修　士　論　文

指導教官　　杉山公造　教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

950049　鈴木 聡

審査委員：　知識　杉山　教授（主査）
知識　下嶋　助教授
知識　亀岡　教授

2001 年 2 月

1

1.1

近年,「ソリューションビジネス」が多くの注目をされている．ソリューションビジネスはもともと米国のコンピュータメーカーがハードウェアビジネスの不振から脱するための戦略として旧来の事業から転換を図ったビジネスである．独立の情報処理会社を除けば，コンピュータメーカーにおいてサービスはハードウェアに附属する保守などの範疇に含まれるもののみであったが，これをハードウェアビジネスから独立させ，収益をあげる事業としたのである．最近は，日本の企業を含めほとんど全ての情報関連会社がソリューションビジネスの看板を掲げている．さらには，重電機メーカーなど，これまで情報関連とは言われていなかった企業においても，その多くが何らかの形でソリューションと銘打ったビジネスを展開するようになってきた．

それでは，ソリューションビジネスとは一体どのようなものなのであろうか．ソリューションビジネスとそうでないビジネスの明確な区別はあるのであろうか．

重電メーカーがソリューションビジネスに進出する現象については，今日では計算機による情報処理が，重電等を含む全ての産業で扱われるシステムの一部となっているため，コンピュータメーカー以外の企業も自らの分野における情報関連事業に手を広げたものであると見ることもできる．しかし，もともと社会財や産業財を提供していた企業が新たにソリューションビジネスを開始するにあたっては，そのようなことは昔からやっていることの呼び方を変えただけではないのか，という反応が起こる場合もある．そのようなときに，従来のビジネスとどこが違いどこが同じなのかを明らかにしなければ事業の改善は望めないであろう．

ソリューションビジネスは今後もますます広がると思われるが，企業が効果的にソリューションビジネスを展開するためには，まずその本質を明らかにし，それに基づいて業務を支援するツールを用意する必要がある．

1.2

このような背景のもとで，本研究ではまずソリューションビジネスの特徴を調べ，従来のビジネスとの違いを明確にする．そしてソリューションビジネスを構成する要素の中から，この特徴を良く反映して効果的なビジネス遂行に重要なファクターとしてソリューションメニューを取り上げる．さらに，ソリューションビジネスを支援するツールとして，ソリューションメニューの形式的記述体系を開発する．

1.3

本章以降では，次のように論を展開する．

第 2 章ではソリューションビジネスの歴史と現状を概観し，現在一般的に言われているソリューションビジネスの定義では説明しにくい事例があることを指摘する．次に基盤となる製品の機能レベルとユーザーニーズの乖離という特性からソリューションビジネスを特徴づける．

第 3 章ではソリューションビジネスをおこなう上で重要なファクターとして，ソリューションメニューを位置付ける．そして，製品開発プロセスにおけるイノベーションの研究の一つである製品システム観の理論から，その役割を考察し，ソリューションビジネスを効果的におこなう上でソリューションメニュー体系が備えるべき性質を明らかにする．

第 4 章では人工知能や知識ベースの分野でおこなわれているオントロジー工学研究について述べ，ソリューションメニュー体系を記述するための方法論としてのオントロジーの機能について考察する．

第 5 章では効果的なソフトウェア開発組織を構成するための知見をデザインパターンの形でまとめた研究について述べ，それをソリューションメニューに変換するこ

との可能性を調べる．

第 6 章では第 3 章と第 4 章の考察を元に, ソリューションメニュー体系を記述するためのオントロジーを設計する．そのオントロジーの有効性を実例に基づいて検証するために，第 5 章で述べたソフトウェア開発組織のデザインパターンからソリューションメニュー体系を作成し，その検討をおこなう．

第 7 章では本研究全体のまとめと，今後の展望を述べる．

2

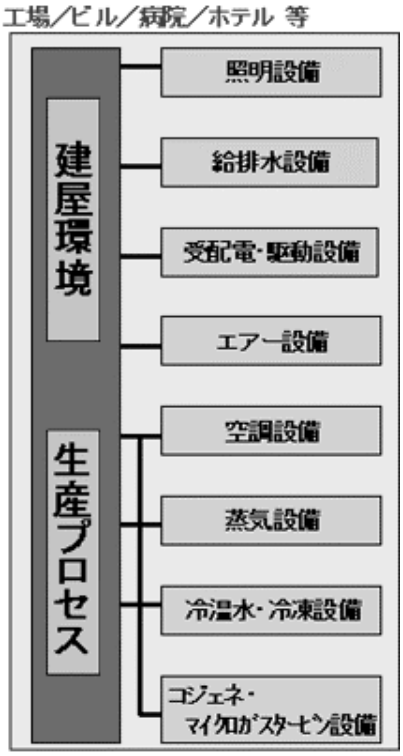
2.1

今日では情報サービス産業をはじめ，多くの企業がソリューションビジネスを掲げている．一般にソリューションとは「顧客の持つ課題の解決方法を提案すること」であるといわれているが，厳密な定義は与えられていない．また，その組織的・経営的特徴を分析した研究も少ないのが現状である．しかし一方で，多様で変化の速い環境下で収益構造を変えるために IT を利用して企業の仕組みそのものを変革することが広く求められており，ソリューションビジネスの需要は一層増加すると考えられ，その効果的な遂行を支援するツールが必要とされている．

「ソリューションビジネス」が注目されるようになったのは，米国のコンピュータメーカーが 1990 年代前半の業績悪化に直面した際に，ハードウェア製造を立てなおすと共にそれ以外での収入の道を得るための戦略として取り入れ，短期間で業績を好転させたためであった[1]．ハードベンダーは 80 年代までは保守サービスをコンピュータ製品に付帯するものとして販売し，大きな利益をあげていた．また，各ハードベンダーは自社独自規格でユーザーを囲い込み，プリンターなど周辺機器をセットで売ることができたのである．しかし，技術の進歩により機器の信頼性が向上すると保守サービスの重要性は低下し，製品付帯サービスの利益は大きく縮小した．また，次第にオープンアーキテクチャが主流となり規格の標準化が進むと，米国外メーカーの参入等によりハードウェア製品そのものの価格も低下し，さらにはユーザーの囲い込みもできなくなった．そこで各社は，ハードとサービスを分離し，自社製品だけでないマルチベンダーのシステムインテグレーションの提供を始め，収益の回復を図った．

この中で，不採算製品ラインを切り離して自社の得意分野に集中するリストラを推進すると共に，他企業との連携を推進して，コンサルティング，システムインテグレーション，アウトソーシング，サポートなどのソリューションを提供できる体制を整えたのである．

さて，このように，ソリューションビジネスは多くが情報技術(IT)関連企業が掲げているものであるが，近年では重電などの従来型産業といわれている分野でもソリューションと銘打ったビジネスを展開している企業も多い．図 2-1に三菱電機のエネルギーソリューション[2]を示す．この例では，工場，ビル，病院などを対象に受配電設備のエンジニアリングとそれに付帯する保守などのサービスをソリューションとして位置付けている．



エネルギー設備

エネルギーエンジニアリング

エネルギー管理システム

エネルギーサービス

設備提供サービス

マネジメントサービス

供給管理サービス

2-1 [2]

今日ではほとんどすべての産業分野でコンピュータと通信による情報活用の必要性が高まっているので，従来型産業の企業が自らの得意分野におけるコンピュータ利

用に結びつけたビジネスに参入したとも考えられる．しかし，これらの中にはコンポーネントとしてのコンピュータを直接は利用していないサービスをソリューションと呼んでいるものも多い．例えば上記のエネルギーソリューションでは，受配電システムの設計時に CAD を使用することや，機器制御のための組み込みコントローラ以外での，コンポーネントとしてのコンピュータを使用しない場合でもソリューションと呼んでいる．これらの社会財，産業財のビジネスでは従来からコンサルティングやシステムインテグレーションがビジネスの柱であったので，それをソリューションと呼んでいるものもある．よって，IT を利用したビジネスがソリューションである，と言うわけにはいかない．

また，近年ではソリューションと銘打っているものの，その内容が本当にソリューションの定義にふさわしいものであるかという疑問を持たれるものもある．例えば「Java チップを核とした組み込み用途向け Java ソリューション」（F 社）があるが，その内容は Java チップ，JTRON（OS），評価キット，開発ツールのセットであり，従来の製品と変わらないように思われる．

2.2 JEIDA

前述のようにソリューションと銘打ったビジネスが急速に拡大し，その用語の使用法などに混乱が見られる中，(社)日本電子工業振興協会（JEIDA）[1]はソリューションビジネス普及促進のために，ソリューションの推進方法を「ソリューションアーキテクチャ」として定義すると共に標準化を行い，各ベンダが顧客に対してソリューションを共通に提供できるようにすると発表している[4]．その定義は次のようである．

・ソリューションの定義：ソリューションとは顧客の経営課題をＩＴと付加サービスを通して解決するビジネス技法
・ソリューションフレームワーク：ソリューションを提供する，商品群をサービス，コンポーネントウエア，ミドルウェア，ネットワーク，プラットフォームに体系化
・ソリューションビジネス技法：ソリューションを構築する手順を４つのステップに

[1] 2000 年 11 月 1 日に(社)日本電子機会工業会と合併し，現在は(社)電子情報技術産業協会（JEITA）である．

標準化

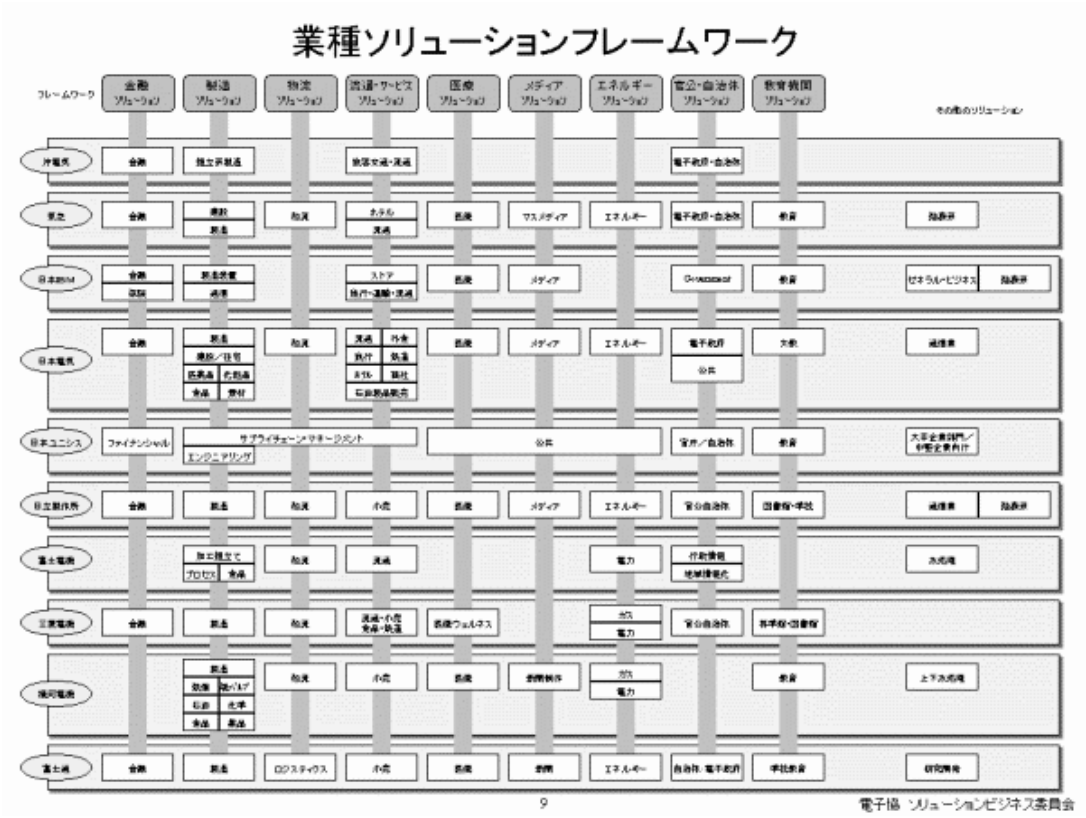
1．コンサルテーション

2．情報システム化構想立案

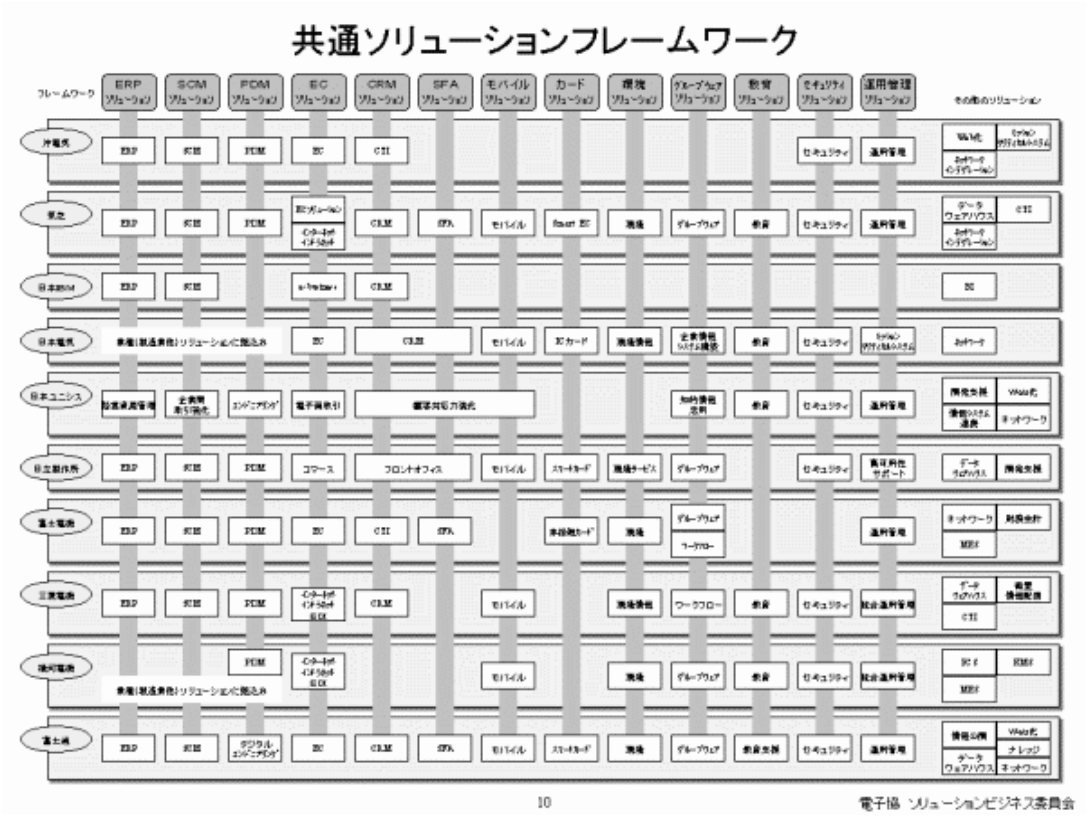
3．業種ソリューションフレームワーク，共通ソリューションフレームワーク，およびソリューション商品マップからソリューションメニュを選択

4．ソリューションフレームワークから，サービス（ＳＩ，ＰＫＧ，アウトソーシングなど)，ミドルウェア，ネットワーク，プラットフォームを選択

図 2-2，図 2-3にソリューションフレームワークの概要を示す.



