MMRC-J-67

コーディネーション・システムとしての製品アーキテクチャ

東京大学大学院経済学研究科
経済産業研究所ファカルティフェロー
奥野　正寛

東京大学21世紀COEものづくり経営研究センター
渡邊　泰典

2006 年 2 月

東京大学21世紀COE〔センター・オブ・エクセレンス〕
ものづくり経営研究センター

コーディネーション・システムとしての製品アーキテクチャ*

東京大学大学院経済学研究科
経済産業研究所ファカルティフェロー

奥野　正寛

東京大学 21 世紀 COE ものづくり経営研究センター

渡邊　泰典

2006 年 2 月

概要 ： Baldwin and Clark(2000)を契機に注目されるようになった、製品アーキテクチャ(architecture)という概念、特にモジュール型アーキテクチャという概念は、ユーザーが使う対象あるいはシステム開発の対象という視点から、製品システムを一つの「コーディネーション・システム」としてとらえていると考えられる。本稿では製品アーキテクチャを、ユーザーの目的を達成するために部品間の複雑な調整を必要とする「製品システム」の設計思想として定義し、モジュール型アーキテクチャによって、(1) 製品機能が明確化され、(2) ユーザーの命令と動作の対応関係が明確化されることを示す。その結果、文脈依存型に行動する人間であっても、高度に複雑化するシステムを操作・利用するだけでなく、分業と協業を組み合わせながら新たなシステムを開発・設計することが可能になる。

* 本稿の多くを、東京大学大学院経済学研究科で行われた「アーキテクチャ理論研究会」における中尾政之（東京大学大学院工学系研究科）、藤本隆宏（東京大学大学院経済学研究科）、安藤晴彦（経済産業省）、中馬宏之（一橋大学イノベーション研究センター）、池田信夫（国際大学グローコム：当時）の各氏の報告と研究会における議論に負っている。また本稿をまとめる上で、滝澤弘和、柳川範之、鶴光太郎、木村友二の各氏から貴重な助言とコメントを頂いた。これらの方々と研究会のメンバーに心から感謝したい。

1 はじめに

歴史を振り返ったとき人間の活動は、量的にも質的にも急速な拡大を遂げてきた。人類が生まれたとき、人は家族単位でしか行動していなかった。しかし、狩猟採集や農耕活動を通じてグループや村落での共同活動が起こり、さらに工業化や交通手段の発達に伴って、都市から国家、さらには地球全体へとその協力行動の規模を広げてきた。

人間活動も、利用する道具や器械の発展で、また人力から家畜・風水力へ、さらに電力・原子力へという動力の高度化につれて、急激に高度化してきた。人類の進歩と共に技術知識が獲得・蓄積され、それがさらに一層高度な技術知識の開発を促してきたからである。その行き着いた先が、現代の電子技術とデジタル処理を基礎にする情報処理技術である。

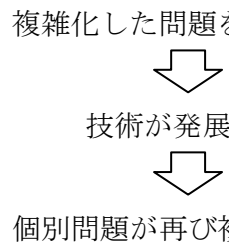
このような人類の発展過程は、膨大な数の人間が、個人としてあるいは企業などの組織を通じて、複雑に相互依存する現代社会を生み出した。同時に、技術知識の深化と共に、人間の道具や機械への依存関係はより一層深化・複雑化してきた。機械や動力を使うことで、人間の外部環境への適応能力がより一層高まるからである。

このような事情を背景に、多数の人間が相互に関連しあう社会も、人間が利用する機械や情報処理システムも、ますます複雑化している。複雑化に伴って、社会やシステムはメンバーや部品の**コーディネーション**という問題に直面する。膨大な数の人間で共同して作業を行うためには、あるいは多数の部品パーツが連携して動作するためには、

- 全体が直面している外部環境や個々のメンバー（部品）が行っている作業がどんなものかという情報をお互いに情報共有し、
- 全体として整合的でしかも外部環境にできるだけ有効に対処できる作業を決定し、
- それを各メンバー（部品）に指示・実行させなければならない。

からである。

人間や社会が採用してきた複雑化への対応が**分業と協力**の組み合わせである。複雑に関連しあっている関係を単純で標準化された作業に分解することで、個々のメンバーや部品が扱える範囲を限定する。他方、限定された作業同士の関係自体をも標準化することで、メンバー同士や部品同士が協力しコーディネートしやすくする。問題を分割することで、各人が技術的に特化しやすく、また競争が活発化することによってシステム全体のパフォーマンスを高めるという効果も生じる。実際に歴史を振り返ってみれば、



という流れを繰り返しながら、社会・経済的な発展が生じていることは明らかである。

このような**分業による協業**は、複雑性への対処という意味で歴史的に非常に有効だった。しかし、近年の情報化技術の発展や技術の高度化、社会・産業の規模は、この分業と協業の組み合わせをさらに一段レベルアップさせたのではないだろうか。

このことをわれわれは、**製品アーキテクチャ**という概念を通じて考えてみたい。Baldwin and Clark（2000）を契機に注目されるようになった、製品アーキテクチャ（architecture）という概念、特にモジュール型アーキテクチャという概念は、ユーザーが使う対象あるいはシステム開発の対象という視点から、製品システムを一つの「コーディネーション・システム」としてとらえていると考えられるからである。

藤本（2003）に従えば、製品設計の基本特性としての製品アーキテクチャとは、

- どのようにして製品を構成部品や工程に分割し、
- そこに製品**機能**を配分し、
- それによって必要となる部品間の・工程間のインターフェースをいかに設計・調整するか、

に関する基本的な設計思想である。

製品システムはまた、他の社会システムと異なって、それを操作するユーザー以外、意思を持った部品が存在しない。インセンティブやモティベーションの問題を捨象できるという意味で、製品システムは考えうるもっとも単純なコーディネーション・システムの一つなのである。

製品システムを考える場合、ユーザーである人間は帰納的な思考を基に文脈依存型の行動をする。逆に、機械的・電子的な仕組みは、定型化された情報の論理的ですばやい処理にその威力を発揮する。マニュアル化さえできれば、機械的・電子的仕組みは高い能力を発揮するが、予測しなかった事態や多数の部品が相互干渉し始めると、それを解決する能力を持たない。その場合こそ、ユーザーである人間の大局的な視点からの経験に基づいた判断が大き

な威力を発揮する。人間と機械的・電子的仕組みは補完的であり、両者の比較優位を生かしつつ、複雑なコーディネーション問題を解決する仕組みが求められることになる。

本稿でわれわれが主張したいのは、モジュール型アーキテクチャとは、

- 暗黙知を使いながら文脈依存型に行動する人間が、機械的・電子的仕組みで動く製品をコントロールするためには、人間の命令を製品システムが判る言葉に翻訳する機能が必要である。
- そのためには、ユーザーが製品の諸機能をコントロールするという文脈で命令を出せるよう、製品機能を明確化する必要がある。
- またユーザーからの機能別命令を機械部品が理解できるよう、機械語のコード体系を明確にし、それと機能別命令との対応関係を明確化する必要がある。
- そうすれば、ユーザーが発した命令に従って、機械システム全体が適切に機能するよう個別部品をコーディネートすることを、機械的制御に任せることが可能になる。
- この「機能別命令の集合」とそれに対応する「機械語コード体系」、さらにそれを実現するための「機械的制御の仕組み」という三つの組を「標準（スタンダード）」として公開し、一定期間固定することにコミットする。

という設計思想だ、という点である。

このようなアーキテクチャによって、

- 標準として公開されている機能別命令、機械語コード、機械的制御にさえしたがった部品モジュールであれば、同じ機能を果たすためにどんな部品モジュールを作ればそれを全体システムに接続できるかが、部外者にも明確になる。
- その結果、部品モジュール間の競争が活発になり、モジュールの取替えによってシステムを多様化することが容易になる。
- システム開発や生産が、部品モジュールごとに独立に分業で行われることになる。
- システム全体のコーディネーション機能は、高まりこそすれ低まることはない。

という結果が生まれる。

このようにモジュール型アーキテクチャとは、単に機能を標準化することで、ユーザーにも使いやすい製品を作り出すという伝統的な形での人間と機械の協業を、新しい製品アーキテクチャを導入することによってさらに一段レベルアップさせた仕組みだと、われわれは考えている。電子的な仕組みによってユーザーの命令を機械語に翻訳し、部品間コーディネー

