Chan, L. Dongwon, and Soundar R. T. K. A Comparative Illustration of AI Planning-based Web Services Composition. *SIGecom Exchanges*, Vol. 5, No. 5, pp. 1–10, 2006.

[66] S. Shlaer and S. J. Mellor. *Object-oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, 1988.

[67] S. J. J. Smith, D. S. Nau, and T. A. Throop. Computer Bridge - A Big Win for AI Planning. *AI Magazine*, Vol. 19, No. 2, pp. 93–106, 1998.

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

[69] B. Srivastava and S. Kambhampati. Challenges in Adapting Automated Planning for Autonomic Computing. In *Proceedings of the 5th International Conference on Automated Planning and Scheduling*, pp. 41–44, 2005.

[70] D. Sykes, W. Heaven, J. Magee, and J. Kramer. From Goals to Components: A Combined Approach to Self-Management. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pp. 1–8, 2008.

[71] J. D. Tenenberg. Planning with Abstraction. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pp. 76–80, 1986.

[72] J. D. Tenenberg. Abstraction in Planning. Technical report, Rochester, NY, USA, 1988.

[73] S. Thiebaux and M. Cordier. Supply Restoration in Power Distribution Systems - A Benchmark for Planning under Uncertainty. In *Proceedings of the 6th European Conference on Planning*, pp. 525–532, 2001.

[74] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., 1988.

[75] 経済産業省. 高度情報化社会における情報システム・ソフトウェアの信頼性及びセキュリティに関する研究会の中間報告書, 2008.

[76] 情報処理推進機構, 社団法人日本情報システム・ユーザー協会. 重要インフラ情報システム信頼性研究会報告書, 2009.

研究業績

研究論文

学術誌論文

- 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.
- 西村一彦, 本位田真一, 複合ビューポイントによる仕様化プロセスの分析, 情報処理学会論文誌, Vol. 34, No. 5, pp. 1074-1086, 1993.
- Shinichi Honiden, Kazuhiko Nishimura, Naoshi Uchihira and Kiyoshi Itoh. An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems, IEEE Transaction on Software Engineering Vol. 20, No. 11,pp. 849-867,1994.
- 西村一彦, 中川博之, 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. 人工知能学会誌, Vol. 26, No. 1, pp. 107-115, 2011.

国際会議・ワークショップ論文

- Masatsugu Oshima, Masayuki Ida, Kazuhiko Nishimura. A Web-based Tool for Aiding SCM and B2B Education, International Conference on Internet-business: business trends, systems, and education (iBIZ2007), 2007.
- Kazuhiko Nishimura, Yasuyuki. Tahara, and Akihiko Ohsuga. A Knowledge-based System for Software Specification. In Proceedings of 4th Joint Conference on Knowledge-Based Software Engineering, pp.238-239, 2010.

国内研究会論文

- 西村一彦, 本位田真一. 複合ビューポイントによる仕様化プロセスの定性的分析, 情報処理学会ソフトウェア工学研究会, Vol.81-6, pp.41-48, 1991.
- 西村一彦, 本位田真一. 協調的仕様化作業のためのプロセス分割可能性の検討, 第4回ソフトウェアプロセスワークショップ, 1992.

- 西村一彦, 春木和仁. 複数ビューに基づくシステム仕様化過程の分析, 情報処理学会全国大会講演論文集 第 48 回平成 6 年前期 (5), pp.105-106, 1994.
- 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, pp.139-146, 情報処理学会, 2008.
- 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. 542-531, 2009.（優秀論文賞受賞）

解説論文

- 折原良平, 荒木大, 西村一彦. 仕様獲得と知識獲得, 情報処理学会, Vol.33, Np.6, pp.605-611, 1992.
- 西村一彦, 春木和仁. 情報システムの企画・計画技術, システム制御情報学会, Vol.37, No.3, pp.152-159, 1993.

出版

共訳書

- 続・オブジェクト指向システム分析, 近代科学社, 1992.
- オブジェクト指向モデリング, 日経 BP, 1994.
- Netscape ONE 完全技術解説, 日経 BP, 1997

雑誌記事

- 西村一彦. Web サービスの“ 今 ”を把握し,“ 将来 ”を展望する, CIO Magazine 2002 年 7 月号, IDG ジャパン, 2002.
- 西村一彦. 激変するビジネス環境に適応する新時代の IT 基盤とは, CIO Magazine 2002 年 12 月号, IDG ジャパン, 2002.
- 吉岡 信和, 横山 重俊, 西村 一彦,Prabin Karanjit. 第 7 章：クラウドが向かう先 クラウド間相互連携, グリッド連携, Software Design 2010 年 11 月号, 技術評論社,2011.

関連論文の印刷公表の方法および時期

全著者名 ：西村一彦, 松本一教

論文題目 ：階層型問題解決システムにおける抽象階層の決定方法

印刷公表の方法および時期 ：人工知能学会誌, Vol. 6, No. 5, pp. 701-709, 1991

（第 2 章の内容）

全著者名 ：Shinichi Honiden, Kazuhiko Nishimura, Naochi Uchihira, Kiyoshi Itoh

論文題目 ：An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems

印刷公表の方法および時期 :IEEE Transaction on Software Engineering, Vol.20, No.11, pp. 849-867, 1994

（第 3 章の内容）

全著者名 ：西村一彦, 中川博之, 中山健, 田原康之, 大須賀昭彦

論文題目 ：階層プランニングによる Web サービスの自動合成

印刷公表の方法および時期 :ソフトウェアエンジニアリングシンポジウム 2008, pp.139-146, 情報処理学会, 2008

（第 3 章の内容）

全著者名 ：西村一彦, 中川博之, 田原康之, 大須賀昭彦

論文題目 ：自律システム実現に向けたアーキテクチャの構築

印刷公表の方法および時期 ：人工知能学会誌, Vol. 26, No. 1, pp. 107-115, 2011

（第 4 章の内容）

全著者名 ：西村一彦, 本位田真一

論文題目 ：複合ビューポイントによる仕様化プロセスの分析

印刷公表の方法および時期 ：情報処理学会論文誌, Vol. 34, No. 5, pp. 1074-1086, 1993

（第 5 章の内容）

全著者名 ：Kazuhiko Nishimura, Yasuyuki. Tahara, and Akihiko. Ohsuga

論文題目 ：A Knowledge-based System for Software Specification

印刷公表の方法および時期 ：Proceedings of 4th Joint Conference on Knowledge-Based Software Engineering, pp. 235-236, 2010

（第 5 章の内容）

著者略歴

西村 一彦（にしむら かずひこ）

1963 年 12 月	長野県に生まれる
1982 年 4 月	東京理科大学理工学部経営工学科卒業
1986 年 4 月	東京理科大学理工学研究科経営工学専攻修士課程入学
1988 年 3 月	東京理科大学理工学研究科経営工学専攻修士課程修了
1988 年 4 月	株式会社東芝入社，ソフトウェア技術研究所配属，ソフトウェア生産技術に関する研究に従事
2001 年 1 月	株式会社アイ・ティ・アール入社
2002 年 1 月	株式会社ボイスリサーチ設立
2008 年 4 月	電気通信大学大学院情報システム学研究科社会知能情報学専攻博士後期課程入学
2011 年 9 月	電気通信大学大学院情報システム学研究科社会知能情報学専攻博士後期課程修了予定