2-2 [4]



2-3 [4]

しかし，この定義に従うと，IT を直接には使用しない産業財などのシステムインテグレーションについてはソリューションビジネスではないとされてしまい，現実のビジネスの広がりに対応できない．そこで，以下の節では情報産業と従来産業の比較を行い，ソリューションというべきビジネスとそうでないものの違いが明らかになるようにソリューションビジネスの特徴付けを行う．

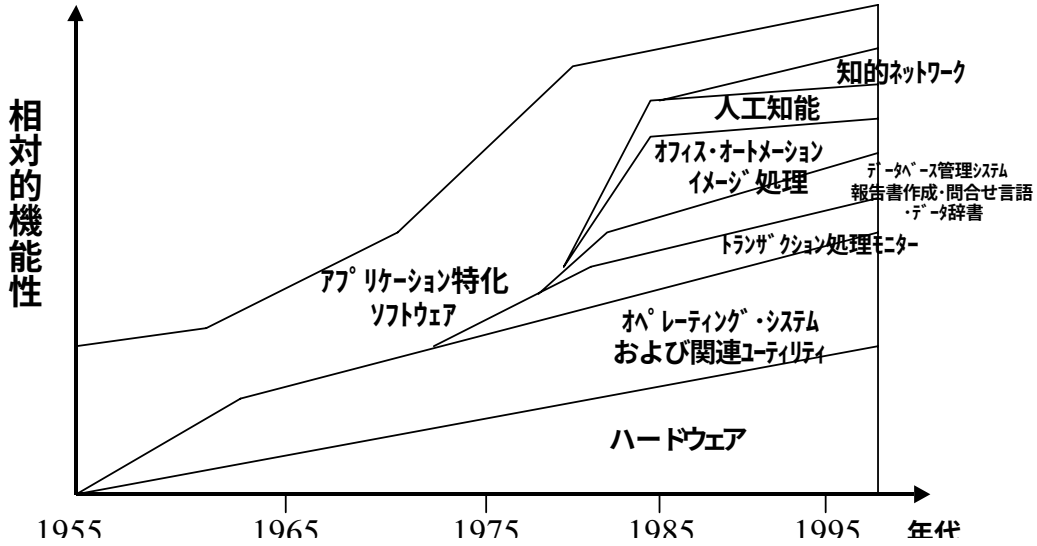
2.3

2.1節で述べたように，「ソリューションビジネス」は歴史的にはコンピュータハードメーカーを中心とする情報産業の戦略として出現した．そこで，ここではまず，情報産業と従来産業との比較をおこなう．

さて，情報産業を情報技術(IT)の進化過程から分析すると，その特質は，デファク

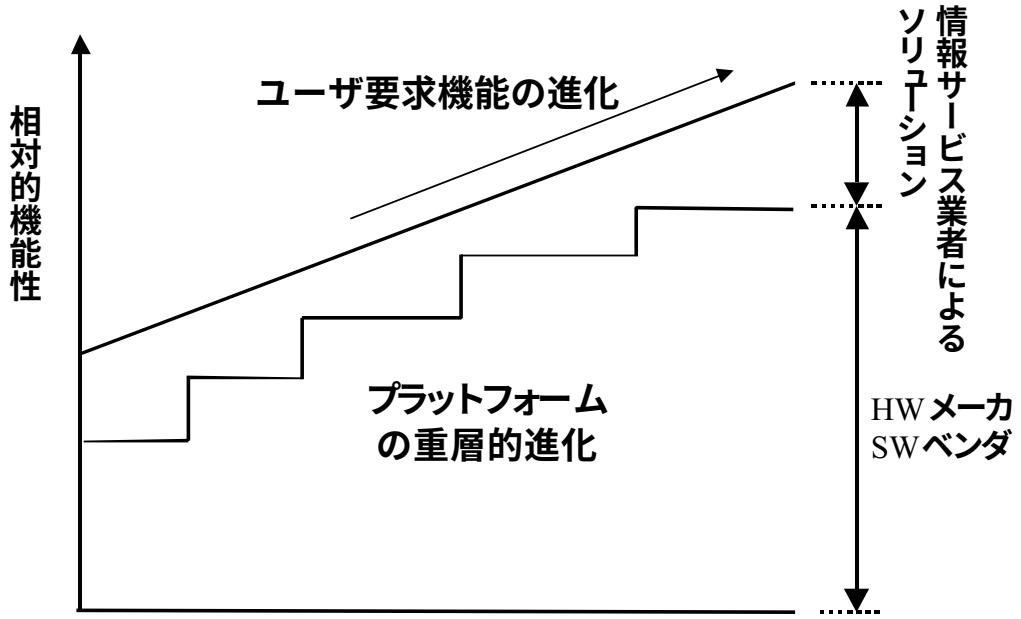
トスタンダードとしてのプラットフォームの階層化と，ソリューション産業形成の２点にあらわれる[2].

情報産業においては，図 2-4に示すように，ハード・ソフトは重層的に進化して階層構造的なプラットフォームを成す．プラットフォームのデファクトスタンダード競争が常に上位に移行していくため，下層のプラットフォームでの差異は次第に吸収され，差別化が難しくなる．2.1節で述べたように，ハードウェア単体でのビジネスが難しくなったことは，情報産業におけるこの性質を裏付けるものである．



2-4 [2]

一方でニーズの多様化，個別化への効果的対応がますます要求され，ユーザ要求水準は上昇し続けるので，その乖離を埋めること，すなわち，具体的効用を持たないコンピュータを個別利用者のニーズに適合させて効用を実現すること（ソリューション）が必要となる（図 2-5）．この部分をビジネスの中心とする戦略が，ソリューションビジネスとなる．ハード単体の収益が悪化したベンダーは，90 年代に業態転換してソリューションベンダーとなったのである．

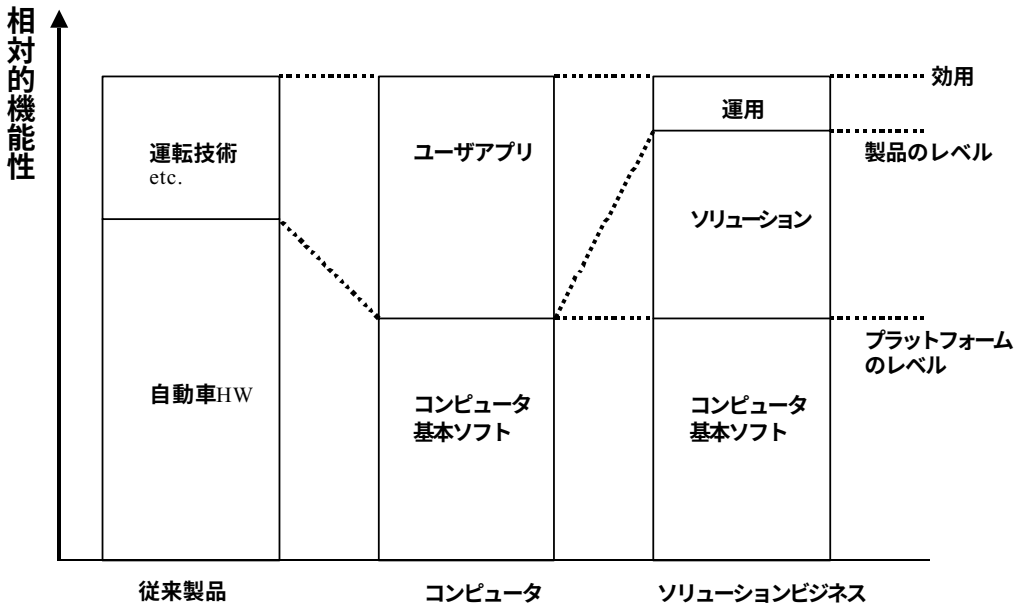


2-5

さて，このような観点から従来産業と情報産業との対比を行うと，つぎのようになる．

これまでのハードウェア（家電，自動車）ビジネスでは，その製品に期待する効用と使い方はユーザーの選択に任されており，ハードベンダーがソリューションに相当するものを提供することは無かった（図 2-6 左）．コンピュータについても，初期のビジネスではハードおよびプラットフォームとなるソフトから具体的な効用を得る方法の開発はすべてユーザーが担っていたのであり，その時点では IT 産業はソリューションビジネスではなく，従来のビジネスと同様であった（図 2-6 中）．しかし，ソリューションビジネスが成立したときには，自動車などでは「プラットフォーム」と顧客に売るものとしての「製品」のレベルが一致しているのに対して，IT では「プラットフォーム」＋「ソリューション」が「製品」のレベルとなり，明確な違いが生じる（図 2-6 右）．ここで，ソリューションの部品としてソフトウェアパッケージを

開発しておき，プラットフォーム＋パッケージ＋個別開発チューニングという形でビジネスを行う場合が多い．広い範囲でこのパッケージが有用ならば，新たなプラットフォームとしてデファクトスタンダードになる可能性があり，プラットフォームの上層への進化が生じるのである．プラットフォームと製品のレベルが一致している従来製品の場合には，このような階層的進化は起こりにくい．



2 - 6

2.4

さて，情報産業の特質に関するこの議論は，社会財や産業財などにおけるシステムインテグレーションでも成り立つものである．

2.1節で例としてあげた受配電設備のソリューションビジネス（図 2-1）では，受配電機器，照明・空調器具，制御機器などをもとに，各顧客に対してそれぞれの要求

にあったシステムを構成し販売するとうビジネスをおこなっている．これらのシステムインテグレーションを主とするビジネスでは，ベースとなる技術や機器があり，それらの組み合わせの上に個別開発部分をのせて顧客ニーズに適合させ，提供する．すなわち，顧客に売るものとしての「製品」はプラットフォームだけではなく，それに加えて顧客個別の効用を実現する「ソリューション」も含まれる．このように，直接 IT を使用していない分野においても，ソリューションビジネスは成立し得るのである．

そこで，ソリューションビジネスの特徴をまとめると，

・各顧客に対する個別開発部分が小さいもの：従来製品

・各顧客に対する個別開発部分が大きいもの：ソリューション

となる．たとえ「システムインテグレーション」であっても，すべての顧客に対して同じものを提供するのであれば，それはソリューションとは言いがたい．一方，IT ではないものであっても，プラットフォームの上に個別開発をおこなって提供するものであれば，それはソリューションである．

3

3.1

前章で述べたような特徴をもつソリューションビジネスを遂行する上で重要な要素は何であろうか.「顧客ニーズを実現する効用を提供すること」を効果的にビジネスとしておこなうためにはどうすれば良いのであろうか.

前節で見たように，ソリューションビジネスでは「プラットフォーム＋ソリューション」が「製品」に相当する．そこで，実際のビジネスではこの「ソリューション」がどのような形で提供されているのかを考える.

例えば営業支援システムの構築というビジネスを取り上げると，プラットフォームとしてはサーバー用コンピュータやネットワークシステム，データベースなどがあり，それらを使って個々の顧客向けにシステムを作る．そのソリューションの提供の際には，「当社ではお客様に合わせて営業支援システムの構築を行います」というような漠然とした形では行われない．一般的にはあらかじめ図 3-1のように，業種やシステムの利用場面など，何らかの形で範囲を限定した体系を用意しておき，その中から顧客のニーズに最も近いものを選び提供する．このことは，2.2節で述べた JEIDA のソリューションアーキテクチャの定義での「業種ソリューションフレームワーク，共通ソリューションフレームワーク，およびソリューション商品マップからソリューションメニュを選択」に相当する．JEIDA の定義では，ソリューションメニューは各社で現在主流となっている業種・分野に固定された形で体系化されているが，本研究で

は前節の考察に基づいて，顧客に販売するものとしての「製品」に相当する「プラットフォーム＋ソリューション」を何らかの形で体系化したものをソリューションメニューと呼ぶこととする．以下の節ではこのソリューションメニューの役割について考察する．

SOLUTIONVISION 一覧

業務ソリューション

- **電子商取引@COMMERCEVISION**
- **CRM@CRMVISION**
- **SFA@SALESFORCEVISION**
- **デジタルエンジニアリング@DIGITALENGINEERINGVISION**
- **IR 活動支援@DISCLOSUREVISION**
- **環境分野@ECOVISION**
- **研究開発部門向け@R&DVISION**

業種ソリューション

- **製造業向け@ECCALSVISION**
- **金融業向け@FINANCIALVISION**
- **物流業向け@LOGISTICSVISION**
- **小売業向け@RETAILVISION**
- **医療機関／事業者向け@HEALTHCAREVISION**
- **自治体向け@INTERCOMMUNITYVISION**
- **情報サービス産業向け@PRESSMEDIAVISION**

3-1 [6]

3.2

ソリューションビジネスにおいてはイノベーションはどのように生起するのだろうか．本節では従来の製品開発イノベーションとの類似点，相違点をもとに，ソリューションビジネスのイノベーションにおけるメニューの役割を分析する．

3.2.1 新製品開発研究におけるイノベーションの理論

自動車や電器製品などの従来産業については，イノベーション研究の一つの分野として企業内での製品開発組織のメカニズムに関して，機能開発部門と製品開発プロジェクトとの関係や，複数プロジェクトにわたる組織的知識の継承の問題などが分析されている．青島[6]は新製品開発研究のレビューを行い，その中で，個別の製品開発プロジェクトを分析の対象として製品開発を独立した問題解決活動と捉える単一プロジェクト研究と，複数製品を分析の対象として個々の製品開発プロジェクトを超えたレベルでの戦略や組織的構造を扱う複数プロジェクト研究を比較し，「製品開発プロセス観」と「製品システム観」に基づく新製品開発研究の理論を提唱している．

この中で，個別プロジェクト研究による製品開発プロセス観については，どのような組織構造が新製品開発成果に影響を与えるのかを議論し，要素技術やコンポーネントの機能別に分かれた組織と，機能横断的なプロジェクト組織とのバランスにより開発成果が最適化されるとしている．例えば，自動車メーカーにおける新製品開発では，製品のコンセプトの段階から生産販売にいたる開発プロセス全体に対して強い影響力を持つマネージャーが各機能別の開発部門から集められたメンバーを統括するマトリックス状組織が，高い成果を上げていると報告されている[8].