ションを自動制御する仕組みを人間と機械部品との間に挿入することによって、製品操作が格段に容易になる。しかも、電子的な機械制御によって各部品モジュールの動作は、いわばマニュアルどおり動くことだけが要求されることになり、部品モジュールの開発や生産も格段に容易になる。さらにシステム全体をモジュール化することによって、分業と協力を組み合わせながら、高度に複雑なシステムを開発・設計・生産・使用することも可能になる。

以下では、コーディネーション・システムとは何か、人間の文脈型思考とは何か、製品の機能とは何か、製品アーキテクチャとは何を指すのか、などを鍵概念として、これらの点を分析してみたい。

2 システムとコーディネート

2.1 コーディネーションシステムとは何か

コンピュータや自動車、あるいは企業組織や企業ネットワークを例に取るまでもなく、さまざまな製品・企業システムの複雑化や多様化には目を見張るものがある。

近年の情報通信技術の発展は、システムが、従来の基準を超えた量の情報を取り扱うことを可能にした。こうした情報通信技術を用いることで、システムを取り囲む環境についての膨大な情報を、システムの運営に反映させることが可能になったのである。しかも、これらの技術を用いて機械的・電子的なシステム運営を行えば、人間が直接行うよりはるかに迅速かつ正確に、システムを運営することが可能である。

他方、システムが望ましい結果を得るためには、システムを取り巻く不確実性（環境情報）が変化するにしたがって、システムを構成する様々な構成要素（組織であれば組織のメンバー・エージェント、製品であればなんらかの部品）をコーディネートすることが必要になる。しかし、環境情報は多様であり、また急速に変化する。しかも先述したように、情報通信技術の発達によって、我々は以前よりも膨大な量の情報を利用することが可能となっている。

2.1.1 機械と人間

その際、いくら進歩した情報通信技術であっても、ソフトを書き換えマニュアルを変更しない限り、これら急激で複雑な環境変化には対応できない。言い換えれば、システムを究極的にコントロールする主体はあくまでも限定合理的な人間でしかない。

ところで確かに、硬直的な規則に縛られつつ、迅速・正確な処理が可能な情報通信技術や機械工学的な技術と異なって、人間は、錯綜する事態を認識し、弾力的な対応を行うことが可能である。しかし同時に、人間も限定合理的であり、膨大な情報を必要な時間内に、正確に認識し、適切に処理することは実質的に不可能である。

したがって、システムが人間にとってコントロール可能であるためには、

- この人間の持つ、「複雑・錯綜した事態を大局的視点から把握し、弾力的に対処できる」という能力と、
- 電子的・機械的な仕組みが持つ、「事前に設計された硬直的な規則の枠内であれば、きわめて迅速・正確に情報処理を行い、必要な対応を行うことが可能である」という能力を、

どのように組み合わせることが望ましいか、という点が重要な問題となる。

要するに、人間と（電気・電子的な仕組みや情報処理システムを含む）機械は、それぞれ異なる比較優位を持っており、システムをうまく機能させるためには、二つの間の補完性を生かすことが必要である。他方、人間と機械とは異なる原理で動いているから、その間の命令伝達や情報処理の仕組みをうまく作る必要もある。従って、人間と機械の間で、必要な作業をどのように分業させるか、その際、どのような情報伝達の仕組みによってお互いの補完性を高めるのかが、問われることになる。

2.1.2 システムとは何か

このような視点からは、モジュール化やすり合わせといった、システムをコントロールするいくつかの標準的な仕組みを統一的に理解するための理論的枠組を提供することが必要だろう。本稿は、そのための一つの試みである。

それにはまず、システムとは何か、という認識を共有しておく必要がある。

本稿では、異なる機能を分担する多数の構成要素や部品から構成され、しかもそれらの要素間・部品間の動作のコーディネーションが、全体のパフォーマンスに重要な役割を果たす組織や製品を、「システム」と呼ぶ。その意味で、既に述べたように、製品設計チームや企業組織、それぞれが部品生産・組立・販売を担当する企業グループ、産学官のネットワーク、社会や経済システム全体など、さまざまなレベルのシステムを考えることが可能である。

とはいえ、それではあまりにも焦点が絞りきれない。そこで以下では、「一人のユーザーが使用する製品システム」に分析の対象を絞ることにする。

具体的には、システムを構成する構成要素（部品・パーツ）が、「意思を持たない」ために、それらの「インセンティブ」を考えないですむ場合を取り出して議論したい。もちろん構成要素の意思やインセンティブは組織構造の決定にあたっては極めて重要だが、それらを捨象することでかえって、コーディネーション問題を純粋な形で抽出できると考えるからで

ある。

2.2 コーディネーション問題

抽象的に定義すれば、本稿で考えるコーディネーション問題は、次のような要素から構成される。

2.2.1 要素部品

製品システムは、膨大な数の部品から成立している。ある製品に含まれる部品のインデックスの集合を $N = \{1, 2, \ldots, n\}$ で表し、i 番目の部品の動作（状態）を $x_i \in X_i$ で表そう。X_i は、部品 i が取れる可能な動作（状態）の集合を表している。

製品全体の動作（状況）とは、各部品がどんな動作を取っているかをすべて記述したもの、つまり、部品動作のプロフィール $\boldsymbol{x} = (x_i)_{i \in N} \in \times_{i \in N} X_i \equiv \boldsymbol{X}$ で表される。製品は、部品動作のプロフィールを変更することによって、製品全体の動作を変えるのである。

逆にユーザーは、与えられた製品が許す範囲内で、自分が望むような動作を製品に選択させることができる。従って以下では、ユーザーが特定の部品動作のプロフィール $\boldsymbol{x} = (x_i)_{i \in N}$ を選択することを、ユーザーが**製品を動作させる**、あるいは**製品を動かす**ということにする。

2.2.2 製品

ユーザーが製品を動作させられるのは、製品が許す範囲内であると述べたのは、製品の物理的特徴には違いがあるからである。同じ目的のために設計された、良く似た製品同士であっても、その内部構造は異なる。

一つの**製品（product）**は $\pi = (N, \boldsymbol{X})$ で表され、別の製品は $\pi' = (N', \boldsymbol{X}')$ で表される。$\pi = (N, \boldsymbol{X})$ の方が $\pi' = (N', \boldsymbol{X}')$ より多くの部品で成り立っていれば、$N \supset N'$ である。また、$\pi = (N, \boldsymbol{X})$ と $\pi' = (N', \boldsymbol{X}')$ には共通の部品が使われていない、つまり、$N \cap N' = \emptyset$ という場合もあるだろう。

また、二つの製品が同時に使っている部品 $i \in N \cap N'$ であっても、$\pi = (N, \boldsymbol{X})$ の方が $\pi' = (N', \boldsymbol{X}')$ よりも高度な部品を使っているため、$X_i \supset X_i'$ であり、X_i のほうが X_i' より多様な値をとることも可能かも知れない。

2.2.3 外部環境

ユーザーとユーザーが使用する製品は、さまざまな不確実性を含む**外部環境**の中に置かれている。外部環境によって、ユーザーが製品にどのようなパフォーマンス（機能）を実現させたいか、そのために製品全体がどのようなコーディネーションを必要とするかが決まるこ

とになる。

以下、ユーザーと製品が直面している**不確実性**を、$\omega \in \Omega$であらわす。これらの不確実性が起こる**確率**を、$p:\Omega \to \mathbb{R}_+$であらわす。つまり、$p(\omega) \in [0,1]$が$\omega$が起こる確率（密度）である。

2.2.4 動作関数

外部環境を認識するのはユーザーであり、製品を操作するのもユーザーである。従って、製品が動くのは、ユーザーが認識した外部環境ごとに、どのように各部品の動作状態を実現するかにかかっている。

他方、ユーザーが製品を操作すると、複数の要素部品が自動的に連携して作動することもある。この場合、ユーザーは、製品を構成するすべての部品を直接操作する必要はない。ユーザーが比較的単純な操作をするだけで、各個別部品が複雑に連携しあって動作することが可能になる。

ユーザーの操作によって動く部分と、ユーザーの操作に伴って製品内部で自動的に部品がコーディネートされる部分を含めて、ユーザーが外部環境を認識し、それにしたがって製品を操作するという関係を、**動作関数（operation function）**、$\phi(\cdot|\pi):\Omega \to \boldsymbol{X}$ で表そう。つまり、ユーザーが製品を操作するという行為を、以下では、ユーザーが認識した外部環境が$\omega \in \Omega$のときには、部品状態を$\phi(\omega|\pi) \in \boldsymbol{X}$にする、というルールとして表すことになる。