また，複数プロジェクト研究による製品システム観については，自動車など複雑な統合アーキテクチャをもつ製品システムでは，顧客がその製品に何を求めるかによって異なるシステム観が存在し，製品開発プロジェクトを通してシステム観の探索・学習が行われ，開発すべき機能要素の分化が生じるとされている．例えば，楠木[9]は，ファクシミリ開発におけるシステム観について次のように述べている．まず，ファクシミリをそれまでのテレックスに代わる通信機というシステム観でとらえた場合に

は，他の通信装置と比較した場合の電送速度が重要な要素となり，ファクシミリを構成するモデムのスピード向上や画像の圧縮などの開発がおこなわれる．しかし，画像を電送できるコピー機であるというシステム観においては，画像の鮮明度が重視される．また，電話機の附属機能として見た場合には，電話のように普及させるための低コスト化開発が優先される．このように一つの製品にも異なるシステム観が存在するのである．

3.2.2 製品開発プロセス観によるソリューションビジネスの分析

さて，前節で見たようにソリューションビジネスでは，プラットフォームとソリューションを体系化したメニューをもとに，個々の顧客のニーズに適応した効用を実現するシステムやサービスを提供する．よって，ソリューションビジネスを製品開発プロセス観の視点で見ると，各メニューに対応して特定のサービスや技術を提供する職能グループが機能開発部門に，個々の顧客に対して実際の製品を提供するためのソリューションプロジェクトが製品開発プロジェクトに相当すると考えられる．自動車などの従来製品に対する研究結果を適用すれば，ソリューションビジネスにおいても，各職能グループがそれぞれのサービスや技術をより良いものに改善していくとともに，個々のソリューションプロジェクトでは各グループから機能横断的に集められたメンバーがプロジェクトマネージャーに統括されて実際の製品を提供するマトリックス組織が望ましい．

3.2.3 製品システム観によるソリューションビジネスの分析

次に，ソリューションビジネスを製品システム観の視点から分析する．前節で述べたように，ソリューションビジネスにおいてはその特質から個々の顧客が求めるものが原則的に異なり，また，ソリューションの基盤となるプラットフォームも重層的に常に進化しつづける．よって，ソリューションビジネスではシステム観探索の傾向がより強まり，どのようなプラットフォームの上にどのような効用を実現するのかという観点を個々のプロジェクトから探索・学習し，メニューを開発することが重要となる．

例として表 3-1にヒューレッドパッカード(HP)におけるサービス部門の組織改革の変遷と，その組織における主な業務を示す[5]．また，そこに見られるシステム観の

変化を示す.

3-1HP

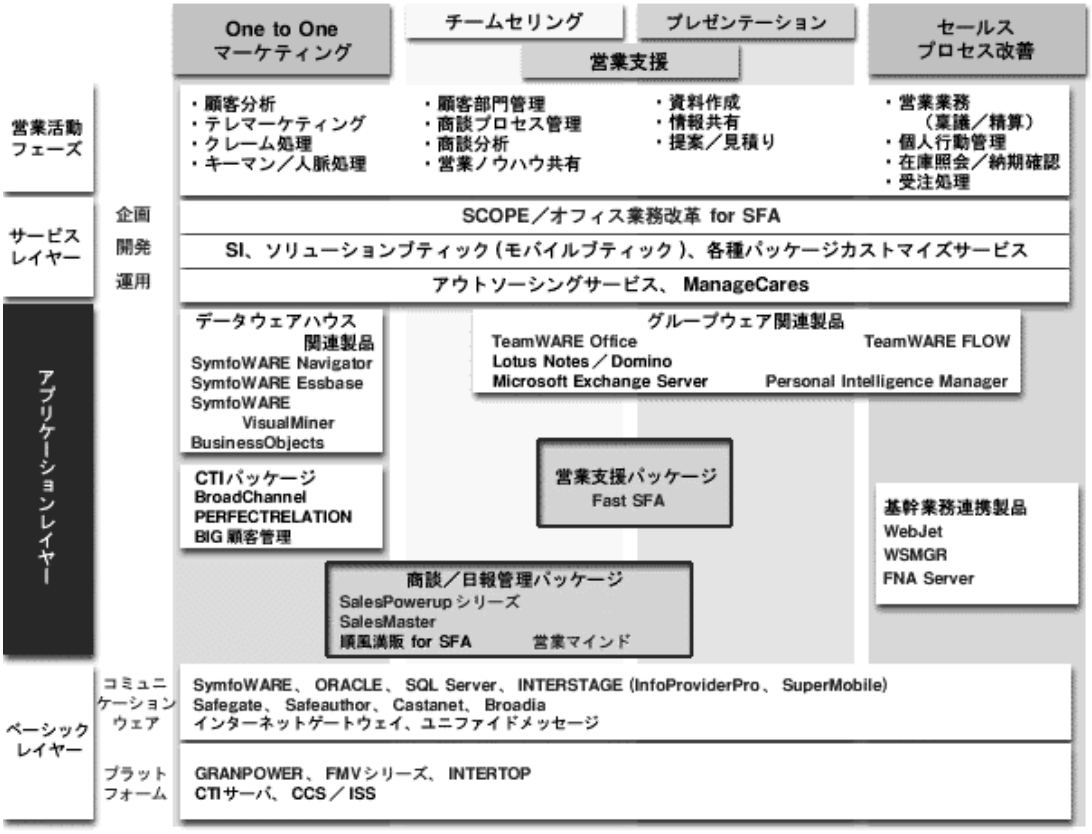
年度	組織	業務	プラットフォーム	提供する効用
初期	ワールドワイドカスタマーサポートオペレーションズ(WSCO)	アフターサービス的保守とサポート	販売済み自社製品	アフターサービス
1991 年	コンピュータサービス機構(CSO)が分離独立	レガシーシステムへのソフトウェアインテグレーション	レガシーシステムと自社 UNIX 機	レガシーシステムを活用すること
1992 年	製品販売組織のユーザー業界単位の縦割り部門への編成に合わせて WSCO と CSO のリソースを各販売部門へ分散	顧客ニーズに密着して自社製品の販売をサポートすること	主に自社製品	分野に密着したアフターサービス
1995 年	WSCO と CSO を統合再編しコンピューティング機構設立	WindowsNT を含むハードソフトのインテグレーション	NT と UNIX（主に自社製品）	マルチプラットフォームシステムのサポート
1997 年	HP コンサルティング設立	HP 製品の販売とは別のオープンシステムインテグレーションやこれに関わるコンサルティング，ネットワーク管理･教育などのサービス	マルチベンダー	戦略プランニング段階からのシステム選定とインテグレーションのサポート

プラットフォームが自社製 UNIX 機からマルチベンダー・マルチ OS へと移行していき，ユーザー効用がアフターサービスから戦略的システム構築へと変化するに従い，「サービス」というメニューを進化させていることが分かる.

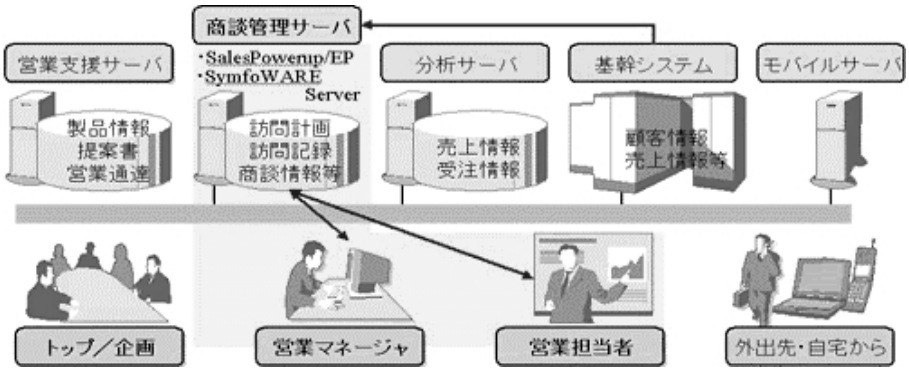
このサポートサービスの例は2.2節の JEIDA の定義におけるソリューションフレームワークのレベルでのメニュー進化の例である．より下位層のメニュー体系の例として，富士通の営業支援システム（SFA）に関するソリューションである，「SALESFORCEVISION」[10]を取り上げる．図 3-2にその体系図を示す.

このソリューションメニューは，チームセリング，OnoToOne マーケティング，営業支援&プレゼンテーション，セールスプロセス改善という 4 つのサブメニューを持

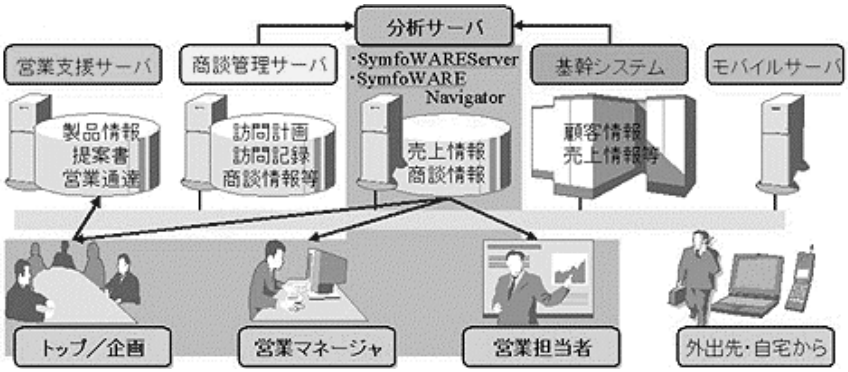
ち，それらを構成する標準コンポーネントが列挙されている．サブメニューのシステム構成例のうち 2 つを図 3-3と図 3-4に示す．また，それぞれのサブメニューに対してカタログ中でされている説明を図 3-5に示す．



3-2 SALESFORCEVISION [10]



3-3 [10]



3-4 OneToOne [10]

チームセリング：
担当者に聞かないと商談進捗状況が分からない
訪問回数のわりには受注が伸び悩んでいる
営業日報が次の仕事に役立っていない
OneToOne **マーケティング：**
どうも最近、営業効率が悪いと感じる
売上高や取引値をみても変化の理由が具体的に見えてこない

3-5 SALESFORCEVISION

これらのソリューションメニューの構成を製品システム観から分析すると，次のことが分かる．

まず，SALESFORCEVISION というメニューを構成するコンポーネントは，それぞれのサブメニューでもほぼ共通である．図 3-2でベーシックレイヤーの部分はすべてのサブメニューで同じであり，アプリケーションレイヤーも各サブメニューにまたがって使用されているものが多い．また，サブメニューのシステム構成例の図でも，全体の構成はすべて共通であり，その中での各コンポーネント間の情報の流れや使用場面の違いが描かれている．

これは，SALESFORCEVISION というメニューで提供されるソリューションは「営業支援をするシステム」という大きなシステム観を持っており，その中のサブメニューであるチームセリング，OneToOne マーケティングなどはそれぞれ，「組織的営業をおこなうためのシステム」，「顧客ターゲットを絞り込み営業効率を上げるためのシステム」などという，より分化したシステム観を持っていることを意味している．この分化したシステム観にしたがって，各コンポーネントの組み合わせで達成する効用が整備され，基本構成は同じシステムで多くのニーズに対応する体系となっているのである．ここでイノベーションとして新たなシステム観の分化が起こり，別のサブメニューとして体系化される場合には，その効用を達成するために各コンポーネントの機能が改善されるか，あるいは部分的なシステム構成が変更されると考えられる．

システム観の変化によるイノベーションによる現象の別の例としては，次のことが挙げられる．2.1節で述べたように，米国のコンピュータハードメーカーは 90 年代以降に不採算ラインを切り離して製品ラインを自社の得意分野に集中させた．これを「サービス」に対するシステム観の変化から考えると，ユーザーが求める効用が「製品のアフターサービス」から「オープンスタンダードなプラットフォーム上でシステムを構築する」ことに移行したことに対応して，ハードウェアの役割も変化して「システムを構成するコンポーネント」としての機能を高めることが重要となり，フルラインアップの製品を揃えるよりも資源を集中することが有利となったのである．

3.2.4　イノベーションのためのメニューの役割

ここまでのソリューションビジネスにおけるイノベーションの分析に基づくと，ソリューションメニューが備えるべき性質には次のような特徴がある．

・最終的な製品は個別のプロジェクトが生み出すものでありメニュー自体ではない．ただし，メニューは顧客に直接見えるものであり，ニーズにマッチした効果のわかりやすいものである必要がある．

・メニューは多様なニーズに対するそれぞれのソリューションを何らかの形で体系化したものであり，メニュー体系の一つの要素に対応する実際のソリューションには多様性がありうる．

・前項とは逆に，メニューの要素に対応する実際のソリューションの多様性はできるだけ小さいほうがビジネス効率上は良い．

・メニューは不変ではなくニーズやスタンダード技術の変化により更新される必要がある．さらに，同じニーズとスタンダード技術の元でも，プロジェクト遂行を通じた学習から，より良いソリューション提供へのイノベーションが生じ，メニューが変革され得る．

従来の製品に近いものとしては，ソリューションメニューの中にはソフトウェアなどの「パッケージ」製品が含まれることがある．これらはデファクトスタンダードのプラットフォームの上に独自のソフトウェアレイヤーを作成している場合が多い．ビジネスの効率上，このパッケージは，個別開発部分はなるべく小さく，同時に適用範囲はできるだけ広くなるように開発しておくことが重要であろう．

また，製品から遠いサービスなどの場合でも，各顧客に対する提供内容の差はできるだけ小さく，かつ広い範囲をカバーするような体系にまとめることが望ましい．

このように，どのようなプラットフォームの上にどのような効用を実現するのかという観点からメニュー体系を開発すれば，ソリューションビジネスに効果的であろう．

3.3

3.3.1 メニュー体系記述の形式

このようなソリューションビジネスのメニュー体系を開発するためにはどのようにすれば良いであろうか.

例にあげた SAFLESFORCEVISION では，カタログの図版とその説明文として営業支援ソリューションに関するメニューが描かれている．しかしその体系化には明示的な規則が無く，構造が不明確である．また，個別のプロジェクト事例と，そのときに使用したメニューとの関係が記述されていない.

また，ソリューションビジネスの支援として，これまでに多く研究されてきている知識ベースシステムなどを活用しようとする場合には，ソリューションメニュー体系を形式的に記述することが必要となる.

そこで，ソリューションメニュー体系の記述について，ここまで考察してきたようなソリューションビジネスの特徴に合致した形式的記述の方法論があれば，より効果が期待できる．その記述形式の要件としては，次のことがあげられる.

(1)ソリューション提供の対象とするドメインの記述および顧客の問題とそれに対するソリューションの記述を備え,

(2)メニュー体系の構成を動的に変更することが可能であり,

(3)各メニューとそれぞれのプロジェクト事例との関係に基づき体系化を支援する機能を持つことが望ましい.

4

4.1

近年，人工知能や知識ベースの構築に関連してオントロジーが広く研究されている．しかし同時に，工学的研究におけるオントロジーの定義や，オントロジーを利用することによる効果などについては合意がなく，模索段階である[11]．Gruber[12]はオントロジーとは概念化の明示的仕様であるとしている．ここで概念化とは，情報処理の対象とする世界に存在するものの概念と，それらの間の関係を示す．対象世界のモデルを構築するときに，どのような方法・考え方で概念を抽出し，関係付けるかを明確にしたものが明示的仕様である．溝口[11]は知識ベースを構築する際の基本概念としてのオントロジーの種類とレベルを次のように分類して説明している．

・オントロジーの種類

1. ドメインオントロジー：
   対象とする領域（ドメイン）に存在するものに関するオントロジー
2. タスクオントロジー：
   問題解決過程に固有の概念化であるタスクに関するオントロジー

・オントロジーの 3 レベル

1. 対象世界に存在する概念の切り出しとそれらの階層構造を記述したもの．
2. 各概念に対する制約や公理的記述を加えることで，概念の意味定義や関係の記述を与えたもの．
3. オントロジーで記述されたものの情報処理における動作を記述したもの．タスクオントロジーはこのレベルに相当し，手続的な記述がされる．

オントロジー工学研究は，その目的によって次の三種類のものがある．

1　特定の分野へのオントロジーを提供するもの

ある対象領域を構成する概念を抽出して整理・体系化し，アプリケーションのなかで使用することができるような定義された語彙として提供している研究がある．企業活動を記述するビジネスプロセスモデリングやエンタープライズモデリングの分野では，ビジネスに関連する諸概念の体系化を図ったエンタープライズオントロジーの構築が行われており，それを利用したビジネスプロセスリエンジニアリング支援の研究がされている[13].

2　オントロジー記述のフレームワークを提供するもの

多くの場合，オントロジーで記述されたデータは情報システムで使用される．近年，情報システム上のデータ記述形式として XML[16]が標準になりつつある．XML は一般の木構造データであるので，これに構文制約をつけてオントロジー記述のフレームワークとして提案している研究がある[11].

3　概念階層における包含関係の規定を与えるもの

概念階層における概念間の包含関係（上下関係）は，その概念階層を作成した人やコミュニティにおける概念化の合意事項に基づいている．しかし，その合意は，他人や他のコミュニティとは必ずしも一致しない．そこで，概念の包含関係を規定する共通の理論提供を目的とした研究が成されている[15]．そこでは全ての概念をいくつかの性質で分類し，その性質が従うべき関係に基づいて概念階層を作成することが提案されている．

4.2

本研究ではソリューションメニュー記述のためのオントロジー（ソリューションオントロジーと呼ぶ）を開発する．ここでは，前節で述べたオントロジー工学の各側面からソリューションオントロジーの機能を検討する．

A. 対象とする分野

ソリューションオントロジーは，任意のソリューションメニュー体系をモデル化するための方法を与えるものである．ソリューションビジネスの対象は一般のビジネスであるが，ソリューションオントロジーはビジネスの具体的な構成物や手順の概念集合を与えるものではなく，この点がエンタープライズオントロジーとは異なる．

B. 記述のフレームワーク

第 3 章で述べたように，ソリューションメニュー体系は

1. 対象とするソリューションビジネスの分野に存在するものの記述
2. 顧客に提供する効用の概念化としてのメニューの記述
3. 実際のプロジェクトの記述およびそれらとメニューとの対応

を表現する必要がある．このうち，1. はドメインオントロジーのうちの，概念化の方法の部分に相当する．2. は問題解決の概念化でありタスクオントロジーであるとも言えるが，具体的な手続きを表すものではない．提供する効用の概念化をうまく表現できる枠組みが必要である．3. の実際のプロジェクトは 1. を使って記述されたドメインの概念で表せると考えられるが，それらと 2. で記述されるメニューとの関係を表現することも考慮しなければならない．

C. 階層構造

ソリューションメニュー体系の階層構造は，ソリューションビジネスとして意味のある規約に基づいて構成する必要がある．

上記の点の詳細は第 6 章で検討する．

5

5.1

本研究では，ソリューションメニューの具体例として，ソフトウェア開発組織のデザインパターン[18]を使用する．

デザインパターンとは，建築の分野で「住み心地」など定量化しにくい設計特性を記述するために提唱された「パターンランゲージ」[16]の考え方をソフトウェア工学に適用したものであり，ソフトウェアの設計で繰返し現れる課題を明らかにし，それに対する解答をパターンという形式で記述したものである[17].

ソフトウェア開発組織のデザインパターンは，さらにこれを組織形態の記述に応用し，いくつかのソフトウェア開発プロジェクトのケーススタディを元に，効果的なソフトウェア開発組織が満たすべき性質をデザインパターンの形にまとめたものである．

一つのパターンは次の 3 つの部分で構成される．

Problem：解決すべき課題

Context：課題が生じる状況

Solution：課題に対する解決策

それぞれの部分は自然言語で記述されており，その具体的な記述の仕方にはいろいろな方法がある．ソフトウェア開発組織のデザインパターンでは，各パターンの粒度と具体性のレベルはさまざまである

5.2

本節では，デザインパターンとソリューションメニュー体系の満たすべき要件との比較をおこない，デザインパターンをソリューションメニューとして使用することの検討をする．

1　対象とする分野に存在するものの記述

デザインパターンには対象分野に存在するものの概念を体系的に記述するパートは無い．それぞれのパターンの中で部分的に言葉の定義がおこなわれる場合もあるが，多くはそのパターンを使用するエンジニアなどのコミュニティで暗黙的に合意されているものである．ソリューションメニューを作成するときにはドメインオントロジーで規定された概念化の方法にしたがって概念を抽出する必要がある．

2　顧客に提供する効用の概念化としてのメニューの記述

一つのパターンはある問題に対する解決策を記述したものであるが，自然言語で記述されているのである程度の曖昧さを持つ．また，粒度や具体性もさまざまである．ソリューションメニューは全ての対象に対して完全に同一のソリューションを提供するものではなく，ある程度の多様性を含む概念化であったので，デザインパターンのこの性質はソリューションメニューと合致する．しかし，デザインパターンは効用の概念化を直接記述したものではなく，あくまでも解決策である．さらにその解決策は，対象の取るべき構造など静的側面を示したものと，解決の手順など動的側面を示したものの両方がある．ソリューションメニューを作成するときには，これらが意味している効用を抽出する必要がある．

3　実際のプロジェクトの記述およびそれらとメニューとの対応

デザインパターンはソリューションに相当する部分のみを記述したものであり，実際の各プロジェクトの記述をする機構は備えていない．ソリューションメニューを作成するときには，各パターンと実際のプロジェクトとの関係が反映されるような記述方式を考慮することが必要である．

以上より，適切なオントロジーを用いれば，デザインパターンをソリューションメニュー体系に変換できると考えられる．具体的変換方法は第 6 章で検討する．本研究ではソフトウェア開発組織のデザインパターンからソリューションメニュー体系を構成し，提案するメニュー開発支援システムの検証とする．

6

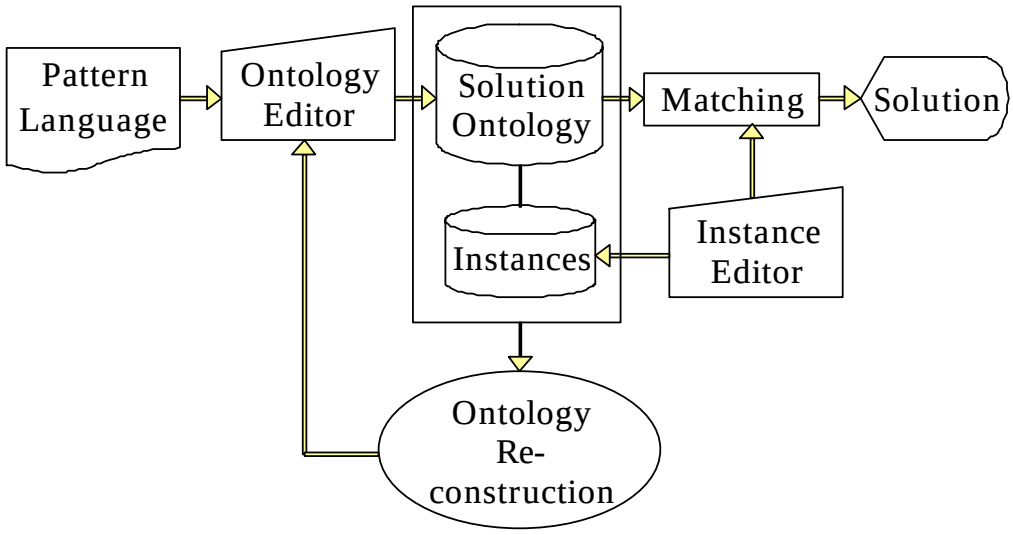
6.1

3.3節で考察した，ソリューションメニュー体系の備えるべき性質に基づいた，想定するシステム全体のソリューションビジネスでの使用イメージを図 6-1に示す．ソリューションメニュー体系は，対象分野に対するケーススタディやパターンランゲージなどをもとにソリューションオントロジーとして記述されている．ソリューションオントロジーは対象分野を記述するドメインオントロジーと，それらに対するメニューオントロジーから構成される．ユーザーはドメインオントロジーを元に実際にビジネスの対象とする顧客などの記述をインスタンスとして行う．ソリューションビジネスの支援として知識ベースシステムとして使用する場合には，インスタンスとメニューオントロジーとのパターンマッチにより，インスタンスで記述された対象に適用すべきソリューションメニューを選択して提示する．ソリューションメニューを適用されたインスタンスはデータベースに蓄積され，これらとソリューションオントロジーとを使って，メニュー体系の再構築に利用する．

なお，図 6-1のシステムでオントロジーと入力されたインスタンスのマッチングによって解答を与えるという部分は，4.1節で述べた BPR 支援システムなど，オントロジーを知識ベースの記述方法として使用する先行研究において類似のシステムが試作されている．しかしそれらのシステムはソリューションビジネスでの使用を考慮したものではないため，そこで使用されているオントロジーは本研究で提案するオントロジーの形式とは異なるものである．

後述するように，先行研究で提案されているオントロジー記述形式ではソリューシ

ョンメニューとしての望ましい性質を備えた記述を行いにくいため，ソリューション支援システムで使用するには不充分である．そこで以下の節ではソリューションメニューをオントロジーとして記述するための方法論を設計し，ソフトウェア開発組織のデザインパターンに適用し，その効用を考察する．



6 - 1

6.2

6.2.1 既存のオントロジー記述形式

ソリューションオントロジーは，情報システムでの扱いやすさを考え，XML で記述する．近年のオントロジー記述形式の研究では XML での利用を考慮したものが多い．ここではそれらの記述形式のソリューションオントロジーへの適用可能性を検討する．

OIL[11]はヨーロッパを中心とする大学と企業の研究者からなる On-To-

Knowledge プロジェクトが提案しているものである．フレームベースの記述言語であり，Description Logic による制約条件を付加できる．図 6-2に OIL による記述の例を示す．この例は XML ではないが，木構造であるのでほぼ同一に扱える．OIL はクラスの階層構造と制約条件を有するため，外部の推論エンジンによる整合性チェックを行えるという特徴があり，大規模知識ベースの記述フレームワークへの適用事例などが研究されている．

```
slot-def eats
slot-def is-part-of
class-def animal
class-def plant
        subclass-of NOT animal
class-def tree
        subclass-of plant
class-def branch
        slot-constraint is-part-of
                has-value tree
class-def defined carnivore
        subclass-of animal
        slot-constraint eats
                value-type animal
class-def defined herbivore
        subclass-of animal, NOT carnivore
        slot-constraint eats
                value-type plant OR (slot-constraint is-part-of has-value plant)
```

6 - 2 OIL

OIL をソリューションメニューに適用すると，図 6-3のような記述が考えられる．これは3.2節で述べた富士通の SALESFORCEVISION の一部を記述したものである．SALESFORCEVISION というメニューのサブメニューとしてチームセリングがあり，チームセリングは顧客分析というニーズに対して SymfoWARE というコンポーネントを使用してソリューションを提供するものである，というメニューオントロジーが記述できている．

しかし，この形式ではメニューオントロジーとインスタンスの関係が記述できない．例えばある顧客に対してチームセリングソリューションを提供した，あるいは，その結果どうなった，というような事項を反映することができない．そのため，3.3節で

あげたソリューションメニュー体系としての要求事項を満たすことができず，このままでは記述形式として不充分である．

```
slot-def needed-for
slot-def has-component
class-def needs
class-def component
class-def 顧客分析
        subclass-of needs
class-def SymfoWARE
        subclass-of componet
class-def SALESFORCEVISION
class-def チームセリング
        subclass-of SALESFORCEVISION
        slot-constraint needed-for
                has-value 顧客分析
        slot-constraint has-component
                has-value SymfoWARE
```