すでに述べたように、

- 動作関数ϕの一部は、ユーザーの命令（以下、**外部操作**と呼ぼう）そのものであり、
- 残りは、ユーザーの命令に従って製品内部でそれが処理される自動的な流れ（以下、**内部動作**と呼ぼう）を示すことになる。

ある製品を（コーディネーション・）システムとしてとらえるということは、ユーザーが操作することによって、製品を構成する各部品の動作がどう変わるか、それによって各要素部品がどうコーディネートされるかを捉えることに他ならない。

したがって、単に部品のインデックスと各部品がとりうる状態の集合のペアである（物理的な）製品$\pi = (N, \boldsymbol{X})$だけでなく、動作関数のうち内部動作を含めて考えれば、ユーザーの操作によって（物理的な）製品全体がどう動作するか、ということまで含んだ概念として製品を定義できる。以下では、**（製品）システム**という言葉で、どんな物理的製品がどんな内部動作関数を持っているのかを表すことにする。

動作関数をこれら二つの部分にどう分割し、その間の情報処理をどう仕組むかということが、実は製品アーキテクチャという概念の基礎にある、というのが本稿の基本的視点のひとつである。

2.2.5 製品が生み出す利得

製品の物理的特性 $\pi=(N,\boldsymbol{X})$ がきまり、それがおかれた環境 $\omega\in\Omega$ が与えられたとしよう。その際、どんな部品動作のプロフィール $x\in\boldsymbol{X}$ が実現されるかによって、ユーザーが獲得する**製品からの利得** $\upsilon(\cdot|\pi):\boldsymbol{X}\times\Omega\to\mathbb{R}$ が決まる。つまり、不確実性 $\omega\in\Omega$ が与えられ、製品の動作が $x\in\boldsymbol{X}$ であるときに、製品 $\pi=(N,\boldsymbol{X})$ が生み出す利得（付加価値）は、$\upsilon(x,\omega|\pi)\in\mathbb{R}$ である。

さらに（外部操作と内部動作の双方を含む）動作関数 $\phi(\cdot|\pi):\Omega\to X$ が与えられれば、製品が実現する付加価値の期待値が決まる。具体的にそれは、

$$V(\pi,\phi(\cdot|\pi))=E\upsilon(\phi(\omega|\pi),\omega|\pi)=\int_{\omega\in\Omega}\upsilon(\phi(\omega|\pi),\omega|\pi)\,p(\omega)d\omega$$

として定義できることになる。

2.2.6 最善解

最後に、与えられた製品の物理特性を最大限に生かすことが可能な場合、つまり最善の動作関数を定義しておこう。

ユーザーが全知全能の能力を持っており、他方、製品側の各部品要素が、ユーザーの命令に弾力的かつ正確・迅速に対応できるなら、ユーザーが選択する動作関数は最善のものになり、製品 $\pi=(N,\boldsymbol{X})$ はその性能を完全に発揮できる。この動作関数を、π の**最善（first best）解**と呼ぶ。

それは、与えられた外部環境 $\omega\in\Omega$ ごとに、

$$\phi^{*}(\omega|\pi)=\arg\max_{x\in\boldsymbol{X}}\upsilon(x,\omega|\pi)$$

として定義される、最善の動作 $\boldsymbol{x}^{*}(\omega|\pi)$ を選ぶことである。

最善解を常に選ぶことができれば、ユーザーは製品 $\pi=(N,\boldsymbol{X})$ を使うことで、

$$V^{*}(\pi)=E\upsilon(\phi^{*}(\omega|\pi),\omega|\pi)=\int_{\omega\in\Omega}\upsilon(\phi^{*}(\omega|\pi),\omega|\pi)\,p(\omega)d\omega$$

を実現できる。

3 人間の限定合理性と製品システムの硬直性

しかし現実の人間は全知全能の神ではない。ユーザーは限定合理的な存在でしかなく、様々な限界を持っている。他方、製品の各部品要素も、ユーザー側の命令の意図を適切に判断し、それに正確・迅速に対応するような知性を備えていない。各部品要素にできるのは、設計者が設計した仕様に従って、ユーザーの命令を、いわば「マニュアル」通りに実行することでしかない。

そこで、このような人間と製品システムが適切な形で分業・協力することによって、システムの利得（付加価値）を高めたいと言うことになる。では、ユーザーである人間と、機械あるいは情報処理システムである製品システムとは、それぞれどんな欠点（限界）と利点（特徴）を持ち、どんな形で補完しあうことが望ましいだろうか。

このことを、ユーザーが製品システムを操作する際のプロセスを分解することによって、検討してみたい。システムを操作するためには

(1) 外部環境の認知
(2) 認知した外部環境に対し、適切なシステム状態を決定
(3) 実際にシステムを操作

という、少なくとも 3 段階の意思決定プロセスを経る必要がある。

このそれぞれを実現するためには、いくつもの制約がある。以下、それらを製品システムにまつわる問題と人間の認知・意思決定プロセスにまつわる問題とに分割して検討しよう。

3.1 製品システムの硬直性

3.1.1 部品動作の相互依存と操作に必要な技術的・専門的知識

ユーザーが製品を操作する（与えられた外部環境の中で適切な動作状態を選択する）ためには、各部品要素がどんな役割を果たしており、それらがどのように相互依存しているかを知らなければならない。

例えば、自動車が高速でカーブを曲がるとき、これでは危険だとユーザーが思えば、車を減速させたいと思う。そのためにユーザーが直接、個々の部品の状態を調整するならば、例えば、エンジンの回転数を減らすことが必要になる。そのためには、エンジンへのガソリン供給速度を下げ、ガソリンの量が減ったから混合比を保つために燃焼室への空気の供給量も調整する必要がある。

エンジンの回転数が減ればスピードが下がり、ラジェーターの放熱効果が減退する。放っておけばオーバーヒートするから、ファンの回転数を上げてラジェーターを冷やさなければならない。そのためには、必要な電力をバッテリーから供給をしてやらなければならない。

このように、製品システムを直接コントロールするためには、各部品がどんな役割を果たしており、一つの部品を調整すると、それに伴って別の部品をどう調整しなければならないか、などの技術的・専門的知識を、ユーザーが持っていなければならない。

また、人間が思考し、個々の操作を行うためには時間が必要である。従って、これらの作業を一瞬で行うことは、人間であるユーザーには無理がある。

そのため、製品システムが複雑化すればするほど、ユーザーがシステム全体を細部まで操作することは困難になる。システムの複雑化に伴って、技術的・専門的知識の必要量が増加し、一つの部品の調整に伴う連携作業も増え、それら全体を操作するための作業量や作業時間が増えるからである。

3.1.2 正確な動作と硬直性

他方、製品システムを構成する各部品は、機械や電子情報によって制御されるから、命令に対して正確に対応し誤りが少ない。また、電気反応や電子情報で制御されれば、操作に必要な時間も一瞬ですむ。それだけ、製品システム全体の信頼性が高まることになる。

また、部品同士の相互連関をあらかじめ計算して、ユーザーが特定部品の動作を変更すれば、関連部品の動作を機械的・自動的に調整する仕組みを内蔵させておくことも可能である。例えば、エンジンの回転数に比例してガソリン供給を自動的に変化させるというように。

しかし、それらが可能なのは、あらかじめ用意された命令に対する反応（マニュアル化された動作）や、あらかじめ準備された部品間の動作調整（コーディネーション）を通じて行うしかない。このようなマニュアル化や自動化は、同時にシステムを硬直化させかねない。エンジン回転数とガソリン供給を自動連動にしたために、山道でエンジンの回転数が落ちたときまで、ガソリン供給が減ってしまっては元も子もない。

また、あまり頻繁に発生するとは考えられない事態の対処を、機械や電子制御に任せると、必要な要素部品の数を増やし、システムを無用に複雑化させかねない。

機械や電子制御に依存しすぎることもまた、システムの弾力性を失わせ、コストを高めてしまうのである。

3.1.3 予想しなかった事態への対応

あらかじめ予想できなかったり、予想しなかった事態が起きた場合、どんな高度な製品シ

ステムも、ユーザーの適切な対処なしにはうまく機能しない。めったに起こらないために、機械や電子制御を準備しておかなかった場合も、同じである。このような場合、その場の状況を見渡して、経験や直感を生かした自発的（spontaneous）な判断をユーザーに求めることが必要になる。

過度に機械的・電子的制御に依存させたり、むやみに厳格な部品間連動の仕組みを作るより、ある程度弾力的な対応の可能性を残しておき、予想しなかった事態には、ユーザーである人間に大局的な判断をさせ、経験や学習に裏打ちされた自発的な対応に任せたほうがよいということでもある。

3.1.4 システムの複雑化と部品の相互干渉

さらに重要なことは、システムが複雑化するにつれて、システムの部品要素が思わぬ形で相互干渉し、そのためにシステムの適切な作動が損なわれることが増える。システムが複雑化すればするほど、どこまでを機械や電子制御に任せ、どこからをユーザーの判断に任せるかの区別が重要になるのである。

3.2 文脈依存型な認知

では、機械や電子制御に細部を任せ、大局的な判断だけをユーザーに任せればよいだろうか。おそらく答えはノーである。

正確ではあるが硬直的な機械や電子制御と、経験と反復学習で学んだ認識・対処能力を持つユーザー（人間）との間の連携作業には、それなりのインターフェースが必要になるからである。そこで次に、人間であるユーザーの認識・対処能力は、どんな特徴を持っているかを検討しよう。

3.2.1 文脈依存型な認知

外部環境には大きな不確実性が付きまとうだけでなく、それ自体が頻繁に変動する。そのため、ユーザーがシステムから最適な結果を得るためには、外部環境が大きく変化しない、短時間のうちにシステム操作を終了させなくてはならない。他方、外部環境を認知しシステム操作につなげるためには、大量の情報を収集し、適切に処理し、必要な動作を各部品に命令・伝達する必要がある。

3.2.2 人間の認識パターン

もし人間の認識能力が、外部環境を物理的・論理的に整理するようにできていれば、人間

の認識をそのまま情報として流すことで、機械が理解できるだろう。しかし、人間が外部環境を認識する仕方は、経験や学習によって学んだ帰納的な整理による。人間の認識や思考は、いわば**文脈依存（context dependent）型**であると共に、形式知だけでなく**暗黙知（tacit knowledge）**にも依存する。

例えば、自動車の運転を例にとろう。ユーザーにとって、自動車という製品システムを的確に操作するためには、現在の走行速度、エンジンの回転数、ラジェーターの温度、道路の傾斜、路面の滑りやすさ、道路の照明、ヘッドランプの光度など、さまざまな外部（およびシステム内部の）情報を知らなければならない。しかしユーザーである人間は、これらの情報群を、個々の変数に分解して論理的・物理的に認識するわけではない。エンジン回転 1,870*rpm*、ラジェーター温度 95.8℃、ヘッドライト光度 12,450 カンデラなどと認識しながら、自動車を運転する人がいるだろうか。

3.2.3 不確実性のパティション

むしろ人間が認識するのは、「今は夜である（従って、路面は暗く、道路の照明とヘッドランプがたよりである）」、「雨が降っている」、「下り坂にしてはスピードが出ている」といった、パターンによる認識である。

このことを以下では、外部環境という不確実性を、

- 個々の要素 $\omega \in \Omega$ として理解するのではなく、

「夜である」という属性を持った ω の部分集合 P_{night}、「雨が降っている」という属性を持った ω の部分集合 P_{rain} というように、

- 特定のパターン（これを、以下では「パティション」と呼ぶ）ごとに Ω の部分集合に含まれる事態が発生したと理解する、

と考えよう。

言い換えれば、人間は、

- 個々の不確実性それ自体（つまり、$\omega \in \Omega$）を認識するのではなく、
- いま直面している不確実性（$\omega \in \Omega$）が、
  - 不確実性全体（Ω）を分割したある**パティション（partition）**（$\mathcal{P}$）を構成する一つの（Ω の）部分集合（$P \in \mathcal{P}$）に含まれる、

— つまり、$\omega \in P \in \mathcal{P}$ である、

ことを認識するのだ
と考える。

人間が認識するパティションは一つではない。従って、人間が認識するのは、いま直面している不確実性は、それぞれのパティションの中のどの部分集合に属しているかという形で認識すると考える。具体的な例を挙げるのが一番分かりやすいだろう。

例えば、いま人間が認識するパティションを、

$$\mathcal{P}_{weather} = \{P_{rain}, P_{cloudy}, P_{sunny}\}$$
$$\mathcal{P}_{time} = \{P_{morning}, P_{afternoon}, P_{evening}, P_{night}\}$$
$$\mathcal{P}_{slope} = \{P_{up}, P_{flat}, P_{down}\}$$

の三つだとしてみよう。このとき、人間がある $\omega \in \Omega$ に直面したとしても、それは例えば、

$$\omega \in P_{rain} \cap P_{evening} \cap P_{down}$$

として、つまり「雨が降っている夕方に、下り坂を」運転しているのだ、と認識すると考える。

このような考え方は決して目新しいものではない。多くの識者が、人間の行動は「文脈依存型」だと指摘してきた。パティション $\mathcal{P}$ とパティションを構成する Ω の部分集合 $P \in \mathcal{P}$ とは、まさに、「天気が晴れである」とか「道が急カーブしている」といった**文脈（context）**を意味している。

3.2.4 文脈依存型な意思決定：形式知と暗黙知

ところで、「人間が外部環境を認識するのは文脈依存型である」ということだけならば、人間が機械を操作することに対する本質的な障害にはならないだろう。なぜなら、人間の認識パターン（あるパティションに属するある Ω の部分集合に直面している）自体を、機械にも認識させればよいからである。言い換えれば、各パティション $\mathcal{P}$ の特定の集合 $P \in \mathcal{P}$ に直面する場合には、機械が特定の動作 $x(P)$ をとる、という動作関数を指定してやればよいからである。

しかし、人間の認識する文脈どおりに、機械にも認識させることは困難な場合が多い。同様に、人間の認識パターンに従いつつ、人間を介さずに機械を操作することも困難な場合が多い。人間の認識パターンは、しばしば**形式知（formal knowledge）**ではなく、**暗黙知（tacit**

knowledge）によって規定されているからである。

暗黙知とは、3.2.1 節の定義に従えば、「機械が認識できるような論理的・抽象的な形では定義できない」Ωのパティション P に他ならない。人間の認識を規定するパティションは、多くの場合、経験や反復学習を通じて形成された、帰納的に定義されたパティションだからである。

3.3 3 節のまとめ

以上をまとめると、次のような事情が明らかになる。

- ユーザーは、人間であるという本質のために、暗黙知で定義される場合を含めて、文脈依存型でしか事態を認識し、システムを操作できない。
- しかし、機械的や電子的な部品から構成されるシステムを機能させるためには、システムが理解できる言語で情報を伝達・命令しなければならない。
- したがって、製品システムは、形式知だけでなく、「暗黙知でも定義される文脈依存型の（人間の）言語」（human language）を、「各構成要素や部品がとるべき動作を指定する（機械の）言語」（machine language）へと変換する機能を内包せねばならない。
- そのためには、文脈依存型言語から機械語へと切り替わるポイントを、システムという概念の境界として考えることができるのではないだろうか。それが、動作関数が、ユーザーが操作する**外部操作関数**と、ユーザーの命令をシステム内部で自動的に処理する手順である**内部動作関数**に分けられる理由ではないか。
- また、ユーザーが認識・理解する各文脈を機械語に翻訳するためには、文脈を機械であるシステムが理解できるように、システムの機能と定義することが必要になるのではないだろうか。

以下では、節を改めて、これらをシステムの「動作要因の標準化」という視点から検討したい。

4 標準化と機能

4.1 製品の開発・設計

新たな製品の開発作業を考えてみよう。この作業では、新たな要素部品を組み入れることで、製品の性能を向上させることも重要である。しかし本稿の視点からいってもっと重要なのは、

- ユーザーが操作しやすい製品の仕組みを作ること

である。

そのためにまず必要なのは、ユーザーが製品を操作する場合の認識パターン、つまり「ユーザーがどのように外部環境を仕分けしているか」を、製品の動作と関連付けて理解することである。製品開発や作業システムを設計する際に、しばしば「作業の標準化」が大事だと言われる。以下では、この作業を「動作要因の標準化」と言う角度から検討したい。