6 - 3 OIL

6.2.2 支援システム中でのオントロジー記述形式の要件

さて，ソリューションメニューの性質から要請される記述形式の要件のほかに，図6-1のような支援システムの中で使用することから必要となる機能もある．

システムにおいては，オントロジーの中身が変わってもその操作プログラムは同一である必要ある．このシステムは，パターンマッチによりインスタンスに適合するソリューションを提示するだけではなく，オントロジーの再構成にも利用することを考慮している．そこで，ソリューションオントロジーによってインスタンスに対して何らかの操作を加えることが必要となるが，この操作の情報をプログラムの中に書きこんでおくことは望ましくない．なぜならば，ソリューションメニュー体系が改善されると，インスタンスに対する具体的な操作は変更される可能性があり，これがプログラム中に記述されていると変化に対応できないからである．そのため，ソリューションオントロジーの記述形式は，その操作プログラムとは独立に内容の変更が可能な形式を備えていなければならない．

6.2.3　ソリューションメニューのオントロジー構造の設計

このように，既存の形式ではソリューションメニューの性質を反映した構造を持つオントロジーの記述には不充分なので，本研究では独自の記述形式を設計する．

第 3 章で見たように，ソリューションメニューはイノベーションの理論におけるシステム観を表すものであった．すなわち，そのメニューに対応するシステムやサービスで何をするのか，を概念化したものである．これを，メニューを適用する対象と，それに対する操作に分解すると，ソリューションメニューは対象とする分野に存在するものを望ましい状態に変化させる操作の体系であると見なせる．例えば，「営業活動」の状態が「効率的でない」という問題がありその状態を「営業支援システム」を構築することで「効率が良い」にかえるという操作が Sales Force Automation に関するソリューションである．

この考察に基づき，本研究ではソリューションメニューを，対象とする分野に存在するものを記述するドメインオントロジーと，対象の状態変化を記述するメニューオントロジーのセットととして構成する．ドメインオントロジーの各概念は複数の状態変数を持つもの，メニューオントロジーはそれらのドメインオントロジーのうち値を変更する状態の始値と終値を指定するもの，と定義する．

Sales Force Automation の例では，図 6-4のようなオントロジーとなる．また，これに対するインスタンスの例は図 6-5のようになる．このインスタンスは図 6-4のメニューオントロジー SFA の初期状態である「営業活動 効率＝悪い」にマッチするので，これを適用すると，インスタンスの状態が「営業活動 効率＝良い」，「**営業支援システム 構築＝Yes」に変更される．**

このようにオントロジーを定義することで，ソリューションメニューとしての要件を備えて，かつソリューション支援システムで使いやすい記述が可能となる．

ドメインオントロジー：組織
　　状態：名称
ドメインオントロジー：営業活動
　　状態：効率
ドメインオントロジー：営業支援システム
　　状態：構築
メニューオントロジー：SFA
　　初期状態
　　　　営業活動
　　　　　　状態：効率＝悪い
　　結果状態
　　　　営業支援システム
　　　　　　状態：構築＝Yes
　　　　営業活動
　　　　　　状態：効率＝良い

6-4 Sales Force Automation

組織
　　状態：名称＝○○営業部
営業活動
　　状態：効率＝悪い
営業支援システム
　　状態：構築＝No

6-5 Sales Force Automation

6.2.4 XML スキーマの設計

前節で定義したドメインオントロジーとメニューオントロジーは XML[16]で記述する．XML は木構造のデータ構造を持つが，それだけでは汎用的にすぎるので，構文を制約するための機構としていくつかのスキーマが提案されている[20]．本研究では XML スキーマとして RELAX[21]を採用する．

図 6-6にドメインオントロジーの記述形式を規定するスキーマを示す．ドメインオントロジーの最上位ノードを Entity で統一し，その type として実際の概念を記述する．また，状態変数は Entity の下位ノード attr とする．これによりプログラム側の処理手順を Entity ノードに対する操作という形で統一できる．図 6-6のスキーマに適合する XML 記述の例を図 6-7に示す．

```
<tag name="Entity">
  <attribute name="type" required="true" type="ID"/>
</tag>
<elementRule label="Entity" role="Entity">
  <ref label="attr" occurs="*"/>
</elementRule>

<tag name="attr">
  <attribute name="name" required="true" type="string"/>
</tag>
<elementRule label="attr" role="attr" type="string"/>
```

6 - 6

```
<Entity type="organization">
        <attr name="name">
                sample
        </attr>
</Entity>
```

6 - 7 XML

```
<tag name="Pattern">
  <attribute name="name" required="true" type="string"/>
</tag>
<elementRule label="Pattern" role="Pattern">
  <choice occurs="*">
    <ref label="InitState"/>
    <ref label="ResultState"/>
  </choice>
</elementRule>

<tag name="InitState">
  <attribute name="entity" required="true" type="string"/>
  <attribute name="attr" required="true" type="string"/>
</tag>
<elementRule label="InitState" role="InitState" type="string"/>

<tag name="ResultState">
  <attribute name="entity" required="true" type="string"/>
  <attribute name="attr" required="true" type="string"/>
</tag>
<elementRule label="ResultState" role="ResultState" type="string"/>
```

6-8

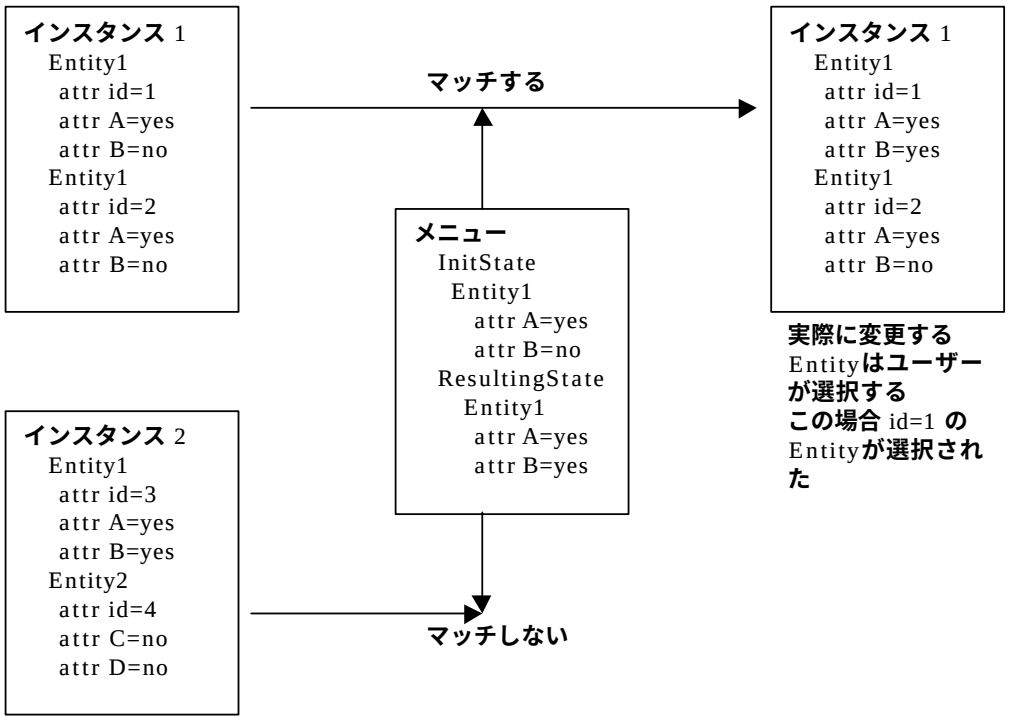
メニューオントロジーの記述形式を規定するスキーマを図 6-8に示す．ドメインオントロジーと同様に，上位ノードは統一し，そのアトリビュートとして実際のメニューを記述している．これによりオントロジーの中身が変わっても同一のプログラムで操作することができる．図 6-9に XML の記述例を示す．

```
<Pattern name="sampleMenu">
<InitState entity="aaa" attr="bbb">ccc</InitState>
<ResultState entity="foo" attr="bar">sss</ResultState>
</Pattern>
```

6-9 XML

6.2.5 ソリューションメニューによるインスタンス操作

図 6-10にメニューオントロジーとして記述したソリューションメニューによるインスタンス操作の概念を示す．インスタンスは各状態変数（attr）の値をセットされた Entity の組で構成される．メニューオントロジーは状態値の始値から終値への変更操作として定義されている．メニューオントロジーの始値とマッチする状態値を持つインスタンスを終値に変更する．このとき，複数のオントロジーがマッチする場合にはユーザーに問合せをして，ユーザーが選択したものの状態値を変更する．このことは，そのソリューションメニューの実現の仕方に多様性が存在することに対応する．



6 - 10

6.3

6.3.1 Fromal Ontology of Properties

概念の階層構造構築の方法論として, Formal Ontology of Properties による方法が提案されている[15]．そこでは哲学的考察に基づき，概念の持つ性質をいくつかに分類し，その性質によって概念の階層構造の決定方法に規則性を持たせている．例えば, x がある概念φの実例であることをφ(x)と書くと，φが rigid である，とは次のように定義される．

$\forall x\, \phi(x) \rightarrow \Box\phi(x)$

この定義に従うと（一般的な意味で）PERSON は rigid であるが STUDENT は non-rigid である．なぜなら，ある x が PERSON の実例であるならば，x はすべての可能世界で PERSON であると考えられるが，x が STUDENT の実例であっても，卒業などにより，そうでなくなる可能性があるからである．また，CATERPILLAR や BUTTERFLY などは non-rigid である．なぜなら，それらの実例は成長により前者から後者に変化するからである．このように，rigid という性質は時間変化との関係が強いものとして位置付けられている．そして，rigid な概念は non-rigid な概念の上位に位置するべき（少なくとも non-rigid が rigid の上位になることはない）とされている．

6.3.2 ソリューションメニュー体系における Rigidity の意味

さて，ソリューションメニュー体系において，ある概念が rigid であるという性質が持つ意味は，その概念の実例が他の概念の実例とならないことである．例えば，富士通の SALESFORCEVISION では，メニューを構成するコンポーネントとして SynfoWAREServer があり，そのサブメニューとして商談管理サーバー（チームセリングで使用）と分析サーバー（OneToOne マーケティングで使用）がある．あるサーバーハードウェア（GRANPOWER）はそのどちらとしても使い得る．よって商談管理サーバーおよび分析サーバーは non-rigid である．

3.2節で述べたように，イノベーションの理論に基づく考察によれば，メニュー体系はあるソリューションビジネスの概念化であった．そして，ある体系の中のあるメ

ニューは，システム観の概念化であった．ここでシステム観の分化が生じると，ある実例に対して分化した両方の概念が対応する．このとき，それらの概念は non-rigid である，と言える．このように，ソリューションメニューにおいては rigid の概念は，時間変化よりむしろシステム観の違いに深く関係している．このため，rigid な概念が non-rigid な概念の上位に位置するべきであるという主張はソリューションメニューにおいても有効である．

6.3.3 オントロジー再構成支援

Formal Ontology of Properties の rigid という性質を用いて, ソリューションオントロジーの再構成を支援することを考える．

前述のように，ソリューションメニュー体系においては，rigid な概念は non-rigid な概念の上位であるべきである．逆に，ある複数の概念が non-rigid であり，それらの実例集合が共通部分を持つとき，それらの概念を下位にもつ rigid な概念が存在することが示唆される. 先にあげた SALESFORCEVISION の例で考えると次のようになる．仮に現在のメニュー体系では，チームセリングで使用する商談管理サーバーと OneToOne マーケティングで使用する分析サーバーを包含する上位の概念が無かったとする．このとき，プロジェクトをいくつか遂行していく中で，これら 2 種類のサーバーを実際には一つのものとして構築して顧客に提供した事例が蓄積されていく．すると, これらの概念の実例集合は共通部分を持つことになるので non-rigid であり，上位の rigid な概念（SymfoWAREServer）が存在することが示唆される．

そこで，あるソリューションオントロジーにおいて，以下に示すような場合にはエンティティ E と F が non-rigid であり, それらの上位のエンティティが存在する可能性があるというサジェスチョンをユーザーに与える．

ドメインオントロジーを$D = \{E_1, E_2, \ldots, E_d\}$とする． $E, F \in D$である．

あるエンティティ$G \in D$が存在し，現在蓄積されているインスタンス全体のうち，次の性質 1.〜2.を満たすインスタンスの割合があらかじめ定めた R より大きいとき，E と F は non-rigid であり，それらの上位エンティティが存在する可能性がある．

1. インスタンスを構成するエンティティの組$(e_1, e_2, \ldots, e_m)$, $e_i \in D$の中にGがある．すなわち，$\exists k, 1 \le k \le m, e_k = G$である．
2. e_kを構成する状態の組を$(s_1, s_2, \ldots, s_e)$とする．Gのある状態変数Sについて，$\exists p, q$（$1 \le p, q \le r$ and $s_p = S$ and $s_q = S$ and s_pの値がE and s_qの値がF）

6.4

—ソフトウェア開発組織のデザインパターンに基づくソリューションメニュー記述—

6.4.1 ソフトウェア開発組織のデザインパターンを使用する意義

設計したソリューションオントロジーの有効性を検証するために，具体例としてソフトウェア開発組織のデザインパターン[18]を用いて，ソリューションメニュー体系の記述をおこなった．第 5 章で述べたように，このデザインパターンはいくつかのケーススタディを元にして作成されたものであり，効果的なソフトウェア開発組織を構成するための方法が 38 個のパターンで記述されている．これは，もともとはソリューションビジネスの中で使用することを想定されたものではない．しかしソフトウェア開発組織の構成という一つのまとまった領域に対する問題解決の方法が体系的に記述されているので，ここからソリューションメニューを作成することは可能であり，提案したオントロジーの記述能力の検討材料として有用である．また，デザインパターンは自然言語で記述されているが，それぞれのパターンは Problem, Context, Solution の組で記述するという制約がある．これは，ある領域での問題解決方法の概念化の汎用的な枠組みを定めたものと考えることができる．提案したソリューションオントロジーはソリューションメニューとしての概念化の枠組み与えるために，デザインパターンよりも多くの制約を加えている．このため，デザインパターンを中間表現として利用し，それからソリューションメニューを記述するという手順をふめば，

作成が容易になると考えられる．

6.4.2 ソリューションメニュー記述の方法

設計したオントロジーにしたがって，ソフトウェア開発組織のデザインパターンからソリューションメニューを次のように作成する．

(1) ドメインオントロジーの抽出

各パターンから状態変数をもつエンティティとしてドメインオントロジーを作成する．これらはメニューオントロジーの記述に使用する．また，実際のソフトウェア開発組織をあらわすインスタンスを構成するために必要な要素でもある．デザインパターンではドメインオントロジーは明示されていないので，記述の中から抽出する必要がある．その際に次の方針を採る．