4.2 動作要因の標準化

すでに述べたように、ユーザーが外部環境を仕分ける仕方（外部環境Ωのパティションの仕方）は人間の言語に依存しており、機械語に基づいた論理的・抽象的な外部環境の仕分け方とは異なっている。しかし、製品を設計するエンジニアが持っている知識の多くは、部品の物理的・電子的特性とその規則だから、最初に作られる製品の多くは機械語型の仕分けを前提に作られている。

4.2.1 ユーザーと製品の使い方

それにもかかわらず、作られた製品を使うのは、機械語型の製品を人間に使いやすい製品に改善してゆくためには、ユーザーが既存の製品をどう使いこなしているかを知ることが重要である。2.2.4 におけるわれわれの定義を使えば、ユーザー自身の**外部操作**と製品に組み込まれた**内部動作**の双方を含む「動作関数」ϕを通じて、ユーザーである人間が、外部環境に応じて製品をどう使いこなしているかを調べることである。これらの事例を分析することを通じて、人間が操作しやすく、正確でスムースな内部動作を行える仕組みを作り出すことが可能になる。

4.2.2 内部動作関数からの機能概念の導出

例えば、ブレーキという概念が存在していない場合を考えてみよう。この場合でも、自動車を操作する際、「高速で急カーブを回る」、「道路上に障害物がある」、「下り坂で停車しようとする」といった外部環境では、どんな製品でもどんなユーザーでも、急激にエンジン回転数を減らそうとしていることが分かるだろう。このような事例を集めてゆけば、車を安全に運転するためには、「急激に自動車を減速する」という**機能（function）**が運転には必要だということがわかる。

他方、どんな場合にこの「急激に減速するか」を機械語で定義することは、実は困難だと

いうこともわかる。時速 60 キロで半径 50 メートルのカーブを曲がる場合と、時速 50 キロで半径 60 メートルのカーブを曲がる場合の違いや、それが雨が降っているかどうかでどう変わるか、昼か夜かでどう変わるかなどを、機械語で叙述することは困難だからである。

そうならば、自動車にはブレーキという、ユーザーが**直接命令できる仕組み（ユーザー・インターフェース）**をつけることが有効だということになる。こう定義した「ブレーキ」は、抽象的な「減速」という機能を発揮するためのユーザー・インターフェースであり、急激な減速がいわゆる「空気ブレーキ」によるものか「ディスク・ブレーキ」かによるものかには依存しないし、それ自身がフットペダルの形状をしている必要もない。

逆に、ブレーキというユーザー・インターフェースが作られることでユーザーの自動車操作が容易になるのは、「高速で急カーブを回る」という文脈では、「減速」という機能を発揮すればよい、そのためには「ブレーキというユーザー・インターフェースを操作すればよい」という、文脈依存型の操作が可能になるからである。

4.2.3 参考事例と動作要因

誤解を避けるために、上の小節で述べたことを、少し抽象的に定義しておこう。

説明を容易にするために、同じような文脈の中で使われている既存の製品がいくつかあり、その使われ方を参考にしながら、新たな製品（システム）を開発しようとしている場合を考える。既存の製品は n 種類あり、それらを $(\pi^1,...,\pi^n)$ としよう。また、各製品 π^j の使われ方（usage、あるいは動作関数）はユーザー別に m_j 種類あり、それらを $(\phi^{j1},...,\phi^{jm_j})$ としよう。したがって、新たな製品開発のために使える「参考事例」は $M=\Sigma_{j=1}^{n}m_j$ 種類ある。

次に、既存の製品 $\pi^j=(N^j,\boldsymbol{X}^j)$ を固定して、その動作関数のひとつ $\phi^{jk}:\Omega\to X^j$（つまり、ユーザー k の製品 π^j の使い方）と、この製品の部品のインデックスのうちのある部分集合 $N_f\subset N$ を取り出してみる。この動作関数 ϕ^{jk} の下で、その部分集合 N_f に属する部品状態のプロフィールが特定の値 $\bar{\boldsymbol{x}}_f=(x_l^j)_{l\in N_f}$ をとる外部環境の集合（Ω の部分集合）を定義することを考えよう。この集合を以下、$\bar{x}_f$ の**動作要因**と呼ぶ。

ユーザー k と彼の π^j の使い方 ϕ^{jk} からなる事例 (j,k) で、π^j が状態 $\bar{x}_f$ を取る動作要因は、

$$P_f(\bar{\boldsymbol{x}}_f\,|\,\pi^j,\phi^{jk})=\{\omega\in\Omega\,|\,\phi_l^{jk}(\omega)=\bar{x}_l,\forall l\in N_f\}$$

として定義される。前小節の例を使えば、$\bar{\boldsymbol{x}}_f$ とは、製品 π^j の部分集合 N_f によって製品 π^j が急激な減速操作を行っている状態であり、そのような操作をユーザー k が行う動作要因として、「高速で急カーブを曲がる」、「道路上に障害物がある」などのパターンに合致する

のが、不確実性の集合 $P_f(\bar{\boldsymbol{x}}_f|\pi^j,\phi^{jk})$ として同定されるというわけである。

4.3 事例の収集と機能の導出

もちろん、動作要因は、特定のユーザーk の癖や行動パターンにも依存する。従って、同じ製品でも、ユーザーが異なれば、同じ動作をとる状況は異なる。とはいえ、異なるユーザー間でも動作要因が共通で、同じ動作をとる状況が同じである場合も多い。もしそうなら、ユーザーを越えて、ある製品の動作 $\bar{\boldsymbol{x}}_f$ の共通した動作要因が、例えば、

$$P_f(\bar{x}_f|\pi^i)=\cap_k P_f(\bar{x}_f|\pi^j,\phi^{jk})$$

として同定されるだろう。この場合、動作要因 $P_f(\bar{\boldsymbol{x}}_f|\pi^i)$ に対応する製品の具体的な動作 $\bar{\boldsymbol{x}}_f$ には、人間が製品を操作するために必要な、普遍的な性質が存在すると考えるべきだろう。

また、製品が異なれば、（空気ブレーキの空気圧が高まるのと、ディスクブレーキのディスクが締まるというように、）動作が異なっていても動作の動作要因は共通しているかもしれない。この場合、二つの製品 $\pi^j=(N^j,\boldsymbol{X}^j)$ と $\pi^{j'}=(N^{j'},\boldsymbol{X}^{j'})$ について、それぞれ適当な部品の組み合わせ N_{fj} と $N_{fj'}$ があるとする。もし、その部品の集合が取りうる状態の組み合わせについて、$P_{fj}(\bar{\boldsymbol{x}}_{fj}|\pi^j)=P_{fj'}(\bar{\boldsymbol{x}}_{fj'}|\pi^{j'})$ ならば、二つの製品 π^j と $\pi^{j'}$ に共通した動作要因があるといえる。

これは、個々の製品を越えて、人間が「この種の製品」に共通して持ってほしいと思う普遍的な動作である。このようにして、**製品の機能（function）**という概念が作られる[1]。

特に、製品がうまく機能しなかった事例を集め、その場合にユーザーがどんな動作を行ったのか、それらに共通な動作要因はなにかを検討することも有益だろう。例えば、自動車事故が起こった事例の中から、「エンジンの回転数を落とす」という動作の共通動作要因として、「下り坂のカーブ」、「高速で運転中」、「人が飛び出した」などの外部環境があげられるなら、「急減速する」という機能が製品にとって重要だということが学ばれるだろうように。

[1] いうまでもなく、機能の概念が生まれるのは、このような事例からの帰納によるだけではない。技術者が製品を作るときに、演繹的に機能を定義する場合もあるだろうし「こういうときにこういう形で動く製品を作って欲しい」というユーザーの希望から機能が作られることもあるだろう。それにもかかわらず、ユーザーの経験から帰納的に機能が定義されることを強調したのは、そうすることで、文脈型で行動するユーザーが、機械語で作動する製品を操作する際の問題点がより明確になると考えたからである。

4.4 機能概念の転換：動作要因から製品性能規定要因へ

4.4.1 動作要因と製品性能規定要因

このように事例の標準化を通じて機能が定義されるならば、それは次の二つの性質を満たすことになる。

- 機能とは、外部環境（ユーザーが直面する不確実性）Ωを、動作要因に従って分割することである。例えば、「ブレーキをかける」という機能 f とは、外部環境を「急にストップする必要がある」、「スピードダウンしたい」、「どちらも必要ない」といった動作要因に分割すること、つまりΩを、

$$\digamma_f = \{F_{sudden}, F_{slowdown}, F_{no}\}$$

とパティションすることである。なお、このパティションの一つの要素 F_i が前節で定義した $P_f(\bar{x}_f | \pi^j)$ に相当すると考えてよい。

- 他方、機能とは単に動作要因を同定することだけではなく、それに伴って製品が実現すべきパフォーマンスも指定している。つまり、上の例で言えば、F_{sudden} という外部環境が起こったときに、「自動車をできるだけ早く、安定な姿勢のまま停止状態に移行させる」ことを要求している。それがスムースに早くできればできるほど、製品全体のパフォーマンス（ユーザーにとっての使い勝手）が高まる、という認識も、機能の定義には含まれている。

4.4.2 機能の標準化

こうして作られた機能のインデックスの集合を、$\mathbb{F}$で表すことにしよう。ユーザーは、各機能 $f \in \mathbb{F}$について、それを「どのような状態、どのような水準、どのような質で実行してほしい」、という命令 $F \in \digamma_f$ をシステムに伝えたい。この、各機能 $f \in \mathbb{F}$別に、製品がどのようなパフォーマンス（機能の作動状態、作動水準、作動品質）を示しているかを表すパラメータを、以下、α_f としよう。上記で述べた参考事例から、「どんな製品（π^j）がどんな場合（ω）にどんな使われ方（ϕ^{jk}）をし、その結果どんな状態をとった（$\boldsymbol{x}_f$）場合、ユーザーがどんな利得を得たかということから、$\alpha_f \in A_f$ が機能 $f \in \mathbb{F}$の水準として決まることになる。

製品システムの機能の標準化とは、機能のインデックス（種別）$\mathbb{F}$と各機能の性能を表す

パラメータ $f \in \mathbb{F}$ の内容（製品が持つ当該機能の性能についての具体的内容）F_f を明確に定義することに他ならない。その結果、システム側は、選択された機能を、システムの**機能別動作（function-based action）**、$\alpha = (\alpha_f)_{f \in \mathbb{F}} \in \times_{f \in \mathbb{F}} A_f$ として実行することになる。

4.4.3 機能別動作の評価

この結果、ユーザーにとって、ある製品システムの動作とは、そのシステムが「さまざまな機能を規定する動作要因が発生したときに、具体的にどのような動作を選択するか」として意識される。例えばオーディオシステムの場合、外部環境を区分けした動作要因は、

$$\mathcal{F}_{media} = \{F_{TV}, F_{DVD}, F_{CD}\}$$
$$\mathcal{F}_{action} = \{F_{play}, F_{record}, F_{erase}\}$$
$$\mathcal{F}_{volume} = \{F_{loud}, F_{medium}, F_{quiet}\}$$

などである。

ユーザーはその中から現状を、例えば $F_{DVD} \cap F_{play} \cap F_{quiet}$、つまり、「DVD」を「低音で」、「再生する」のが望ましいといった外部要因として認識する。問題は、それをこのようなプロセスを経て、製品に対する評価をより普遍的な尺度から見ることができるようになる。つまり、われわれが出発したのは、特定の製品 π^j が特定の不確実性 $\omega \in \Omega$ の下で特定の動作状態 $\boldsymbol{x} \in \boldsymbol{X}$ をとることを、特定のユーザー k がどう評価するかを表す、主観的な評価関数 $\upsilon^k(\boldsymbol{x}, \omega | \pi^j)$ だった。しかし、機能のパフォーマンスベクトル $\alpha = (\alpha_f)_{f \in \mathbb{F}}$ を媒介することで、製品システムの評価は、より普遍性を持つ $\hat{\upsilon}(\alpha, \omega)$ という形であらわされることになる。

4.5 ユーザーの操作関数

つまるところ、文脈型で行動するユーザーが、与えられた外部環境を基に製品（システム）を操作するのは、製品の内部状態を直接操作するのではなく、「製品の諸機能を特定の水準（状態）に制御する」、という命令を製品に伝えているのだと考えられる。

製品を操作しているユーザーにとって、文脈型の認識とは、与えられた外部環境が $\times_i \mathcal{P}_i$ のどれに対応しているかという認識である。上で述べた外部環境（不確実性）の集合 Ω とは、ユーザーにとっては実は $\times_i \mathcal{P}_i$ として認識されており、個々の外部環境 ω は、文脈の中でどこに位置するのか $\omega \in \times_i \mathcal{P}_i$ という形で認識される。他方、製品に与える命令は、製品がどんな機能（$f \in \mathbb{F}$）を、どのような水準や状態に（どんな α_f に）保ちたいかという情報である。

従って、ユーザーが製品を操作する場合、その関係は、次のように定義される**ユーザー操作関数（user operation function）**で表されることになる。

$$\psi : \times_i \mathcal{P}_i \to \times_{f \in \mathbb{F}} A_f$$

つまり、ユーザーは与えられた外部環境の文脈が認識$\omega \in \times_i \mathcal{P}_i$を基にして、ユーザーが発揮すべき機能の程度$\psi(\omega) \in \times_{f \in \mathbb{F}} A_f$を製品に命令するのである。

4.6 4節のまとめ

以上を、4節で述べたことを簡単にまとめると、次のようになる。

- ユーザーは製品を、文脈依存型でしか製品を操作できない。ユーザーはまた、製品の操作の多くを、経験によって帰納的に学習する。
- 従って、ユーザーが操作しやすいような製品を設計・開発するためには、ユーザーが共通して同じ操作を行う環境がどんな場合であるかを通じて、製品が持つべき機能を同定することが望ましい。
- その結果、製品の性能は、内包する機械的・電子的な仕組みが、それらの機能をどれだけ発揮させられるかという点で評価できることになる。
- このため、機能が発揮すべき製品動作が決まる。その意味で普通、機能は外部環境ではなく、製品動作を意味していると認識されることになる。
- この製品動作に反映された機能という概念（製品を文脈的に操作するために言語）を介して、ユーザーの命令が、機械・電子部品に伝達・処理されるのである。

5 アーキテクチャ

5.1 ユーザーと機械のインターフェース

ところで、ユーザーは文脈型に行動し、製品システムを機能（これも、文脈型の認識パターンを製品種類に応じて定義しなおしたものでしかない）を介して操作しようとする。しかし、製品システムは論理的・物理的・電子的に制御されるから、機能という概念を通じた（文脈型言語で定義された）命令はそのままでは理解できない。

従って、製品システムは、入力される機能別の命令を、製品がそれに対応する適切な状態に移行できるよう、物理的・電子的に定義される機械語に翻訳する仕組みを持っていなければならない。いわば、人間であるユーザーと機械である製品システムの間の**インターフェース**が必要になる。

以下、このインターフェースを鍵概念としながら、製品システムのあり方（アーキテクチャ）を検討しよう。

5.2 機械語

5.2.1 コード

製品システムの側から見た場合、製品が適切な形で機能し、各部分部分の動きを適切にコーディネートするためには、物理的に動作する部分には（歯車の回転や電力といった）物理的な形で、電子的に動作する部分には（デジタル・コードなどでの）電子情報によって、命令が伝達され、お互いの動作自体に関する情報交換がなされなければならない。これらの命令や情報は、製品が正確に動作するよう、論理的で紛れのない形で定義されていなければならい。また、命令や情報はあらかじめ準備されているプログラムによって処理されるから、プログラムが誤りなく動作するよう、いったん定義されたらその意味が変更されないことも重要である。

これらの命令や情報の記述形態は、普通、**プログラム（program）**と**コード（code）**と呼ばれる。以下では、製品システムが内部処理のために、物理的・電子的に命令や情報を記述するコード体系を c で表そう。いったん、命令や情報が c で記述されれば、ユーザーとは独立に、部品システムが情報伝達や部品間のコーディネーションを、内部で物理的・電子的に処理できるというわけである。

5.2.2 プログラム言語

パソコンなどを考える場合、実はこのコード体系と、それを使ったプログラム言語という、大別すると二段階の体系があると考えられる。

下級言語　一つは、**下級言語（low level language）**と呼ばれる一連のバイト列で定義された（c がバイト列の体系として定義される）機械語であり、それを使って記述される命令は、CPU 設計に関わる専門メーカーの専門技術者や高度のハッカーだけにしか理解できないような言語である。機械語を使うことでシステムを直接コンピュータ制御することが可能になり、その結果、複雑化したシステム内部のコーディネーションをコンピュータに任せて自動化することが可能になる。