・ソフトウェア開発組織を構成する基本要素となるもの

それぞれのパターンから，ソフトウェア開発における特定の役割など，組織構成の記述をおこなう際の語彙となる概念を抽出してエンティティとする．一般にパターンの中ではこれらのエンティティの状態は明示されていないが，「有る」（あるいは設定されている，確立されている）か「無い」という値を取るものとする．

・組織の状態を記述するための要素となるもの

それぞれのパターンから，組織の状態を記述する際の語彙となる概念を抽出してエンティティとする．これらのエンティティは主にパターン中の Problem と Context に，「A が B（という状態）である」という形で現れる．

実際には，一つのエンティティがこれらの両方の性質を備えている場合が多い．

(2) メニューオントロジーの記述

それぞれのパターンで記述されている問題解決方法を，いくつかのドメインオントロジーの状態値を変化させる動作としてとらえる．その初期状態と終了状態の組としてメニューオントロジーを記述する．

・Problem に記述されている事項は，あるエンティティの変化させるべき状態としてとらえ，初期状態にはその状態を記述する．終了状態には変化させたあとの状態を記述する．ただし，デザインパターンでは問題解決後の記述はされていないことも

あるので，その場合には適宜補う必要がある．

・Context に記述されている事項は，あるエンティティの状態としてとらえ，初期状態に記述する．

・Solution として，「A を作成せよ／使用せよ」のように指示されているものは，初期状態では A というエンティティが「無い／使用されていない」という状態であり，終了状態ではそれが「有る／使用されている」という状態である，とする．また「A を B の役割に割当てよ」のように複合的に指示されているものは，初期状態では A というエンティティの役割という状態の値に B がセットされてなく，かつ B というエンティティが「割り当てられていない」という状態であるとし，終了状態では A の役割という状態の値が B で，かつ B が「割り当てられている」という状態であるとする．

これらとは別に，次の点に注意する．デザインパターンの記述では，ある状態値であることが明示されてはいないが暗黙に仮定されている場合がある．特に Problem や Context に記述が無くても, Solution の中であるものがその状態にあることを前提として解決方法を記述している場合がある．そのときには初期状態でそれらを記述することを忘れないようにする必要がある．また実際にはそれぞれのエンティティは複数のパターンにまたがって出現するので，一つのエンティティが複数の状態変数を持ち，それらが複数のパターンで別々に記述されていることがほとんどである．

6.4.3 オントロジーによるソリューションメニュー記述の結果

1 パターンからソリューションメニューへの変換

図 6-11にソフトウェア開発組織のデザインパターンの一つを示す．これを元に，提案するオントロジーにしたがって記述したものを図 6-12に示す．また，巻末にソリューションメニューのドメインオントロジーと, 全 38 パターンのうち 24 個のメニューオントロジーを示す．また 14 パターンについてはデザインパターンの記述を示す．

Pattern name : Firewalls
Problem : Project implementors are often distracted by outsiders who feel a need to offer input and criticism.
Context : An organization of developers has formed, in a corporate or social context where they are scrutinized by peers, funders, customers, and other “outsiders.”
Solution : Create a Firewall role, who shields other development personnel from interaction with external roles. The responsibility of this role is “to keep the pests away.”

6-11 Firewalls

「Firewalls」というパターンを，前述の方法にしたがって次のようにメニューに変換している．

(1) Problem の記述から次のように変換する

初期状態：エンティティ organization の状態 distracted の値が yes である

終了状態：エンティティ organization の状態 distracted の値が no である

(2) Context の記述から次のように変換する

初期状態：エンティティ organization の状態 formed の値が yes である

(3) Solution の記述から次のように変換する

初期状態：エンティティ Firewall の状態 staffed の値が no である

終了状態：エンティティ Firewall の状態 staffed の値が yes である

エンティティ person の状態 role の値が Firewall である

```
<Pattern name="Firewalls">
<InitState entity="organization" attr="formed">yes</InitState>
<InitState entity="organization" attr="distracted">yes</InitState>
<InitState entity="Firewall" attr="staffed">no</InitState>
<ResultState entity="Firewall" attr="staffed">yes</ResultState>
<ResultState entity="person" attr="role">Firewall</ResultState>
<ResultState entity="organization" attr="distracted" >no</ResultState>
</Pattern>
```

6-12 Firewalls

この変換によって，Firewall というメニューオントロジーを,「organization が形成されており（formed）外部からの雑音がある（distracted)」状態を「Firewall の役割を持つ person があり，外部からの雑音が無い」状態に変化させる，という操作として記述している.

さて，あるインスタンスを構成するエンティティ organization が formed = yes および distracted = no の状態を持つとき，これに対してメニューFirewalls を適用すると, そのインスタンスのエンティティ person の状態 role の値に Firewall がセットされ，organization の distracted の値が no に変更される．ここで，あるインスタンスは実際のソフトウェア開発組織を示すので，一般に一つのインスタンスには複数の person が属する．よってシステム実行時には，どの person に対して role の値を Firewall にセットするかがユーザーに問われ, その選択にしたがってインスタンスが変更される．このことは実際の組織で，誰が Firewall の役に就くかという選択に相当する.

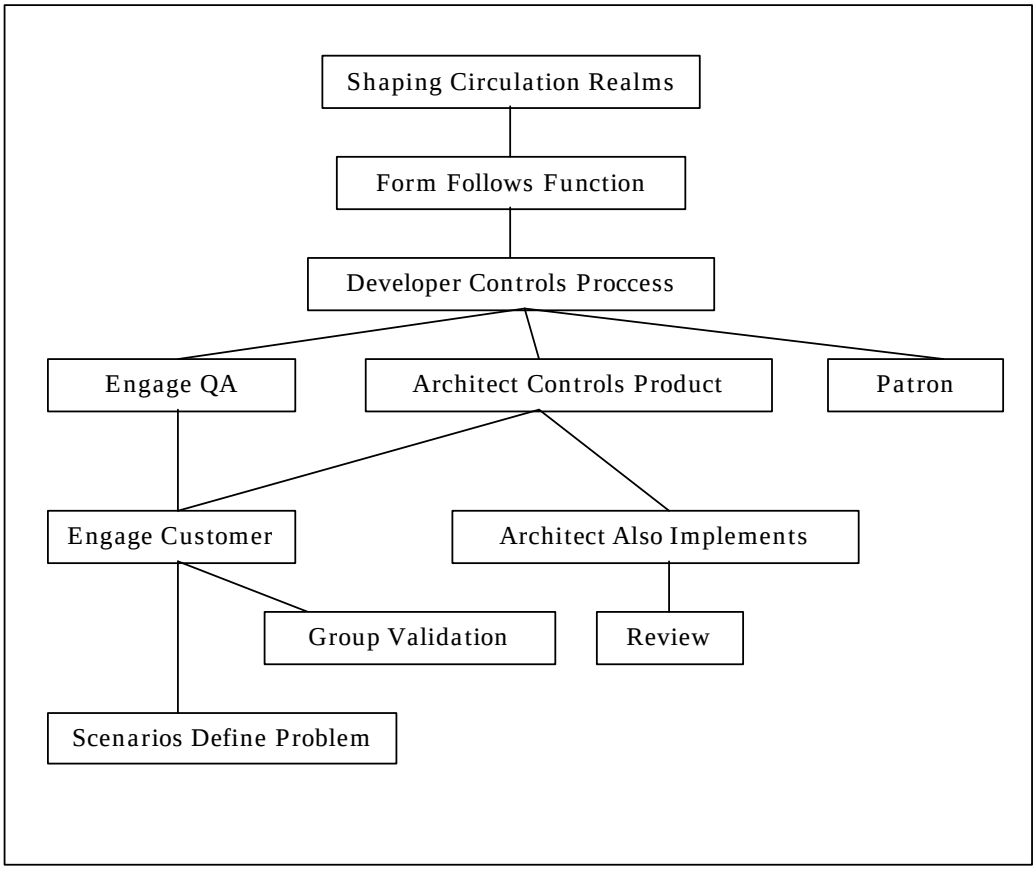
ドメインオントロジーの例として，organization の記述を図 6-13に示す．ドメインオントロジーはすべてのメニューオントロジーに現れる状態 attr を持つように記述する.

```
<Entity type="organization">
        <attr name="size"/>
        <attr name="geographically_distributed"/>
        <attr name="architectual_partitioning"/>
        <attr name="reflect_market"/>
        <attr name="roles"/>
        <attr name="technical_direction"/>
        <attr name="compatible_with_architecture"/>
        <attr name="formed"/>
        <attr name="distracted"/>
        <attr name="communication"/>
        <attr name="interaction"/>
        <attr name="coupling"/>
        <attr name="people" type="Integer"/>
        <attr name="social_network"/>
        <attr name="condition"/>
</Entity>
```

6 - 13

2　ソリューションメニューとしての体系化

各メニュー間の関係は，初期状態の値と終了状態の値との関係で表現することができる．すなわち，あるメニューA の終了状態となっている状態値を初期状態に持つメニューB は，A よりも後に適用されなければならない．この規則に基づき，メニューが体系化される．作成したソフトウェア開発組織のメニュー体系の一部を図 6-14に示す．このような体系化によって，ソリューションビジネスを遂行する効率が向上すると期待される．



6 - 14

3　Rigidity によるオントロジー再構成支援の例

ソフトウェア開発組織における役割として，Firewall と Gatekeeper という概念があらわれる．これらは別の種類の役割として記述されているが,（明示されてはいないが）一人が兼務しても良い．もし，ソフトウェア開発組織に対するメニュー体系を作った当初は Firewall と Gatekeeper を兼務している事例が無かったが, その後いくつかの組織ではこれらを兼務していることが認識されるとすると，これら概念は non-rigid であり, 上位概念の存在が示唆される. その示唆に基づき, 例えば Manager という役割を設定することができる.

具体的には6.3節で定義した条件によって，次のように判定する.

エンティティ G として *person* をとる．現在蓄積されているインスタンス全体のうち，次の性質 1.〜2.を満たすインスタンスの割合があらかじめ定めた R より大きいとき，*Firewall* と *Gatekeeper* は non-rigid であり，それらの上位エンティティが存在する可能性がある.

1. インスタンスを構成するエンティティの組$(e_1, e_2, \ldots, e_m)$, $e_i \in D$の中に *person* がある．すなわち，$\exists k, 1 \le k \le m, e_k =$ *person* である.
2. e_kを構成する状態の組を$(s_1, s_2, \ldots, s_e)$ とする．*person* の状態変数 *role* について，$\exists p, q$（$1 \le p, q \le r$ and $s_p =$ *person* and $s_q =$ *person* and s_p の値が *Firewall* and s_q の値が *Gatekeeper*）

6.5

ソフトウェア開発組織のデザインパターンからソリューションメニュー体系を作成した結果から，本研究で開発したソリューションオントロジーに対して次のような評価をした.

1　ドメインオントロジー

(1) 性質の違う概念を統一的に記述することができる

ソフトウェア開発組織のデザインパターンは「組織」など抽象度の高い概念から，ソフトウェア開発の具体的な手法である「CRC」など具体性のあるものまで，いろいろな性質の概念を含んでいる．状態を持つエンティティというドメインオントロジーを使用することにより，これらの概念を同一の形式で記述することができた．例えば「組織」という単語は多くのパターンに出現するが，そこでは組織の効率の良し悪しなど抽象的な概念で問題点やその解決方法が述べられている．そこで「組織」というエンティティは「効率」という状態変数を持ち，その値は「良い／悪い」であるという記述をおこなった．一方，CRC については，あるパターンで問題解決の方法として「CRC を使用せよ」という記述がされている．そこで「CRC」というエンティティは「使用」という状態を持ち，その値は「されている／されていない」であるという記述をおこなった．

一般のソリューションメニューにおいても，ソフトウェア開発組織のデザインパターンと同様に，組織活動の抽象的な概念からハードウェアなど具体的なものまでを含むので，本研究で提案したドメインオントロジーのこの性質は有用である．

(2) メニューの記述を容易におこなえる

(1)で述べたようにソフトウェア開発組織のデザインパターンは性質の大きく違う概念を含んでおり，そのため，それぞれのパターンで問題解決方法の記述にも大きな違いがある．次表にいくつかの例を示す．

6-1

パターン名	問題（Problem）	解決方法（Solution）
Size the Organization	組織の大きさはどのくらいにすれば良いか	25KSLOC 以上の開発では原則として 10 人を選べ
Architect Controls Product	プロダクトに elegance と cohesiveness が欠けている	Architect を設置せよ
Group Validation	製品の品質を保証すること	CRC カードを使用せよ

このように，具体的な数値や手法を与えるものも，elegance や cohesiveness のような抽象的な概念で記述されているものもある．一般のソリューションメニューにおいても，抽象的な記述でソリューションを記述し，その具体的な適用の方法については運用者に任されている場合が多い．本研究で提案したメニューオントロジーではこ

れらをエンティティの状態の変更という統一的な枠組みで記述することができた.

(3) 概念抽出の基準を与えることができる

デザインパターンの自然言語の記述の中から，どの概念を抽出して使用するかという問題がある．本研究ではメニューオントロジーをエンティティに対する状態変化の操作として定義したことに従い，デザインパターンから概念を抽出する方法を，前節で述べたように

・Problem に記述されている事項は, あるエンティティの変化させるべき状態としてとらえ，初期状態にはその状態を，終了状態には変化させたあとの状態を記述する.
・Context に記述されている事項は，あるエンティティの状態としてとらえ，初期状態に記述する.
・Solution として,「A を作成せよ」のように指示されているものは，初期状態では A というエンティティが「無い」という状態であり，終了状態ではそれが「有る」という状態である，とする.

と定めることができた．これにより，記述されている概念をどのように使用すべきかという判断を，一定の基準にしたがっておこなうことができた．これは，提案したオントロジーがソリューションメニューとしての性質を持つように構造化されていることの利点といえる.