しかし、一方で機械語によるプログラムは特定の製品・機械への依存度が高く、ある機械語に習熟していたとしても他の製品・機械の機械語を操れることにはならない。結果として、人間にとっては同じような文脈の中で利用する製品同士であっても、構成部品が異なることでまったく異なる機械語を習得しなければならないなどの問題が生じることになる。

上級言語　いまひとつは、**上級言語（high level language）**と呼ばれるCやフォートランなどのコンパイラー型の言語であり、普通のユーザーには利用不可能でも、ある程度の専門知識を持ったエンジニアなら比較的簡単に理解可能な言語である。使われるコードは、アルファベットを並べた命令を基礎としており、プログラムによって指定される機能が機械語よりも抽象化されている。従って、人間が普通使っている言語（単語）との類推で、習熟すればその意味を理解できる。

この言語を理解できる技術者さえいれば、cで書かれた命令を入力として使う個別部品を開発することが、普通の家電メーカーなどでも可能になる。また、プログラムによって指定される機能が抽象化さていることから、個別の製品・機械に依存する機械語への翻訳の部分をコンパイラにゆだねることによって、個別の製品・機械の変化に対しても開発者が容易に対応できる。

これから述べる「システムの翻訳・統御関数」という立場からパソコンを眺めれば、CPU・プリンタ・グラフィックなど個別のデバイスを動作させるためには、前者の機械語による命令が必要となる。そのために、ユーザーから入力を機械語へ書き換えていくための仕組みがデバイスドライバであり、それを統括して管理するWindowsなどのOSということになる。

本稿では記述を明快にするために、前者には立ち入らない。従って、本稿で定義するコード体系cとは、コンパイラー型言語であり、本来の機械語のことではない。

5.3 人間言語の機械語への翻訳

5.3.1 システムの翻訳・統御関数

他方、ユーザーが入力した命令は、相変わらず文脈依存型の、機能別の命令という形で行われる。従って、システムが適切な形で機能するためには、

- ユーザーの命令を、システム側が理解可能なコードcに翻訳し、物理的・電子的に理解可能な命令に置き換えること[2]、
- 逆に、システム側からフィードバックされてきた（どのように命令を実行したか、その過程でどんな問題が発生した）などの情報を、
- システムの各部分からフィードバックされてきた情報を（当該部分が適切に動作するためには、システムの他の部分にどのような動作をしてもらうことが必要か、などの

[2] そのためには、機能別に入力されるユーザーの命令を、システムの機械的・電子的動作命令に置き換える仕組みとしての**ユーザー・インターフェース**も必要である。例えば、自動車のステアリング・ウィール、ブレーキ、アクセルなど、あるいはパソコンのキーボードやマウス、GUIなどである。しかし、これも本稿の議論にとって本質的ではないので、以下では捨象して考える。

情報を)、それが全体のコーディネーションに合致しているかどうかを判断した上で、システムの他の部分に伝達することが必要になる。

以下ではこの仕組みを、**システムの翻訳・統御関数（translation and control function）**τとして、次のように定義しよう。

$$\tau : \times_{f \in \mathbb{F}} A_f \to \mathcal{C}$$

つまり、ユーザーが入力した機能別の命令$\alpha \in \times_{f \in \mathbb{F}} A_f \equiv A_{\mathbb{F}}$を、製品システム側が理解可能な言語$\tau(\alpha) \in \mathcal{C}$に置き換える仕組みである。

5.3.2 ユーザーの製品操作

この、システムの翻訳・統御関数（τ）を所与として、ユーザーが自分の操作関数（$\psi : \times_i \mathcal{P}_i \to A_{\mathbb{F}}$）をえらべば、ユーザーがこのシステムを操作することは、

$$\tau \circ \psi = \times_i \mathcal{P}_i \to \mathcal{C}$$

という合成写像として与えられることになる。つまり、ユーザーの文脈型認識$\omega \in \times_i \mathcal{P}_i$を所与として、ユーザーがシステムに命令$\alpha = \psi(\omega)$を入力し、それを翻訳・統御システムが$\tau(\alpha) = \tau(\psi(\omega)) \in \mathcal{C}$という機械語に翻訳し、それをシステム内部に伝達するというわけである。

5.4 部品モジュール

翻訳・統御関数（τ）が与えられると、製品システム全体は、二つの部分に切り分けられる。

- 人間であり機械語（C）の理解能力を全く持たない「ユーザー」と、
- 文脈型言語を理解できず、機械語しか理解できないが、それを受けて、2.2.4 で定義した**内部動作関数**として機能する「部品モジュール」

である。しかも、このように分割することで、両者の一層高度な協業が可能になる。

5.4.1 部品の機能とその発現能力

具体的には、次のような定義を考えることができる。いま、製品システムを構成している

部品モジュールの集合を $\mathbb{M}$ で表そう[3]。

ここで、

- 個々の一つの部品モジュール（$m \in \mathbb{M}$）とは、
  - ― 当該部品が果たすべき機能の集合 $\mathbb{F}_m$ と、
  - ― 当該部品の機能の発現能力である内部動作関数

$$\mu_m : \mathcal{C} \to \times_{f \in \mathbb{F}_m} A_f \equiv A_{\mathbb{F}_m}$$

の組（ペア）として定義される。

5.4.2 ユーザーと部品システム

これらの部品モジュール全体が、システム全体の内部動作関数 $\mu \equiv (\mu_m)_{m \in \mathbb{M}}$ に対応する。言い換えると、われわれがイメージする**製品システム**とは、翻訳・統御関数と部品モジュール全体とのペアである（τ, μ）に他ならない。

従って、システムは次のように作動する。

- ユーザーが外部環境 $\omega \in \times_i \mathcal{P}_i = \Omega$ を文脈的に認識する。
- ユーザーはそれに基づいて、製品システムに発揮してほしい機能状態 $\alpha = \psi(\omega) \in \times_{f \in \mathbb{F}} A_f \equiv A_{\mathbb{F}}$ を機械システムに入力する。
- 機械システムは、翻訳・統御システムがユーザーからの入力（α）を、$\tau(\alpha) \in \mathcal{C}$ と機械語のコード体系 $\mathcal{C}$ に変換し、各モジュールに入力する。
- 各部品モジュール（m）は、翻訳・統御システムから受け取った機械語のコード入力 $c = \tau(\alpha) \in \mathcal{C}$ を見て、自分が設計された手順に従って、担当する機能 $\mathbb{F}_m$ の発現状態 $\overline{\alpha}_f \in A_{\mathbb{F}_m}$ を実現する。
- 部品モジュール全体は、機能 $\overline{\alpha} \in A_{\mathbb{F}}$ を発現するから、それが自分の期待した機能 $\alpha = \psi(\omega)$ とどの程度似ていてどの程度異なるかを見て、必要に応じてユーザーは命令をシステムに入力しなおす。

このような操作が可能になることで、ユーザーは部品モジュールの内部が、どのような機械的・電子的な仕組みで作動するかという知識を全く持たなくても、製品システムを操作す

[3] ただしここでいう部品モジュールとは、モジュール型アーキテクチャの構成部品であることを**意味するわけではない**。ここでの部品モジュールとは、単に、製品システムを構成する一つのコンポーネント、あるいは、部品の集まりではあるが、製品全体ではない部品の集まり、という意味である。

ることが可能になる。いわば、部品モジュールの内部構造が**カプセル化（encapsulate）**されることで、部品モジュールの作動形態の知識さえあればそれを使いこなすことが可能になるのである。

5.5 製品システムとそのアーキテクチャ

5.5.1 製品システムの構造

こう考えると、製品システムの構造は、

- 製品が果たすべき機能（$f \in \mathbb{F}$）と、各機能が果たすべき性能（$A_{\mathbb{F}} = \times_{f \in \mathbb{F}} A_f$）を明確にする、
- ユーザーが入力する機能を機械語に翻訳する仕組みを定める、
  — 機械語が記述されるコード体系（$\mathcal{C}$）を定める、
  — コードと機能（の内容）との間の対応関係（$\tau : A_{\mathbb{F}} \to \mathcal{C}$）を定める、
- 製品システムの機械的内部構造、つまり、機械語によって製品システムが動作するための部品モジュールの集合（$\mathbb{M}$）を定める。各モジュール（$m \in \mathbb{M}$）は、
  — 当該部品モジュールが発揮すべき機能 $\mathbb{F}_m$ と、
  — 機械語の入力（$c \in \mathcal{C}$）に応じて、当該部品モジュールがどんな機能を発揮するよう物理的・電子的に設計されているかという仕組み（$\mu_m(c) \in A_{\mathbb{F}_m}$）、