(4) ソフトウェア開発組織の記述をインスタンスとしておこなうことができる

ドメインオントロジーで定義されたエンティティの各状態値を設定したインスタンスで，このデザインパターンを適用する対象である，実際のソフトウェア開発組織の状態を記述することができる．また，これにメニューオントロジーで定義された各パターンの操作を適用することができる．そのインスタンスにどのパターンが適用できるかは，メニューオントロジーの初期状態として定義された状態値と，そのインスタンスの状態値が一致するかどうかで判断できる．インスタンスにパターンを適用してある問題を解決したという状況は，そのインスタンスの状態値をメニューオントロジーの終了状態として定義された状態値に変更することで表現することができる.

(5) 記述能力には制限がある

本研究で提案したメニューオントロジーでは，インスタンスの状態値と，メニューの初期状態の状態値の一致条件は，メニューの初期状態で記述されている状態全ての

値がインスタンスの対応する状態の値と一致することである．よって，複数の条件を列挙しておき，そのいずれかにマッチするインスタンスに適用するようなメニューは記述できない．メニューオントロジーにより柔軟な記述能力を持たせるためには，この点の改良が必要である．

また，現在のドメインオントロジーの定義では，エンティティの状態値として取り得るデータの型が指定できず，全て文字列として扱われる．しかし実際の値としては論理値や数値など他のデータ型として扱うべきものもある．例えば前節で述べた Firewall パターンの記述では，person というエンティティの状態 role の値として「Firewall」を入れていた．これは実際には Firewall という別のエンティティを指していると解釈されるので, 状態 rolo の値のデータ型はエンティティであると定義しておく方が自然である．この点も改良が必要である．

7

7.1

本研究では、ソリューションビジネスの特徴とソリューションメニューの役割を明確にし、それを反映したメニュー記述形式とメニュー再構成支援方法を開発した.

近年，ソリューションビジネスが注目され，その重要性は増しているが，しかしソリューションビジネスとはなんであるのか，ソリューションビジネスを効果的に支援するにはどのようにすればよいのか，という問題に対する研究は十分でなかった．そこで本研究では，まずソリューションビジネスの特徴とソリューションメニューの役割を明確にした.

ソリューションビジネスの特徴を明確にするために，先行研究を元に情報産業および社会インフラ産業の特徴を次のようにまとめた.

1. それぞれの顧客に共通のハードウェアや基幹ソフトなどのプラットフォームを用いたシステムを提供する
2. それぞれの顧客のニーズは異なるものであり，プラットフォームが提供する機能とユーザーが求める効用との間に乖離がある

この考察をもとに，基礎となるプラットフォームがあり，それらを使用して個々の顧客の異なるニーズに適合したものを提供するビジネスがソリューションビジネスである，と定義した.

次に，製品開発メカニズムのイノベーション研究の一つである「製品システム観」の理論を参考にして，ソリューションビジネスにおけるソリューションメニューは，そのソリューションでどのような効用をどのように提供するのか，を概念化したもの

であると定義した．この定義をもとに，ソリューションメニューの役割を次のように明確にした．

1. ソリューションメニューは，ソリューションを構成するサービスやコンポーネントの機能開発の指針を与えるものである．
2. 顧客に提供する実際のソリューションプロジェクトを通した学習により獲得された知識がソリューションメニューとして形式化される．

ここまでの考察から，ソリューションメニューの記述形式には次の 3 点が必要であることがわかった．

A. ソリューション対象分野に存在するものの記述
B. 提供する効用の概念化としてのメニューの記述
C. 各プロジェクトの記述およびそれらとメニューとの対応

次に，これらの結果を元に，オントロジーに基づくメニュー記述形式とメニュー再構成支援方法を開発した．既存のオントロジーでは上記 A～C 全てに対応した記述には不充分であったので，新たにソリューションメニュー記述のためのオントロジーを開発した．オントロジーが備えるべき機能である，概念化の枠組み，記述形式，概念階層のそれぞれについて，次のような設計をおこなった．

1. 対象を概念化するための枠組み
   ・ドメインオントロジー：状態変数を持つエンティティの定義（上記 A に対応）
   ・メニューオントロジー：エンティティの状態値を変化させる操作の定義（同 B）
   ・インスタンス：状態変数に値がセットされたエンティティの集合（同 C）
2. 記述形式の定義：XML スキーマ言語で定義
3. 知識体系化のための概念階層の定義
   ・ドメインオントロジー：エンティティの rigidity による関係
   ・メニューオントロジー：状態変数の変化の順序による関係

開発したオントロジーの有効性を確認するために，ソフトウェア開発組織のデザインパターンからソリューションメニュー体系を作成した．その結果，メニューの階層構造をと rigidity に基づくオントロジー再構成の例を示すことができた．

ソフトウェア開発組織のソリューションメニュー記述結果の考察から，開発したオントロジーの能力に対して，次の 4 つの評価をおこなった．

1. 性質の違う概念を統一的に記述することができる

2. メニューの記述を容易におこなえる
3. 概念抽出の基準を与えることができる
4. ソフトウェア開発組織の記述をインスタンスとしておこなうことができる

これらの能力はソフトウェア開発組織だけでなく，一般のソリューションメニューの作成においても有効であり，開発したオントロジーはソリューションメニュー開発支援ツールとして使用できると判断した．

7.2

今回開発したオントロジーはより広いシステムのなかで使用することを想定したものである．このオントロジーは，記述内容が変更になっても，取扱いは同一のプログラムで可能としてあり，システムの整合性は高いと思われるが，その検証は今後の課題である．

提案したオントロジー再構成支援方法については，ソフトウェア開発組織をソリューションメニューとして記述する中で，簡単な例を示すことができた．しかし，実際のソリューションビジネスをおこなう中で意味のあるオントロジー再構成ができるかどうかは，実運用による検証をおこなわなければ分からない．

オントロジーの記述能力については，本研究で提案したメニューオントロジーでは，複数の条件を列挙しておき，そのいずれかにマッチするインスタンスに適用するようなメニューは記述できない．メニューオントロジーにより柔軟な記述能力を持たせるためには，この点の改良が必要である．

本研究をおこなうにあたっては，学内，学外を問わず，様々な方々のお世話になりました．ここに改めて感謝いたします．

とりわけ主テーマ指導教官の杉山公造教授には，知識教育センター長，学科長としてご多忙の中，貴重な助言と指導をしていただきました．

また講座副指導教官の下嶋篤助教授，副テーマを指導していただいた永田晃也助教授にも率直なご意見をいただきました．

研究室の学生の皆さんとの日々の交流の中からも率直な意見と感想を頂きました．

最後に，この研究をおこなうきっかけと多大なサポートをしてくださった富士電機株式会社に感謝いたします．

ありがとうございました．

2001 年 2 月 13 日

鈴木 聡

[1] 長谷川英一：米国におけるハードウェア・ベンダのソリューション・ビジネス戦略（前編），電子工業月報平成 11 年 2 月号，日本電子工業振興協会，1999, http://it.jeita.or.jp/jhistory/document/geppou/kaigai/ny99-02.html .

[2] 三菱電機エネルギーソリューション，http://www.melco.co.jp/service/eneso/solution/index.htm .

[3] 関口益照：情報産業（IT 産業）の特殊性とその技術的背景，第２回 日本社会情報学会大会，1997.

[4] (社)日本電子工業振興協会：ソリューションビジネス普及促進のために「ソリューションアーキテクチャ」を標準化，http://it.jeita.or.jp/jhistory/japanese/committee/SOL/000927.html .

[5] 長谷川英一：米国におけるハードウェア・ベンダのソリューション・ビジネス戦略（後編），電子工業月報平成 11 年 3 月号，日本電子工業振興協会，1999, http://it.jeita.or.jp/jhistory/document/geppou/kaigai/ny99-03.html .

[6] 富士通(株)：SolutionVision，http://www.fujitsu.co.jp/hypertext/svision/ .

[7] 青島矢一：新製品開発研究の視点, ビジネスレビュー, vol.45, No.1, pp.161-179, 1997.

[8] Clark, K. B. and T. Fujimoto : Product Development Performance : Strategy, Organization, and Management in the World Auto Industry, Harvard Business School Press, 1991.

[9] 楠木建：システム分化の組織論，ビジネスレビュー，vol.45，No.1，pp.129-150，1997.

[10] 富士通(株)：@SALESFORCEVISION，http://www.fujitsu.co.jp/hypertext/

svision/sfa/index.html .

[11] 溝口理一郎：オントロジー研究の基礎と応用，人工知能学会誌，Vol.14，No.6，pp.977-988，1999.

[12] Gruber, T. : What-is-an-ontology?, /http://www-ksl.stanford.edu/what -is-an-ontology.html .

[13] 伊藤亮吾，河口知幸，山口高平：細粒度エンタープライズオントロジーに基づく BPR 支援環境, 人工知能学会知識ベースシステム研究会（第 46 回）, pp.29-34, 2000.

[14] On-To-Knowledge : http://www.ontoknowledge.org/oil/ .

[15] Guarino, N. and Welty, C. : A Formal Ontology of Properties，Proc. of 12th Int. Conf. on Knowledge Engineering and Knowledge Management，2000.

[16] Alexander, C : The Timeless Way of Building, Oxford University Press, 1979.

[17] Ｆ．ブッシュマン，Ｒ．ムニエ，Ｈ．ローネルト，Ｐ．ゾンメルラード，Ｍ．スタル：ソフトウェアアーキテクチャ，トッパン，1999.

[18] Coplien，J．O．：A Development-Process Generative Pattern Language, Proceedings of PLoP'94, pp.183-237.

[19] W3C : http://www.w3.org/XML/ .

[20] ブレット・マクラフリン：XML Schema 活用講座，JavaWorld，Vol.5，No.3，pp.139-149.

[21] ISO/IEC DTR 22250-1, RELAX Core, 2000, http://www.xml.gr.jp/relax/ .

[1] 鈴木聡，杉山公造：モデル化とシミュレーションを活用した研究開発支援，人工知能学会知識ベースシステム研究会 （第 46 回），pp.35-40，2000.
[2] 鈴木聡，杉山公造：ソリューションビジネスにおける「メニュー」体系開発支援，情報処理学会 第 62 回全国大会，2001.

ソフトウェア開発組織のデザインパターンから作成したソリューションメニュー

ドメインオントロジー

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE DOntology SYSTEM "PPDomain.dtd">

<DOntology name="ADevelopmentProcessGenerativePatternLanguage">

<Entity type="organization">
        <attr name="size" type="String"/>
        <attr name="geographically_distributed"/>
        <attr name="architectual_partitioning"/>
        <attr name="reflect_market"/>
        <attr name="roles"/>
        <attr name="technical_direction"/>
        <attr name="compatible_with_architecture"/>
        <attr name="formed"/>
        <attr name="distracted"/>
        <attr name="communication"/>
        <attr name="interaction"/>
        <attr name="coupling"/>
        <attr name="people" type="Integer"/>
        <attr name="social_network"/>
        <attr name="condition"/>
        <attr name="context"/>
</Entity>

<Entity type="project">
        <attr name="schedule"/>
        <attr name="size"/>
        <attr name="target_date"/>
        <attr name="well_defined_rolls"/>
        <attr name="architectural_principles"/>
        <attr name="continuity"/>
        <attr name="customer_satisfaction"/>
        <attr name="customer_vis ion"/>
```

```
        <attr name="documentation"/>
</Entity>

<Entity type="developers">
        <attr name="reward"/>
</Entity>

<Entity type="role">
        <attr name="partial_definition_from_activities"/>
        <attr name="accountability_to_market"/>
        <attr name="name"/>
</Entity>

<Entity type="product">
        <attr name="size"/>
        <attr name="domain"/>
        <attr name="quality_is_to_be_assessed"/>
        <attr name="quality"/>
</Entity>

<Entity type="Developer">
        <attr name="staffed"/>
        <attr name="characterization"/>
        <attr name="place"/>
        <attr name="keep_up_with_code_change"/>
        <attr name="code_ownership"/>
        <attr name="coupled_with"/>
</Entity>

<Entity type="Patron">
        <attr name="staffed"/>
</Entity>

<Entity type="Architect">
        <attr name="staffed"/>
        <attr name="implement"/>
        <attr name="revie w"/>
        <attr name="coupled_with"/>
</Entity>

<Entity type="Testing">
        <attr name="staffed"/>
</Entity>

<Entity type="QA">
        <attr name="staffed"/>
        <attr name="place"/>
        <attr name="coupled_with"/>
</Entity>

<Entity type="Customer">
        <attr name="staffed"/>
```

```
        <attr name="coupled_with"/>
</Entity>

<Entity type="Mercenary_Analyst">
        <attr name="staffed"/>
</Entity>

<Entity type="Firewall">
        <attr name="staffed"/>
</Entity>

<Entity type="Gatekeeper">
        <attr name="staffed"/>
</Entity>

<Entity type="person">
        <attr name="domein_expert"/>
        <attr name="role"/>
</Entity>

<Entity type="communication">
        <attr name="control"/>
</Entity>

<Entity type="activities">
        <attr name="key_atomic_process"/>
</Entity>

<Entity type="test">
        <attr name="schedule"/>
        <attr name="level"/>
</Entity>

<Entity type="validate">
        <attr name="method"/>
</Entity>

<Entity type="functional_requirements">
        <attr name="method"/>
</Entity>

</DOntology>
```

メニューオントロジー

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE MOntology SYSTEM "PPMenu.dtd">

<MOntology name="ADevelopmentProcessGenerativePatternLanguage">

<Pattern name="Size the Organization">
<InitState entity="organization" attr="size">question</InitState>
<InitState entity="product" attr="size">more25KSLOC</InitState>
<ResultState entity="organization" attr="size">10people</ResultState>
</Pattern>

<Pattern name="Size the Organization">
<InitState entity="organization" attr="size">question</InitState>
<InitState entity="product" attr="size">more25KSLOC</InitState>
<ResultState entity="organization" attr="size">10people</ResultState>
</Pattern>

<Pattern name="Solo Virtuoso">
<InitState entity="organization" attr="size">question</InitState>
<InitState entity="product" attr="size">less25KSLOC</InitState>
<ResultState entity="organization" attr="size">1people</ResultState>
</Pattern>

<Pattern name="Size the Schedule">
<InitState entity="project" attr="schedule">question</InitState>
<InitState entity="project" attr="size">estimated</InitState>
<ResultState entity="developers" attr="reward">for_meet_the_schedule</ResultState>
<ResultState entity="project" attr="target_date">flexible</ResultState>
</Pattern>

<Pattern name="Form Follows Function">
<InitState entity="organization" attr="formed" >yes</InitState>
<InitState entity="project" attr="well_defined_rolls">no</InitState>
<InitState entity="activities" attr="key_atomic_process">known</InitState>
<ResultState entity="role" attr="partial_definition_from_activities">yes</ResultState>
</Pattern>


<Pattern name="Domain Expertise in Roles">
<InitState entity="Developer" attr="characterization">known</InitState>
<InitState entity="person" attr="domein_expert">yes</InitState>
<ResultState entity="person" attr="role">Developer</ResultState>
</Pattern>

<Pattern name="Organization Follows Location">
<InitState entity="organization" attr="geographically_distributed">yes</InitState>
```