という三つの部分から構成されることになる。

5.5.2 製品アーキテクチャ

われわれは、こう定義された製品システムのうち、

- 製品が果たす機能と各機能が果たすべき性能、
- ユーザーの機能別入力命令を機械語に翻訳する、翻訳統御関数

を、つまり（$\mathbb{F}, A_{\mathbb{F}}, \tau$）というトリプルを、**製品アーキテクチャ**と呼ぶ[4] ことにする。

以下、節を変えて、モジュール型アーキテクチャの特徴を、このように定義した「製品アーキテクチャ」概念を通じて検討したい。

[4] 実はこのトリプルを製品アーキテクチャと呼ぶことは不適切かもしれない。なぜなら、「次の節で述べるような形でこのトリプルを扱う」という仕組み（藤本のいう設計思想）こそが、モジュール型アーキテクチャという製品アーキテクチャだ、とも考えられることである。この点では、本論文の最終稿を作成する段階で、再検討してみたい。

6 モジュール型アーキテクチャ

6.1 スタンダードとしての製品アーキテクチャ

IBMPC を始めとする、いわゆる**モジュール型アーキテクチャ**の第一の特徴は、上に定義したような意味での製品アーキテクチャが、**標準化（standardize）**されていることである。

つまり、製品システムとしての IBMPC が発揮すべき機能があらかじめ決まっており、諸機能を発現するための機械的コード（言語）もあらかじめ決まっているという点である。これが IBMPC といわれる**製品システム群**の特徴であり、アップルの PC で動くパソコンや、従来型のワーク・ステーションやスーパー・コンピューターとの違いである。

製品システムとしては細かい点で違いがあるにせよ、製品アーキテクチャとしての性質が同じであること、つまり製品アーキテクチャが標準化されているからこそ、

- ユーザーは、同じことをしたければ同じ命令を発すればよく[5]、
- 同一のアーキテクチャで機能する限り、異なる部品モジュールも、ユーザーからの同じ命令に対して同じ（あるいは良く似た）機能を発現するから、それらの部品モジュールはお互いに代替可能であり、
- 異なる部品モジュールに組み替えることで、製品システムの多様化が容易になる。
- どんな機械語がどんな機能に対応しているのかが明らかだから、他の機能を担当する部品モジュールの開発・製作とは独立に、個々の部品モジュールを開発・製作することが可能になる。

という利点を持つことになる.

6.2 製品アーキテクチャの公開

モジュール型アーキテクチャの第二の特徴は、製品アーキテクチャが一つの企業の内部や専門家の間だけで標準化されているのではなく、広く社会全体に公開されている、**オープン・アーキテクチャ（open architecure）**だという点である。

すなわち、

- 組み替え可能な部品モジュールの開発・生産が誰にでも容易に行え、それだけ部品同

[5] もちろん、ユーザー・インターフェースの違いによって、異なる操作が必要になることはあるかもしれない。しかし、マウス操作の何がキーボード操作の何に対応しているのか、という文脈さえわかれば、ユーザーは比較的容易に製品システム同士を乗り換えることができる。

士の競争が起こり、製品システム全体の質が改善し価格が低下する。

- 製品システムを構成する各機能（各機能群）単位で、企業や技術者ごとに専門知識やノウハウを生かし、地域や国単位の消費者ニーズに特化した、部品モジュールの開発が可能になる。
- 機能を新しい形で組み合わせることで、新しい製品システムや新しい部品モジュールを開発する可能性が社会全体で共有され、それだけそのインセンティブが高まる。

6.3 スタンダードへの一定期間のコミットメント

モジュール型アーキテクチャの三番目の特徴は、いったん決めたアーキテクチャ（$\mathbb{F}, A_{\mathbb{F}}, \tau$）の実装形態が一定期間固定され、そのこと自体があらかじめ周知されることである。いわば、製品アーキテクチャに対する**コミットメント**がなされることになる。

IMBPC でいえば、MSWindows とインテルの CPU の組み合わせは、いつ新しい実装バージョンが公開され発売されるかが、あらかじめ予測可能である。すくなくともそれまでの間、現バージョンと整合的な部品モジュールを作れば、それがパソコン・システムの中で動作することが予測できる。このようなコミットメントがあるからこそ、

- 開発のインセンティブが部外者にも共有され、
- ユーザーも、製品システムの購入に高額の出費をしても、部品モジュールの付け替えでシステムを徐々に改善できると考える、

ことになる。

7 終わりに

従来型の製品システムは、製品が果たす諸機能を媒介として、暗黙知や文脈に依存して行動するユーザーにも操作しやすい仕組み作りが目指された。部品や機能間のコーディネーションは、ユーザーが事後的に行うことが前提とされ、ユーザーのコーディネーションがしやすい製品作りが目指される。そのため製品の開発・設計段階での主な作業は、ユーザーが使いやすい（事後的なコーディネーションをしやすい）部品と機能の組み合わせを、いかにあらかじめすり合わせるか（部品と機能を事前にコーディネートするか）にあったのである。

これに対してモジュール型アーキテクチャは、ユーザーと部品モジュールの間に翻訳機能と部品間コーディネーションを担当する仕組みを挿入することで、ユーザーが担当していた、部品間・機能間のコーディネーションという負担を大きく軽減した。このアーキテクチャで

は、製品が果たす機能と各機能が果たすべき製品動作が明確に定義され、実行すべき動作情報を各部品に伝達し、それを実行するために各部品が他部品にどんな動作を要求するかという情報を事前に決めておくことで、事後的なコーディネーションの負担を軽減することが可能になったのである。

最後にアーキテクチャとは何かということについて補足しておきたい. 再掲になるが藤本（2003）によれば、アーキテクチャとは

- どのようにして製品を構成部品や工程に分割し、
- そこに製品**機能**を配分し、
- それによって必要となる部品間の・工程間のインターフェースをいかに設計・調整するか、

に関する基本的な設計思想である。同様に Ulrich（1995）においても、機能要素と物理的要素の関係とインターフェースの重要性が強調されている。しかし、我々は本稿においてアーキテクチャを、システムを構成する要素間のコーディネーションの仕組として定義し、機能と物理要素の対応関係よりもむしろどのようにシステムを外部環境に合わせるかという調整プロセスの重要性を提起した。このような見方は我々だけの特殊なものというわけではない。Aoki（2001）においては、特に企業組織について市場コーディネーションを代替するという側面に注目し、組織の参加者の「行動をコーディネートするために、組織参加者が彼らのあいだで情報の収集、伝達、利用（意思決定）、蓄積を組織する仕方は、多様でありうる（pp.107）」とし、組織的アーキテクチャがその組織が置かれている環境に応じてさまざまに変わりうるとして議論している。このように、どのようなアーキテクチャを選択するかという問題は、どのようにコーディネーションを行うかという問題と同じなのである。

本稿ではアーキテクチャとは何かを分析するための枠組を提示し、モジュール型アーキテクチャの定義を明確化することで、そこから導かれる性質を述べた。しかし、このような製品アーキテクチャをコーディネーション・システムとして捉え分析する試みはまだ始まったばかりであり、藤本（2005）に述べられているような、製品アーキテクチャと組織アーキテクチャの相性などについてはさらなる研究が必要だろう。

参考文献

Aoki, Masahiko（2001）*Toward a Comparative Institutional Analysis*: The MIT Press.（青木昌彦訳，瀧澤弘和・谷口和弘監訳，『比較制度分析へ向けて』，NTT 出版，2001

年）.

Baldwin, Carliss Y. and Kim B. Clark (2000) *Design Rules*, Vol.1: The MIT Press.Ulrich, K.T.（1995）“The Role of Product Architecture in the Manufacturing Firm”, *Research Policy*, Vol.24, pp. 419−440.

藤本隆宏（2003）『能力構築競争：日本の自動車産業はなぜ強いのか』，中公新書，中央公論社.

――（2005）「アーキテクチャの比較優位に関する一考察」, MMRC Discussion Paper Series J-24，東京大学 21 世紀 COE ものづくり経営研究センター.