<ResultState entity="organization" attr="architectual_partitioning">geographic_partition</ResultState>
</Pattern>

<Pattern name="Organization Follows Market">
<InitState entity="role" attr="accountability_to_market">no</InitState>
<InitState entity="project" attr="architectural_principles">yes</InitState>
<ResultState entity="organization" attr="reflect_market">yes</ResultState>
</Pattern>

<Pattern name="Developer Controls Process">
<InitState entity="role" attr="partial_definition_from_activities">yes<InitState>
<InitState entity="communication" attr="control">no</InitState>
<InitState entity="product" attr="domain">imperfectly_understood</InitState>
<ResultState entity="Developer" attr="staffed">yes</ResultState>
<ResultState entity="Developer" attr="place">hub</ResultState>
<ResultState entity="communication" attr="control">yes</ResultState>
</Pattern>

<Pattern name="Patron">
<InitState entity="project" attr="continuity">no</InitState>
<InitState entity="organization" attr="roles">defined</InitState>
<InitState entity="Developer" attr="place">hub</InitState>
<InitState entity="Patron" attr="staffed">no</InitState>
<ResultState entity="Patron" attr="staffed">yes</ResultState>
<ResultState entity="person" attr="role">Patron</ResultState>
</Pattern>

<Pattern name="Architect Controls Product">
<InitState entity="organization" attr="technical_direction">needed</InitState>
<InitState entity="Developer" attr="staffed">yes</InitState>
<InitState entity="Architect" attr="staffed">no</InitState>
<ResultState entity="person" attr="role">Architect</ResultState>
<ResultState entity="Architect" attr="staffed">yes</ResultState>
</Pattern>

<Pattern name="Conways Law">
<InitState entity="Architect" attr="staffed">yes</InitState>
<InitState entity="organization" attr="compatible_with_architecture">no</InitState>
<ResultState entity="organization" attr="compatible_with_architecture">yes</ResultState>
</Pattern>

<Pattern name="Architect Also Implements">
<InitState entity="organization" attr="technical_direction">needed</InitState>
<InitState entity="Architect" attr="staffed">yes</InitState>
<InitState entity="Architect" attr="implement">no</InitState>
<ResultState entity="Architect" attr="implement">yes</ResultState>
</Pattern>

<Pattern name="Review">
<InitState entity="product" attr="quality_is_to_be_assessed">yes</InitState>
<InitState entity="Architect" attr="review">no</InitState>
<InitState entity="Architect" attr="implement">yes</InitState>

<ResultState entity="Architect" attr="review">yes</ResultState>
<ResultState entity="project" attr="documentation">needed</ResultState>
</Pattern>

<Pattern name="Code Ownership">
<InitState entity="Developer" attr="keep_up_with_code_change">no</InitState>
<InitState entity="Developer" attr="code_ownership">no</InitState>
<ResultState entity="Developer" attr="code_ownership">yes</ResultState>
</Pattern>

<Pattern name="Application Design is Bounded By Test Design">
<InitState entity="Testing" attr="staffed">yes</InitState>
<InitState entity="test" attr="schedule">question</InitState>
<ResultState entity="test" attr="level">scenario_driven</ResultState>
<ResultState entity="test" attr="schedule">scenario_requirements_agreed</ResultState>
<ResultState entity="test" attr="level">low_level</ResultState>
<ResultState entity="test" attr="schedule">interfaces_stabilized</ResultState>
</Pattern>

<Pattern name="Engage QA">
<InitState entity="product" attr="quality">guarantee</InitState>
<InitState entity="Developer" attr="staffed">yes</InitState>
<ResultState entity="QA" attr="staffed">yes</ResultState>
<ResultState entity="QA" attr="place">central</ResultState>
<ResultState entity="QA" attr="coupled_with">Developer</ResultState>
<ResultState entity="person" attr="role">QA</ResultState>
<ResultState entity="Developer" attr="coupled_with">QA</ResultState>
</Pattern>

<Pattern name="Engage Customer">
<InitState entity="project" attr="customer_satisfaction">needed</InitState>
<InitState entity="QA" attr="staffed">yes</InitState>
<InitState entity="Developer" attr="staffed">yes</InitState>
<InitState entity="Architect" attr="staffed">yes</InitState>
<ResultState entity="Customer" attr="staffed">yes</ResultState>
<ResultState entity="Customer" attr="coupled_with">Developer</ResultState>
<ResultState entity="Customer" attr="coupled_with">Architect</ResultState>
<ResultState entity="Customer" attr="coupled_with">QA</ResultState>
<ResultState entity="Developer" attr="coupled_with">Customer</ResultState>
<ResultState entity="Architect" attr="coupled_with">Customer</ResultState>
<ResultState entity="QA" attr="coupled_with">Customer</ResultState>
</Pattern>

<Pattern name="Group Validation">
<InitState entity="product" attr="quality">guarantee</InitState>
<InitState entity="Customer" attr="staffed">yes</InitState>
<ResultState entity="validate" attr="method">CRC_cards</ResultState>
</Pattern>

<Pattern name="Scenarios Define Problem">
<InitState entity="project" attr="customer_vision">needed</InitState>
<InitState entity="Customer" attr="staffed">yes</InitState>

<ResultState entity="functional_requirements" attr="method">use_case</ResultState>
</Pattern>

<Pattern name="Mercenary Analyst">
<InitState entity="Customer" attr="staffed">yes</InitState>
<InitState entity="project" attr="documentation">needed</InitState>
<ResultState entity="person" attr="role">Mercenary_Analyst</ResultState>
<ResultState entity="Mercenary_Analyst" attr="staffed">yes</ResultState>
</Pattern>

<Pattern name="Firewalls">
<InitState entity="organization" attr="formed">yes</InitState>
<InitState entity="organization" attr="distracted">yes</InitState>
<InitState entity="Firewall" attr="staffed">no</InitState>
<ResultState entity="Firewall" attr="staffed">yes</ResultState>
<ResultState entity="person" attr="role">Firewall</ResultState>
<ResultState entity="organization" attr="distracted" >no</ResultState>
</Pattern>

<Pattern name="Gatekeeper">
<InitState entity="organization" attr="formed">yes</InitState>
<InitState entity="organization" attr="communication">need_to_balance</InitState>
<ResultState entity="Gatekeeper" attr="staffed">yes</ResultState>
<ResultState entity="person" attr="role">Gatekeeper</ResultState>
<ResultState entity="organization" attr="distracted" >no</ResultState>
</Pattern>

<Pattern name="Shaping Circulation Realms">
<InitState entity="organization" attr="interaction" >not_as_should_be</InitState>
<InitState entity="organization" attr="formed" >no</InitState>
<ResultState entity="organization" attr="formed" >yes</ResultState>
</Pattern>

<Pattern name="Move Responsibilities">
<InitState entity="organization" attr="coupling" >undesirable</InitState>
<ResultState entity="organization" attr="coupling" >balanced</ResultState>
</Pattern>

</MOntology>

デザインパターンの Problem，Context，Solution の各部をタグで囲ったもの

```
<?xml version="1.0" encoding="Shift_JIS"?>
<PatternLanguage>

<Pattern name="Buffalo Mountain">
<Problem>
We want to optimize communications in a large software development organization, whose members are working together on a common product.
</Problem>
<Context>
A development organization straddling several domains, where effective interpersonal communication is key to project success.
A nominal “hub” may already have been established for the organization.
</Context>
<Solution>
1. For any significant project interaction, the distance of the two collaborating roles from the “center” of the organization should sum to a distance less than the smallest distance necessary to span the entire organization.
2. Avoid coupling with neighbors (those equidistant from the center of the process as yourself; i.e., those equally
coupled to the process as a whole as yourself) if you are in the outlying 50% of the organization.
3. The intensity of any interaction should be inversely proportional to the sum of the differences of the roles' distance to the center of the process.
Means A: Shuffle responsibilities between roles in a way that moves the associated collaborations, to effect the above patterns (pattern Move Responsibilities on page 21). This is Robert Lai's idea; it's simple, but profoundly effective.
Means B: Physically relocate people to enhance their opportunity to communicate (see Shaping Circulation Realms on page 21).
Means C: Increase span of control of a role in the project (akin to merging multiple roles into one). It is probably best to Merge Roles with Similar Responsibilities, or better, to Merge Roles with Similar Collaborations. Hey, those are patterns too!
</Solution>
</Pattern>

<Pattern name="Work flows inward">
<Problem>
 Work that adds value directly to the product should be done by authoritarian roles.
</Problem>
<Context>
An existing organization where the flow of information can be analyzed. The organization exhibits a management pecking order.
</Context>
<Solution>
Work should be generated by customers, filter through supporting roles, and be carried out by implementation experts at the center. You should not put managers at the center of the communication grid: they will become overloaded and make decisions that are less well-considered, and they will make decisions that don’t take day-to-day dynamics into account.
</Solution>
</Pattern>
```

<Pattern name="3 to 7 Helpers per Role">
<Problem>
Uneven distribution of communication.
</Problem>
<Context>
An organization whose basic social network has been built.
</Context>
<Solution>
Ensure that each role has between 3 and 7 helpers.
</Solution>
</Pattern>

<Pattern name="Named Stable Bases">
<Problem>
How frequently do you integrate?
</Problem>
<Context>
A schedule framework has been determined.
</Context>
<Solution>
Stabilize system interfaces – the architecture – no more frequently than once a week.
</Solution>
</Pattern>

<Pattern name="Divide and Conquer">
<Problem>
Organizations grow to the point where they cannot easily manage themselves.
</Problem>
<Context>
The roles have been defined for a process and organization, and the interactions between them are understood.
</Context>
<Solution>
Find clusters of roles that have strong mutual coupling, but that are loosely coupled to the rest of the organization.
Form a separate organization and process from those roles.
</Solution>
</Pattern>

<Pattern name="De-couple Stages">
<Problem>
How do you de-couple stages (architecture, design, coding) in a development process?
</Problem>
<Context>
A design and implementation process for a well-understood domain.
</Context>
<Solution>
For known and mature domains, serialize the steps. Handoffs between steps should take place via well-defined interfaces. This makes it possible to automate one or more of the steps, or to create a pattern that lets inexpert staff carry out the step.
</Solution>
</Pattern>

<Pattern name="Hub-Spoke-and-Rim">
<Problem>
How do you de-couple stages (architecture, design, coding) in a serialized development process while maintaining responsiveness?
</Problem>
<Context>
A design and implementation process. A well-understood domain. De-couple Stages on page 27 has been applied.
</Context>
<Solution>
Link each role to a central role that orchestrates process activities. Parallelism can be re-introduced if the central role pipelines activities.
</Solution>
</Pattern>

<Pattern name="Aesthetic Pattern">
<Problem>
An organization has an irregular structure
</Problem>
<Context>
The organization has been designed through preceding patterns. The project is in early development of the first release of its product. The organization must plan how to evolve itself beyond the product's first release, and eventually into the product's maintenance phase.
</Context>
<Solution>
Make sure the organization has identifiable sub-domains that can grow into departments of their own as the project thrives and expands to serve a maintained market.
</Solution>
</Pattern>

<Pattern name="Coupling Decreases Latency">
<Problem>
The process is not responsive enough; development intervals are too long; market windows are not met.
</Problem>
<Context>
A service process and, perhaps in special cases, a small design/implementation process using an iterative or incremental approach.
</Context>
<Solution>
Open communication paths between roles to increase the overall coupling/role ratio, particularly between central process roles.
</Solution>
</Pattern>

<Pattern name="Prototype">
<Problem>
Early acquired requirements are difficult to validate without testing.
</Problem>
<Context>
Trying to gather requirements necessary for test planning, as in pattern Application Design is Bounded By Test Design on page 15, and for the architecture, as for the pattern Architect Controls Product on page 13.

</Context>
<Solution>
Build a prototype, whose purpose is to understand requirements. Prototypes are particularly useful for external interfaces. Throw the prototype away when you ' re done.
</Solution>
</Pattern>

<Pattern name="Take No Small Slips">
<Problem>
How long should the project take?
</Problem>
<Context>
The product is under way and progress must be tracked.
</Context>
<Solution>
" We found a good way to live by ' Take no small slips ' from... ' The Mythical Man Month. ' Every week, measure how close the critical path (at least) of the schedule is doing. If it 's three days beyond schedule, track a ' delusion index ' of three days. When the delusion index gets too ludicrous, then slip the schedule. This helps avoid churning the schedule. " - Chisholm.
</Solution>
</Pattern>

<Pattern name="Interrupts Unjam Blocking">
<Problem>
The events and tasks in a process are too complex to schedule development activities as a time-linear sequence.
</Problem>
<Context>
A high productivity design/implementation process or low-latency service process.
</Context>
<Solution>
If a role is about to block on a critical resource, interrupt the role that provides that resource so they stop what they're doing to keep you unblocked.
If the overhead is small enough, it doesn't affect throughput. It will always improve local latency.
</Solution>
</Pattern>

<Pattern name="Dont Interrupt an Interrupt">
<Problem>
The pattern Interrupts Unjam Blocking on page 32 is causing people to thrash.
</Problem>
<Context>
Execution of pattern for Interrupts Unjam Blocking on page 32.
</Context>
<Solution>
If a role is about to block on a critical resource, interrupt the role that provides that resource so they stop what they're doing to keep you unblocked.
If the overhead is small enough, it doesn't affect throughput. It will always improve local latency.
</Solution>
</Pattern>

<Pattern name="Compensate Success">

<Problem>
Providing appropriate motivation for success.
</Problem>
<Context>
A group of developers meeting tight schedules in a high-payoff market.
</Context>
<Solution>
Establish lavish rewards for individuals contributing to make-or-break projects. The entire team (social unit) should receive comparable rewards, to avoid de-motivating individuals who might assess their value by their salary relative to their peers.
A celebration is a particularly effective reward.28
</Solution>
</Pattern>

</PatternLanguage>